Package ‘BB’

October 12, 2022

Version 2019.10-1

Title Solving and Optimizing Large-Scale Nonlinear Systems

Description Barzilai-Borwein spectral methods for solving nonlinear system of equations, and for optimizing nonlinear objective functions subject to simple constraints. A tutorial style introduction to this package is available in a vignette on the CRAN download page or, when the package is loaded in an R session, with vignette(``BB'').

Depends R (>= 2.6.1)

Imports stats, quadprog

Suggests setRNG, survival, Hmisc, numDeriv

BuildVignettes true

LazyLoad yes

ByteCompile yes

License GPL-3

URL http://www.jhsph.edu/agingandhealth/People/Faculty_personal_pages/Varadhan.html

NeedsCompilation no

Author Ravi Varadhan [aut, cph, trl],
Paul Gilbert [aut, cre],
Marcos Raydan [ctb] (with co-authors, wrote original algorithms in fortran. These provided some guidance for implementing R code in the BB package.),
JM Martinez [ctb] (with co-authors, wrote original algorithms in fortran. These provided some guidance for implementing R code in the BB package.),
EG Birgin [ctb] (with co-authors, wrote original algorithms in fortran. These provided some guidance for implementing R code in the BB package.),
W LaCruz [ctb] (with co-authors, wrote original algorithms in fortran. These provided some guidance for implementing R code in the BB package.)

Maintainer Paul Gilbert <pgilbert.ttv9z@ncf.ca>

Repository CRAN

Date/Publication 2019-10-18 04:50:11 UTC

R topics documented:

`BB-package` *Solving and Optimizing Large-Scale Nonlinear Systems*

Description

Non-monotone Barzilai-Borwein spectral methods for the solution and optimization of large-scale nonlinear systems.

Details

A tutorial style introduction to this package is available in a vignette, which can be viewed with vignette("BB").

The main functions in this package are:

```
BBsolve A wrapper function to provide a robust stategy for solving large
        systems of nonlinear equations. It calls dfsane with different
algorithm control settings, until a successfully converged solution
is obtained.

BBoptim A wrapper function to provide a robust stategy for real valued function
        optimization. It calls spg with different algorithm control settings,
until a successfully converged solution is obtained.

dfsane  function for solving large systems of nonlinear equations using a
        derivative-free spectral approach

sane function for solving large systems of nonlinear equations using
        spectral approach
```

```
spg function for spectral projected gradient method for large-scale
        optimization with simple constraints
```

Author(s)

Ravi Varadhan

References

J Barzilai, and JM Borwein (1988), Two-point step size gradient methods, *IMA J Numerical Analysis*, 8, 141-148.

Birgin EG, Martinez JM, and Raydan M (2000): Nonmonotone spectral projected gradient methods on convex sets, *SIAM J Optimization*, 10, 1196-1211.

Birgin EG, Martinez JM, and Raydan M (2001): SPG: software for convex-constrained optimization, *ACM Transactions on Mathematical Software*.

L Grippo, F Lampariello, and S Lucidi (1986), A nonmonotone line search technique for Newton's method, *SIAM J on Numerical Analysis*, 23, 707-716.

W LaCruz, and M Raydan (2003), Nonmonotone spectral methods for large-scale nonlinear systems, *Optimization Methods and Software*, 18, 583-599.

W LaCruz, JM Martinez, and M Raydan (2006), Spectral residual method without gradient information for solving large-scale nonlinear systems of equations, *Mathematics of Computation*, 75, 1429-1448.

M Raydan (1997), Barzilai-Borwein gradient method for large-scale unconstrained minimization problem, *SIAM J of Optimization*, 7, 26-33.

R Varadhan and C Roland (2008), Simple and globally-convergent methods for accelerating the convergence of any EM algorithm, *Scandinavian J Statistics*, doi: 10.1111/j.1467-9469.2007.00585.x.

R Varadhan and PD Gilbert (2009), BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function, *J. Statistical Software*, 32:4, http://www.jstatsoft.org/v32/i04/

BBoptim *Large=Scale Nonlinear Optimization - A Wrapper for spg()*

Description

A strategy using different Barzilai-Borwein steplengths to optimize a nonlinear objective function subject to box constraints.

Usage

```
  BBoptim(par, fn, gr=NULL, method=c(2,3,1), lower=-Inf, upper=Inf,
   project=NULL, projectArgs=NULL,
control=list(), quiet=FALSE, ...)
```

Arguments

`par`	A real vector argument to `fn`, indicating the initial guess for the root of the nonliinear system of equations `fn`.
`fn`	Nonlinear objective function that is to be optimized. A scalar function that takes a real vector as argument and returns a scalar that is the value of the function at that point (see details).
`gr`	The gradient of the objective function `fn` evaluated at the argument. This is a vector-function that takes a real vector as argument and returns a real vector of the same length. It defaults to `NULL`, which means that gradient is evaluated numerically. Computations are dramatically faster in high-dimensional problems when the exact gradient is provided. See *Example*.
`method`	A vector of integers specifying which Barzilai-Borwein steplengths should be used in a consecutive manner. The methods will be used in the order specified.
`upper`	An upper bound for box constraints. See `spg`
`lower`	An lower bound for box constraints. See `spg`
`project`	The projection function that takes a point in R^n and projects it onto a region that defines the constraints of the problem. This is a vector-function that takes a real vector as argument and returns a real vector of the same length. See `spg` for more details.
`projectArgs`	list of arguments to `project`. See `spg()` for more details.
`control`	A list of parameters governing the algorithm behaviour. This list is the same as that for `spg` (excepting the default for `trace`). See `details` for important special features of control parameters.
`quiet`	logical indicating if messages about convergence success or failure should be suppressed
`...`	arguments passed fn (via the optimization algorithm).

Details

This wrapper is especially useful in problems where (`spg` is likely to experience convergence difficulties. When `spg()` fails, i.e. when `convergence > 0` is obtained, a user might attempt various strategies to find a local optimizer. The function `BBoptim` tries the following sequential strategy:

1. Try a different BB steplength. Since the default is `method = 2` for `dfsane`, BBoptim wrapper tries `method = c(2, 3, 1)`.
2. Try a different non-monotonicity parameter `M` for each method, i.e. BBoptim wrapper tries `M = c(50, 10)` for each BB steplength.

The argument `control` defaults to a list with values `maxit = 1500, M = c(50, 10), ftol=1.e-10, gtol = 1e-05, maxfeval = 10000, maximize = FALSE, trace = FALSE, triter = 10, eps = 1e-07, checkGrad=NULL`. It is recommended that `checkGrad` be set to FALSE for high-dimensional problems, after making sure that the gradient is correctly specified. See `spg` for additional details about the default.

If `control` is specified as an argument, only values which are different need to be given in the list. See `spg` for more details.

Value

A list with the same elements as returned by spg. One additional element returned is cpar which contains the control parameter settings used to obtain successful convergence, or to obtain the best solution in case of failure.

References

R Varadhan and PD Gilbert (2009), BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function, *J. Statistical Software*, 32:4, http://www.jstatsoft.org/v32/i04/

See Also

BBsolve, spg, multiStart optim grad

Examples

```
# Use a preset seed so test values are reproducable.
require("setRNG")
old.seed <- setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion",
    seed=1234))

rosbkext <- function(x){
# Extended Rosenbrock function
n <- length(x)
j <- 2 * (1:(n/2))
jm1 <- j - 1
sum(100 * (x[j] - x[jm1]^2)^2 + (1 - x[jm1])^2)
}

p0 <- rnorm(50)
spg(par=p0, fn=rosbkext)
BBoptim(par=p0, fn=rosbkext)

# compare the improvement in convergence when bounds are specified
BBoptim(par=p0, fn=rosbkext, lower=0)

# identical to spg() with defaults
BBoptim(par=p0, fn=rosbkext, method=3, control=list(M=10, trace=TRUE))
```

BBsolve *Solving Nonlinear System of Equations - A Wrapper for dfsane()*

Description

A strategy using different Barzilai-Borwein steplengths to solve a nonlinear system of equations.

Usage

```
  BBsolve(par, fn, method=c(2,3,1),
control=list(), quiet=FALSE, ...)
```

Arguments

`par`	A real vector argument to `fn`, indicating the initial guess for the root of the nonliinear system of equations `fn`.
`fn`	Nonlinear system of equation that is to be solved. A vector function that takes a real vector as argument and returns a real vector of the same length.
`method`	A vector of integers specifying which Barzilai-Borwein steplengths should be used in a consecutive manner. The methods will be used in the order specified.
`control`	A list of parameters governing the algorithm behaviour. This list is the same as that for `dfsane` and `sane` (excepting the default for `trace`). See `details` for important special features of control parameters.
`quiet`	logical indicating if messages about convergence success or failure should be suppressed
`...`	arguments passed fn (via the optimization algorithm).

Details

This wrapper is especially useful in problems where the algorithms (`dfsane` or `sane`) are likely to experience difficulties in convergence. When these algorithms with default parameters fail, i.e. when `convergence > 0` is obtained, a user might attempt various strategies to find a root of the nonlinear system. The function `BBsolve` tries the following sequential strategy:

1. Try a different BB steplength. Since the default is `method = 2` for `dfsane`, the BBsolve wrapper tries `method = c(2, 1, 3)`.
2. Try a different non-monotonicity parameter `M` for each method, i.e. BBsolve wrapper tries `M = c(50, 10)` for each BB steplength.
3. Try with Nelder-Mead initialization. Since the default for `dfsane` is `NM = FALSE`, BBsolve does `NM = c(TRUE, FALSE)`.

The argument `control` defaults to a list with values `maxit = 1500, M = c(50, 10), tol = 1e-07, trace = FALSE, triter = 10, noimp = 100, NM = c(TRUE, FALSE)`. If `control` is specified as an argument, only values which are different need to be given in the list. See `dfsane` for more details.

Value

A list with the same elements as returned by `dfsane` or `sane`. One additional element returned is `cpar` which contains the control parameter settings used to obtain successful convergence, or to obtain the best solution in case of failure.

References

R Varadhan and PD Gilbert (2009), BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function, *J. Statistical Software*, 32:4, http://www.jstatsoft.org/v32/i04/

See Also

BBoptim, dfsane, sane multiStart

Examples

```
# Use a preset seed so test values are reproducable.
require("setRNG")
old.seed <- setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion",
    seed=1234))

broydt <- function(x) {
n <- length(x)
f <- rep(NA, n)
h <- 2
f[1] <- ((3 - h*x[1]) * x[1]) - 2*x[2] + 1
tnm1 <- 2:(n-1)
f[tnm1] <- ((3 - h*x[tnm1]) * x[tnm1]) - x[tnm1-1] - 2*x[tnm1+1] + 1
f[n] <- ((3 - h*x[n]) * x[n]) - x[n-1] + 1
f
}

p0 <- rnorm(50)
BBsolve(par=p0, fn=broydt)  # this works
dfsane(par=p0, fn=broydt) # but this is highly unliikely to work.

# this implements the 3 BB steplengths with M = 50, and without Nelder-Mead initialization
BBsolve(par=p0, fn=broydt, control=list(M=50, NM=FALSE))

# this implements BB steplength 1 with M = 50 and 10, and both with and
#   without Nelder-Mead initialization
BBsolve(par=p0, fn=broydt, method=1, control=list(M=c(50, 10)))

# identical to dfsane() with defaults
BBsolve(par=p0, fn=broydt, method=2, control=list(M=10, NM=FALSE))
```

---

dfsane *Solving Large-Scale Nonlinear System of Equations*

Description

Derivative-Free Spectral Approach for solving nonlinear systems of equations

Usage

```
dfsane(par, fn, method=2, control=list(),
       quiet=FALSE, alertConvergence=TRUE, ...)
```

Arguments

- `fn` — a function that takes a real vector as argument and returns a real vector of same length (see details).
- `par` — A real vector argument to `fn`, indicating the initial guess for the root of the nonlinear system.
- `method` — An integer (1, 2, or 3) specifying which Barzilai-Borwein steplength to use. The default is 2. See *Details*.
- `control` — A list of control parameters. See *Details*.
- `quiet` — A logical variable (TRUE/FALSE). If `TRUE` warnings and some additional information printing are suppressed. Default is `quiet = FALSE` Note that `quiet` and the `control` variable `trace` affect different printing, so if `trace` is not set to `FALSE` there will be considerable printed output.
- `alertConvergence` — A logical variable. With the default `TRUE` a warning is issued if convergence is not obtained. When set to `FALSE` the warning is suppressed.
- `...` — Additional arguments passed to `fn`.

Details

The function `dfsane` is another algorithm for implementing non-monotone spectral residual method for finding a root of nonlinear systems, by working without gradient information. It stands for "derivative-free spectral approach for nonlinear equations". It differs from the function `sane` in that `sane` requires an approximation of a directional derivative at every iteration of the merit function $F(x)^t F(x)$.

R adaptation, with significant modifications, by Ravi Varadhan, Johns Hopkins University (March 25, 2008), from the original FORTRAN code of La Cruz, Martinez, and Raydan (2006).

A major modification in our R adaptation of the original FORTRAN code is the availability of 3 different options for Barzilai-Borwein (BB) steplengths: `method = 1` is the BB steplength used in LaCruz, Martinez and Raydan (2006); `method = 2` is equivalent to the other steplength proposed in Barzilai and Borwein's (1988) original paper. Finally, `method = 3`, is a new steplength, which is equivalent to that first proposed in Varadhan and Roland (2008) for accelerating the EM algorithm. In fact, Varadhan and Roland (2008) considered 3 similar steplength schemes in their EM acceleration work. Here, we have chosen `method = 2` as the "default" method, since it generally performe better than the other schemes in our numerical experiments.

Argument `control` is a list specifing any changes to default values of algorithm control parameters. Note that the names of these must be specified completely. Partial matching does not work.

M A positive integer, typically between 5-20, that controls the monotonicity of the algorithm. `M=1` would enforce strict monotonicity in the reduction of L2-norm of `fn`, whereas larger values allow for more non-monotonicity. Global convergence under non-monotonicity is ensured by enforcing the Grippo-Lampariello-Lucidi condition (Grippo et al. 1986) in a non-monotone line-search algorithm. Values of `M` between 5 to 20 are generally good, although some problems may require a much larger M. The default is `M = 10`.

maxit The maximum number of iterations. The default is `maxit = 1500`.

tol The absolute convergence tolerance on the residual L2-norm of `fn`. Convergence is declared when $\|F(x)\|/\sqrt(npar) <$ tol. Default is `tol = 1.e-07`.

trace A logical variable (TRUE/FALSE). If `TRUE`, information on the progress of solving the system is produced. Default is `trace = !quiet`.

triter An integer that controls the frequency of tracing when `trace=TRUE`. Default is `triter=10`, which means that the L2-norm of `fn` is printed at every 10-th iteration.

noimp An integer. Algorithm is terminated when no progress has been made in reducing the merit function for `noimp` consecutive iterations. Default is `noimp=100`.

NM A logical variable that dictates whether the Nelder-Mead algorithm in `optim` will be called upon to improve user-specified starting value. Default is `NM=FALSE`.

BFGS A logical variable that dictates whether the low-memory L-BFGS-B algorithm in `optim` will be called after certain types of unsuccessful termination of `dfsane`. Default is `BFGS=FALSE`.

Value

A list with the following components:

`par`	The best set of parameters that solves the nonlinear system.
`residual`	L2-norm of the function at convergence, divided by `sqrt(npar)`, where "npar" is the number of parameters.
`fn.reduction`	Reduction in the L2-norm of the function from the initial L2-norm.
`feval`	Number of times `fn` was evaluated.
`iter`	Number of iterations taken by the algorithm.
`convergence`	An integer code indicating type of convergence. `0` indicates successful convergence, in which case the `resid` is smaller than `tol`. Error codes are `1` indicates that the iteration limit `maxit` has been reached. `2` is failure due to stagnation; `3` indicates error in function evaluation; `4` is failure due to exceeding 100 steplength reductions in line-search; and `5` indicates lack of improvement in objective function over `noimp` consecutive iterations.
`message`	A text message explaining which termination criterion was used.

References

J Barzilai, and JM Borwein (1988), Two-point step size gradient methods, *IMA J Numerical Analysis*, 8, 141-148.

L Grippo, F Lampariello, and S Lucidi (1986), A nonmonotone line search technique for Newton's method, *SIAM J on Numerical Analysis*, 23, 707-716.

W LaCruz, JM Martinez, and M Raydan (2006), Spectral residual mathod without gradient information for solving large-scale nonlinear systems of equations, *Mathematics of Computation*, 75, 1429-1448.

R Varadhan and C Roland (2008), Simple and globally-convergent methods for accelerating the convergence of any EM algorithm, *Scandinavian J Statistics*.

R Varadhan and PD Gilbert (2009), BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function, *J. Statistical Software*, 32:4, http://www.jstatsoft.org/v32/i04/

See Also

BBsolve, sane, spg, grad

Examples

```
  trigexp <- function(x) {
# Test function No. 12 in the Appendix of LaCruz and Raydan (2003)
    n <- length(x)
    F <- rep(NA, n)
    F[1] <- 3*x[1]^2 + 2*x[2] - 5 + sin(x[1] - x[2]) * sin(x[1] + x[2])
    tn1 <- 2:(n-1)
    F[tn1] <- -x[tn1-1] * exp(x[tn1-1] - x[tn1]) + x[tn1] * ( 4 + 3*x[tn1]^2) +
        2 * x[tn1 + 1] + sin(x[tn1] - x[tn1 + 1]) * sin(x[tn1] + x[tn1 + 1]) - 8
    F[n] <- -x[n-1] * exp(x[n-1] - x[n]) + 4*x[n] - 3
    F
    }

    p0 <- rnorm(50)
    dfsane(par=p0, fn=trigexp)  # default is method=2
    dfsane(par=p0, fn=trigexp, method=1)
    dfsane(par=p0, fn=trigexp, method=3)
    dfsane(par=p0, fn=trigexp, control=list(triter=5, M=5))
######################################
 brent <- function(x) {
  n <- length(x)
  tnm1 <- 2:(n-1)
  F <- rep(NA, n)
  F[1] <- 3 * x[1] * (x[2] - 2*x[1]) + (x[2]^2)/4
  F[tnm1] <-  3 * x[tnm1] * (x[tnm1+1] - 2 * x[tnm1] + x[tnm1-1]) +
              ((x[tnm1+1] - x[tnm1-1])^2) / 4
  F[n] <- 3 * x[n] * (20 - 2 * x[n] + x[n-1]) +  ((20 - x[n-1])^2) / 4
  F
  }

  p0 <- sort(runif(50, 0, 20))
  dfsane(par=p0, fn=brent, control=list(trace=FALSE))
  dfsane(par=p0, fn=brent, control=list(M=200, trace=FALSE))
```

`multiStart` *Nonlinear Optimization or Root-Finding with Multiple Starting Values*

Description

Start BBsolve or BBoptim from multiple starting points to obtain multiple solutions and to test sensitivity to starting values.

Usage

```
  multiStart(par, fn, gr=NULL, action = c("solve", "optimize"),
method=c(2,3,1),  lower=-Inf, upper=Inf,
project=NULL, projectArgs=NULL,
control=list(),  quiet=FALSE, details=FALSE, ...)
```

Arguments

`par`	A real matrix, each row of which is an argument to `fn`, indicating initial guesses for solving a nonlinear system `fn = 0` or for optimizing the objective function `fn`.
`fn`	see `BBsolve` or `BBoptim`.
`gr`	Only required for optimization. See `BBoptim`.
`action`	A character string indicating whether to solve a nonlinear system or to optimize. Default is "solve".
`method`	see `BBsolve` or `BBoptim`.
`upper`	An upper bound for box constraints. See `spg`
`lower`	An lower bound for box constraints. See `spg`
`project`	A projection function or character string indicating its name. The projection function that takes a point in R^n and projects it onto a region that defines the constraints of the problem. This is a vector-function that takes a real vector as argument and returns a real vector of the same length. See `spg` for more details.
`projectArgs`	A list with arguments to the `project` function.
`control`	See `BBsolve` and `BBoptim`.
`quiet`	A logical variable (TRUE/FALSE). If `TRUE` warnings and some additional information printing are suppressed. Default is `quiet = FALSE` Note that the `control` variable `trace` and `quiet` affect different printing, so if `trace` is not set to `FALSE` there will be considerable printed output.
`details`	Logical indicating if the result should include the full result from `BBsolve` or `BBoptim` for each starting value.
`...`	arguments passed fn (via the optimization algorithm).

Details

The optimization or root-finder is run with each row of `par` indicating initial guesses.

Value

list with elements `par`, `values`, and `converged`. It optionally returns an attribute called "details", which is a list as long as the number of starting values, which contains the complete object returned by `dfsane` or `spg` for each starting value.

References

R Varadhan and PD Gilbert (2009), BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function, *J. Statistical Software*, 32:4, http://www.jstatsoft.org/v32/i04/

See Also

BBsolve, BBoptim, dfsane, spg

Examples

```
# Use a preset seed so the example is reproducable.
require("setRNG")
old.seed <- setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion",
    seed=1234))

# Finding multiple roots of a nonlinear system
brownlin <- function(x) {
# Brown's almost linear system(A.P. Morgan, ACM 1983)
# two distinct solutions if n is even
# three distinct solutions if n is odd
   n <- length(x)
   f <- rep(NA, n)
nm1 <- 1:(n-1)
f[nm1] <- x[nm1] + sum(x) - (n+1)
f[n] <- prod(x) - 1
f
}

p <- 9
n <- 50
p0 <- matrix(rnorm(n*p), n, p)  # n starting values, each of length p
ans <- multiStart(par=p0, fn=brownlin)
pmat <- ans$par[ans$conv, 1:p] # selecting only converged solutions
ord1 <- order(abs(pmat[,1]))
round(pmat[ord1, ], 3)  # all 3 roots can be seen

# An optimization example
rosbkext <- function(x){
n <- length(x)
j <- 2 * (1:(n/2))
jm1 <- j - 1
sum(100 * (x[j] - x[jm1]^2)^2 + (1 - x[jm1])^2)
}

p0 <- rnorm(50)
spg(par=p0, fn=rosbkext)
BBoptim(par=p0, fn=rosbkext)

pmat <- matrix(rnorm(100), 20, 5)  # 20 starting values each of length 5
ans <- multiStart(par=pmat, fn=rosbkext, action="optimize")
ans
```

```
attr(ans, "details")[[1]]  #

pmat <- ans$par[ans$conv, 1:5] # selecting only converged solutions
round(pmat, 3)
```

---

project *spg Projection Functions*

Description

Projection function implementing contraints for spg parameters.

Usage

```
projectLinear(par, A, b, meq)
```

Arguments

par	A real vector argument (as for fn), indicating the parameter values to which the constraint should be applied.
A	A matrix. See details.
b	A vector. See details.
meq	See details.

Details

The function `projectLinear` can be used by `spg` to define the constraints of the problem. It projects a point in R^n onto a region that defines the constraints. It takes a real vector `par` as argument and returns a real vector of the same length.

The function `projectLinear` incorporates linear equalities and inequalities in nonlinear optimization using a projection method, where an infeasible point is projected onto the feasible region using a quadratic programming solver. The inequalities are defined such that: `A %*% x - b > 0` . The first 'meq' rows of A and the first 'meq' elements of `b` correspond to equality constraints.

Value

A vector of the constrained parameter values.

See Also

`spg`

Examples

```
# Example
fn <- function(x) (x[1] - 3/2)^2 + (x[2] - 1/8)^4

gr <- function(x) c(2 * (x[1] - 3/2) , 4 * (x[2] - 1/8)^3)

# This is the set of inequalities
# x[1] - x[2] >= -1
# x[1] + x[2] >= -1
# x[1] - x[2] <= 1
# x[1] + x[2] <= 1

# The inequalities are written in R such that:  Amat %*% x  >= b
Amat <- matrix(c(1, -1, 1, 1, -1, 1, -1, -1), 4, 2, byrow=TRUE)
b <- c(-1, -1, -1, -1)
meq <- 0  # all 4 conditions are inequalities

p0 <- rnorm(2)
spg(par=p0, fn=fn, gr=gr, project="projectLinear",
      projectArgs=list(A=Amat, b=b, meq=meq))

meq <- 1  # first condition is now an equality
spg(par=p0, fn=fn, gr=gr, project="projectLinear",
      projectArgs=list(A=Amat, b=b, meq=meq))


# box-constraints can be incorporated as follows:
# x[1] >= 0
# x[2] >= 0
# x[1] <= 0.5
# x[2] <= 0.5

Amat <- matrix(c(1, 0, 0, 1, -1, 0, 0, -1), 4, 2, byrow=TRUE)
b <- c(0, 0, -0.5, -0.5)

meq <- 0
spg(par=p0, fn=fn, gr=gr, project="projectLinear",
   projectArgs=list(A=Amat, b=b, meq=meq))

# Note that the above is the same as the following:
spg(par=p0, fn=fn, gr=gr, lower=0, upper=0.5)


# An example showing how to impose other constraints in spg()

fr <- function(x) { ## Rosenbrock Banana function
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
  }

# Impose a constraint that sum(x) = 1
```

```
proj <- function(x){ x / sum(x) }

spg(par=runif(2), fn=fr, project="proj")

# Illustration of the importance of `projecting' the constraints, rather
#   than simply finding a feasible point:

fr <- function(x) { ## Rosenbrock Banana function
x1 <- x[1]
x2 <- x[2]
100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
# Impose a constraint that sum(x) = 1

proj <- function(x){
# Although this function does give a feasible point it is
#  not a "projection" in the sense of the nearest feasible point to `x'
x / sum(x)
}

p0 <- c(0.93, 0.94)

# Note, the starting value is infeasible so the next
#   result is "Maximum function evals exceeded"

spg(par=p0, fn=fr, project="proj")

# Correct approach to doing the projection using the `projectLinear' function

spg(par=p0, fn=fr, project="projectLinear", projectArgs=list(A=matrix(1, 1, 2), b=1, meq=1))

# Impose additional box constraint on first parameter

p0 <- c(0.4, 0.94)    # need feasible starting point

spg(par=p0, fn=fr,  lower=c(-0.5, -Inf), upper=c(0.5, Inf),
  project="projectLinear", projectArgs=list(A=matrix(1, 1, 2), b=1, meq=1))
```

---

sane *Solving Large-Scale Nonlinear System of Equations*

Description

Non-Monotone spectral approach for Solving Large-Scale Nonlinear Systems of Equations

Usage

```
sane(par, fn, method=2, control=list(),
     quiet=FALSE, alertConvergence=TRUE, ...)
```

Arguments

fn	a function that takes a real vector as argument and returns a real vector of same length (see details).
par	A real vector argument to fn, indicating the initial guess for the root of the nonlinear system.
method	An integer (1, 2, or 3) specifying which Barzilai-Borwein steplength to use. The default is 2. See *Details*.
control	A list of control parameters. See *Details*.
quiet	A logical variable (TRUE/FALSE). If TRUE warnings and some additional information printing are suppressed. Default is quiet = FALSE Note that quiet and the control variable trace affect different printing, so if trace is not set to FALSE there will be considerable printed output.
alertConvergence	A logical variable. With the default TRUE a warning is issued if convergence is not obtained. When set to FALSE the warning is suppressed.
...	Additional arguments passed to fn.

Details

The function sane implements a non-monotone spectral residual method for finding a root of nonlinear systems. It stands for "spectral approach for nonlinear equations". It differs from the function dfsane in that it requires an approximation of a directional derivative at every iteration of the merit function $F(x)^t F(x)$.

R adaptation, with significant modifications, by Ravi Varadhan, Johns Hopkins University (March 25, 2008), from the original FORTRAN code of La Cruz and Raydan (2003).

A major modification in our R adaptation of the original FORTRAN code is the availability of 3 different options for Barzilai-Borwein (BB) steplengths: method = 1 is the BB steplength used in LaCruz and Raydan (2003); method = 2 is equivalent to the other steplength proposed in Barzilai and Borwein's (1988) original paper. Finally, method = 3, is a new steplength, which is equivalent to that first proposed in Varadhan and Roland (2008) for accelerating the EM algorithm. In fact, Varadhan and Roland (2008) considered 3 equivalent steplength schemes in their EM acceleration work. Here, we have chosen method = 2 as the "default" method, as it generally performed better than the other schemes in our numerical experiments.

Argument control is a list specifing any changes to default values of algorithm control parameters. Note that the names of these must be specified completely. Partial matching will not work. Argument control has the following components:

M A positive integer, typically between 5-20, that controls the monotonicity of the algorithm. M=1 would enforce strict monotonicity in the reduction of L2-norm of fn, whereas larger values allow for more non-monotonicity. Global convergence under non-monotonicity is ensured by enforcing the Grippo-Lampariello-Lucidi condition (Grippo et al. 1986) in a non-monotone line-search algorithm. Values of `M` between 5 to 20 are generally good, although some problems may require a much larger M. The default is `M = 10`.

maxit The maximum number of iterations. The default is `maxit = 1500`.

tol The absolute convergence tolerance on the residual L2-norm of `fn`. Convergence is declared when $\|F(x)\|/\sqrt(npar) <$ tol. Default is `tol = 1.e-07`.

trace A logical variable (TRUE/FALSE). If `TRUE`, information on the progress of solving the system is produced. Default is `trace = !quiet`.

triter An integer that controls the frequency of tracing when `trace=TRUE`. Default is `triter=10`, which means that the L2-norm of `fn` is printed at every 10-th iteration.

noimp An integer. Algorithm is terminated when no progress has been made in reducing the merit function for `noimp` consecutive iterations. Default is `noimp=100`.

NM A logical variable that dictates whether the Nelder-Mead algorithm in `optim` will be called upon to improve user-specified starting value. Default is `NM=FALSE`.

BFGS A logical variable that dictates whether the low-memory L-BFGS-B algorithm in `optim` will be called after certain types of unsuccessful termination of `sane`. Default is `BFGS=FALSE`.

Value

A list with the following components:

`par`	The best set of parameters that solves the nonlinear system.
`residual`	L2-norm of the function evaluated at `par`, divided by `sqrt(npar)`, where "npar" is the number of parameters.
`fn.reduction`	Reduction in the L2-norm of the function from the initial L2-norm.
`feval`	Number of times `fn` was evaluated.
`iter`	Number of iterations taken by the algorithm.
`convergence`	An integer code indicating type of convergence. `0` indicates successful convergence, in which case the `resid` is smaller than `tol`. Error codes are `1` indicates that the iteration limit `maxit` has been reached. `2` indicates error in function evaluation; `3` is failure due to exceeding 100 steplength reductions in line-search; `4` denotes failure due to an anomalous iteration; and `5` indicates lack of improvement in objective function over `noimp` consecutive iterations.
`message`	A text message explaining which termination criterion was used.

References

J Barzilai, and JM Borwein (1988), Two-point step size gradient methods, *IMA J Numerical Analysis*, 8, 141-148.

L Grippo, F Lampariello, and S Lucidi (1986), A nonmonotone line search technique for Newton's method, *SIAM J on Numerical Analysis*, 23, 707-716.

W LaCruz, and M Raydan (2003), Nonmonotone spectral methods for large-scale nonlinear systems, *Optimization Methods and Software*, 18, 583-599.

R Varadhan and C Roland (2008), Simple and globally-convergent methods for accelerating the convergence of any EM algorithm, *Scandinavian J Statistics*.

R Varadhan and PD Gilbert (2009), BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function, *J. Statistical Software*, 32:4, http://www.jstatsoft.org/v32/i04/

See Also

BBsolve, dfsane, spg, grad

Examples

```
  trigexp <- function(x) {
# Test function No. 12 in the Appendix of LaCruz and Raydan (2003)
    n <- length(x)
    F <- rep(NA, n)
    F[1] <- 3*x[1]^2 + 2*x[2] - 5 + sin(x[1] - x[2]) * sin(x[1] + x[2])
    tn1 <- 2:(n-1)
    F[tn1] <- -x[tn1-1] * exp(x[tn1-1] - x[tn1]) + x[tn1] * ( 4 + 3*x[tn1]^2) +
        2 * x[tn1 + 1] + sin(x[tn1] - x[tn1 + 1]) * sin(x[tn1] + x[tn1 + 1]) - 8
    F[n] <- -x[n-1] * exp(x[n-1] - x[n]) + 4*x[n] - 3
    F
    }

    p0 <- rnorm(50)
    sane(par=p0, fn=trigexp)
    sane(par=p0, fn=trigexp, method=1)
######################################
brent <- function(x) {
  n <- length(x)
  tnm1 <- 2:(n-1)
  F <- rep(NA, n)
  F[1] <- 3 * x[1] * (x[2] - 2*x[1]) + (x[2]^2)/4
  F[tnm1] <- 3 * x[tnm1] * (x[tnm1+1] - 2 * x[tnm1] + x[tnm1-1]) +
               ((x[tnm1+1] - x[tnm1-1])^2) / 4
  F[n] <- 3 * x[n] * (20 - 2 * x[n] + x[n-1]) +  ((20 - x[n-1])^2) / 4
  F
  }

  p0 <- sort(runif(50, 0, 10))
  sane(par=p0, fn=brent, control=list(trace=FALSE))
  sane(par=p0, fn=brent, control=list(M=200, trace=FALSE))
```

---

spg *Large-Scale Optimization*

Description

Spectral projected gradient method for large-scale optimization with simple constraints.

Usage

```
spg(par, fn, gr=NULL, method=3, lower=-Inf, upper=Inf,
      project=NULL, projectArgs=NULL,
control=list(), quiet=FALSE, alertConvergence=TRUE, ...)
```

Arguments

`par` A real vector argument to `fn`, indicating the initial guess for the optimization of nonlinear objective function `fn`.

`fn` Nonlinear objective function that is to be optimized. A scalar function that takes a real vector as argument and returns a scalar that is the value of the function at that point (see details).

`gr` The gradient of the objective function `fn` evaluated at the argument. This is a vector-function that takes a real vector as argument and returns a real vector of the same length. It defaults to "NULL", which means that gradient is evaluated numerically. Computations are dramatically faster in high-dimensional problems when the exact gradient is provided. See *Example*.

`method` An integer (1, 2, or 3) specifying which Barzilai-Borwein steplength to use. The default is 3. See *Details*.

`upper` An upper bound for box constraints.

`lower` An lower bound for box constraints.

`project` A projection function or character string indicating its name. The projection function takes a point in R^n and projects it onto a region that defines the constraints of the problem. This is a vector-function that takes a real vector as argument and returns a real vector of the same length. See *Details*. If a projection function is not supplied, arguments `lower` and `upper` will cause the use of an internally defined function that enforces the implied box constraints.

`projectArgs` A list with arguments to the `project` function. See *Details*.

`control` A list of control parameters. See *Details*.

`quiet` A logical variable (TRUE/FALSE). If `TRUE` warnings and some additional information printing are suppressed. Default is `quiet = FALSE` Note that `quiet` and the `control` variable `trace` affect different printing, so if `trace` is not set to `FALSE` there will be considerable printed output.

`alertConvergence` A logical variable. With the default `TRUE` a warning is issued if convergence is not obtained. When set to `FALSE` the warning is suppressed.

`...` Additional arguments passed to `fn` and `gr`. (Both must accept any specified arguments, either explicitly or by having a ... argument, but they do not need to use them all.)

Details

R adaptation, with significant modifications, by Ravi Varadhan, Johns Hopkins University (March 25, 2008), from the original FORTRAN code of Birgin, Martinez, and Raydan (2001). The original is available at the TANGO project http://www.ime.usp.br/~egbirgin/tango/downloads.php

A major modification in our R adaptation of the original FORTRAN code is the availability of 3 different options for Barzilai-Borwein (BB) steplengths: `method = 1` is the BB steplength used in Birgin, Martinez and Raydan (2000); `method = 2` is the other steplength proposed in Barzilai and Borwein's (1988) original paper. Finally, `method = 3`, is a new steplength, which was first proposed in Varadhan and Roland (2008) for accelerating the EM algorithm. In fact, Varadhan and Roland (2008) considered 3 similar steplength schemes in their EM acceleration work. Here, we have chosen `method = 3` as the "default" method. This method may be slightly slower than the other 2 BB steplength schemes, but it generally exhibited more reliable convergence to a better optimum in our experiments. We recommend that the user try the other steplength schemes if the default method does not perform well in their problem.

Box constraints can be imposed by vectors `lower` and `upper`. Scalar values for `lower` and `upper` are expanded to apply to all parameters. The default `lower` is `-Inf` and `upper` is `+Inf`, which imply no constraints.

The `project` argument provides a way to implement more general constraints to be imposed on the parameters in `spg`. `projectArgs` is passed to the `project` function if one is specified. The first argument of any `project` function should be `par` and any other arguments should be passed using its argument `projectArgs`. To avoid confusion it is suggested that user defined `project` functions should not use arguments `lower` and `upper`.

The function `projectLinear` incorporates linear equalities and inequalities. This function also provides an example of how other projections might be implemented.

Argument `control` is a list specifing any changes to default values of algorithm control parameters. Note that the names of these must be specified completely. Partial matching will not work. The list items are as follows:

M A positive integer, typically between 5-20, that controls the monotonicity of the algorithm. `M=1` would enforce strict monotonicity in the reduction of L2-norm of `fn`, whereas larger values allow for more non-monotonicity. Global convergence under non-monotonicity is ensured by enforcing the Grippo-Lampariello-Lucidi condition (Grippo et al. 1986) in a non-monotone line-search algorithm. Values of `M` between 5 to 20 are generally good. The default is `M = 10`.

maxit The maximum number of iterations. The default is `maxit = 1500`.

ftol Convergence tolerance on the absolute change in objective function between successive iterations. Convergence is declared when the change is less than `ftol`. Default is `ftol = 1.e-10`.

gtol Convergence tolerance on the infinity-norm of projected gradient `gr` evaluated at the current parameter. Convergence is declared when the infinity-norm of projected gradient is less than `gtol`. Default is `gtol = 1.e-05`.

maxfeval Maximum limit on the number of function evaluations. Default is `maxfeval = 10000`.

maximize A logical variable indicating whether the objective function is to be maximized. Default is `maximize = FALSE` indicating minimization. For maximization (e.g. log-likelihood maximization in statistical modeling), this may be set to `TRUE`.

trace A logical variable (TRUE/FALSE). If `TRUE`, information on the progress of optimization is printed. Default is `trace = !quiet`.

triter An integer that controls the frequency of tracing when `trace=TRUE`. Default is `triter=10`, which means that the objective `fn` and the infinity-norm of its projected gradient are printed at every 10-th iteration.

eps A small positive increment used in the finite-difference approximation of gradient. Default is 1.e-07.

checkGrad NULL or a logical variable TRUE/FALSE indicating whether to check the provided analytical gradient against a numerical approximation. With the default NULL the gradient is checked if it is estimated to take less than about ten seconds. A warning will be issued in the case it takes longer. The default can be overridden by specifying TRUE or FALSE. It is recommended that this be set to FALSE for high-dimensional problems, after making sure that the gradient is correctly specified, possibly by running once with TRUE specified.

checkGrad.tol A small positive value use to compare the maximum relative difference between a user supplied gradient gr and the numerical approximation calculated by grad from package **numDeriv**. The default is 1.e-06. If this value is exceeded then an error message is issued, as it is a reasonable indication of a problem with the user supplied gr. The user can either fix the gr function, remove it so the finite-difference approximation is used, or increase the tolerance so the check passes.

Value

A list with the following components:

par	Parameters that optimize the nonlinear objective function, if convergence is successful.
value	The value of the objective function at termination.
gradient	L-infinity norm of the projected gradient of the objective function at termination. If convergence is successful, this should be less than gtol.
fn.reduction	Reduction in the value of the function from its initial value. This is negative in maximization.
iter	Number of iterations taken by the algorithm. The gradient is evaluated once each iteration, so the number of gradient evaluations will also be equal to iter, plus any evaluations necessary for checkGrad.
feval	Number of times the objective fn was evaluated.
convergence	An integer code indicating type of convergence. 0 indicates successful convergence, in which case the projcted gradient is smaller than pgtol or the change in objective function is smaller than ftol. Error codes are: 1 indicates that the maximum limit for iterations maxit has been reached. 2 indicates that maximum limit on function evals has been exceeded. 3 indicates failure due to error in function evaluation. 4 indicates failure due to error in gradient evaluation. 5 indicates failure due to error in projection.
message	A text message explaining which termination criterion was used.

References

Birgin EG, Martinez JM, and Raydan M (2000): Nonmonotone spectral projected gradient methods on convex sets, *SIAM J Optimization*, 10, 1196-1211.

Birgin EG, Martinez JM, and Raydan M (2001): SPG: software for convex-constrained optimization, *ACM Transactions on Mathematical Software*.

L Grippo, F Lampariello, and S Lucidi (1986), A nonmonotone line search technique for Newton's method, *SIAM J on Numerical Analysis*, 23, 707-716.

M Raydan (1997), Barzilai-Borwein gradient method for large-scale unconstrained minimization problem, *SIAM J of Optimization*, 7, 26-33.

R Varadhan and C Roland (2008), Simple and globally-convergent methods for accelerating the convergence of any EM algorithm, *Scandinavian J Statistics*, doi: 10.1111/j.1467-9469.2007.00585.x.

R Varadhan and PD Gilbert (2009), BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function, *J. Statistical Software*, 32:4, http://www.jstatsoft.org/v32/i04/

See Also

projectLinear, BBoptim, optim, nlm, sane, dfsane, grad

Examples

```
sc2.f <- function(x){ sum((1:length(x)) * (exp(x) - x)) / 10}

sc2.g <- function(x){ (1:length(x)) * (exp(x) - 1) / 10}

p0 <- rnorm(50)
ans.spg1 <- spg(par=p0, fn=sc2.f)  # Default is method=3
ans.spg2 <- spg(par=p0, fn=sc2.f, method=1)
ans.spg3 <- spg(par=p0, fn=sc2.f, method=2)
ans.cg <- optim(par=p0, fn=sc2.f, method="CG")  #Uses conjugate-gradient method in "optim"
ans.lbfgs <- optim(par=p0, fn=sc2.f, method="L-BFGS-B")  #Uses low-memory BFGS method in "optim"

# Now we use exact gradient.
# Computation is much faster compared to when using numerical gradient.
ans.spg1 <- spg(par=p0, fn=sc2.f, gr=sc2.g)

############
# Another example illustrating use of additional parameters to objective function
valley.f <- function(x, cons) {
  n <- length(x)
  f <- rep(NA, n)
  j <- 3 * (1:(n/3))
  jm2 <- j - 2
  jm1 <- j - 1
  f[jm2] <- (cons[2] * x[jm2]^3 + cons[1] * x[jm2]) * exp(-(x[jm2]^2)/100) - 1
  f[jm1] <- 10 * (sin(x[jm2]) - x[jm1])
  f[j] <- 10 * (cos(x[jm2]) - x[j])
  sum(f*f)
  }

k <- c(1.003344481605351, -3.344481605351171e-03)
p0 <- rnorm(30)  # number of parameters should be a multiple of 3 for this example
ans.spg2 <- spg(par=p0, fn=valley.f, cons=k, method=2)
ans.cg <- optim(par=p0, fn=valley.f, cons=k, method="CG")
ans.lbfgs <- optim(par=p0, fn=valley.f, cons=k, method="L-BFGS-B")

####################################################################
# Here is a statistical example illustrating log-likelihood maximization.
```

```
poissmix.loglik <- function(p,y) {
  # Log-likelihood for a binary Poisson mixture
  i <- 0:(length(y)-1)
  loglik <- y*log(p[1]*exp(-p[2])*p[2]^i/exp(lgamma(i+1)) +
        (1 - p[1])*exp(-p[3])*p[3]^i/exp(lgamma(i+1)))
  return (sum(loglik) )
  }

# Data from Hasselblad (JASA 1969)
poissmix.dat <- data.frame(death=0:9, freq=c(162,267,271,185,111,61,27,8,3,1))
y <- poissmix.dat$freq

# Lower and upper bounds on parameters
lo <- c(0.001,0,0)
hi <- c(0.999, Inf, Inf)

p0 <- runif(3,c(0.2,1,1),c(0.8,5,8))  # randomly generated starting values

ans.spg <- spg(par=p0, fn=poissmix.loglik, y=y, lower=lo, upper=hi,
     control=list(maximize=TRUE))

# how to compute hessian at the MLE
  require(numDeriv)
  hess <- hessian(x=ans.spg$par, poissmix.loglik, y=y)
  se <- sqrt(-diag(solve(hess)))  # approximate standard errors
```

Index