Package 'daarem'

July 22, 2025

Type Package

Title Damped Anderson Acceleration with Epsilon Monotonicity for Accelerating EM-Like Monotone Algorithms

Version 0.7

Date 2022-03-21

Maintainer Nicholas Henderson <nchender@umich.edu>

Imports stats, utils

Description Implements the DAAREM method for accelerating the convergence of slow, monotone sequences from smooth, fixed-point iterations such as the EM algorithm. For further details about the DAAREM method, see Henderson, N.C. and Varadhan, R. (2019) <doi:10.1080/10618600.2019.1594835>.

License GPL-2

URL https://doi.org/10.1080/10618600.2019.1594835

Encoding UTF-8

NeedsCompilation no

Author Nicholas Henderson [cre, aut], Ravi Varadhan [aut]

Repository CRAN

Date/Publication 2022-03-23 07:20:02 UTC

Contents

daarem *Damped Anderson Acceleration with Restarts and Epsilon-Montonicity for Accelerating Slowly-Convergent, Monotone Fixed-Point Iterations*

Description

An 'off-the-shelf' acceleration scheme for accelerating the convergence of *any* smooth, monotone, slowly-converging fixed-point iteration. It can be used to accelerate the convergence of a wide variety of montone iterations including, for example, expectation-maximization (EM) algorithms and majorization-minimization (MM) algorithms.

Usage

```
daarem(par, fixptfn, objfn, ..., control=list())
```

Arguments

`par`	A vector of starting values of the parameters.
`fixptfn`	A vector function, G that denotes the fixed-point mapping. This function is the most essential input in the package. It should accept a parameter vector as input and should return a parameter vector of the same length. This function defines the fixed-point iteration: $x_{k+1} = G(x_k)$. In the case of an EM algorithm, G defines a single E and M step.
`objfn`	This is a scalar function, L, that denotes a "merit" function which attains its local maximum at the fixed-point of G. The function L should accept a parameter vector as input and should return a scalar value. In the EM algorithm, the merit function L is the log-likelihood function. It is not necessary for the user to provide this argument though it is preferable.
`control`	A list of control parameters specifying any changes to default values of algorithm control parameters. Full names of control list elements must be specified, otherwise, user-specifications are ignored. See *Details*.
`...`	Arguments passed to `fixptfn` and `objfn`.

Details

Default values of `control` are: `maxiter=2000`, `order=10`, `tol=1e-08`, `mon.tol=0.01`, `cycl.mon.tol=0.0`, `alpha=1.2`, `kappa=25`, `resid.tol=0.95`, `convtype="param"`

`maxiter` An integer denoting the maximum limit on the number of evaluations of `fixptfn`, G. Default value is 2000.

`order` An integer 1 denoting the order of the DAAREM acceleration scheme.

`tol` A small, positive scalar that determines when iterations should be terminated. When `convtype` is set to "param", iteration is terminated when $||x_k - G(x_k)|| < tol$. Default is `1.e-08`.

`mon.tol` A nonnegative scalar that determines whether the montonicity condition is violated. The monotonicity condition is violated whenver $L(x[k+1]) < L(x[k]) - mon.tol$. Such violations determine how much damping is to be applied on subsequent steps of the algorithm. Default value of mon.tol is `1.e-02`.

`cycl.mon.tol` A nonegative scalar that determines whether a montonicity condition is violated after the end of the cycle. This cycle-level monotonicity condition is violated whenver $L(x[endcycle]) < L(x[startcycle]) - cycl.mon.tol$. Here, $x[startcycle]$ refers to the value of x at the beginning of the current cycle while $x[endcycle]$ refers to the value of x at the end of the current cycle. Such violations also determine how much damping is to be applied on subsequent steps of the algorithm.

`kappa` A nonnegative parameter which determines the "half-life" of relative damping and how quickly relative damping tends to one. In the absence of monotonicity violations, the relative damping factor is $<= 1/2$ for the first `kappa` iterations, and it is then greater than $1/2$ for all subsequent iterations. The relative damping factor is the ratio between the norm of the unconstrained coefficients in Anderson acceleration and the norm of the damped coefficients. In the absence of any monotonicity violations, the relative damping factor in iteration k is $1/(1+\alpha^(\kappa-k))$.

`alpha` A parameter > 1 that determines the initial relative damping factor and how quickly the relative damping factor tends to one. The initial relative damping factor is $1/(1+\alpha^{\kappa})$. In the absence of any monotonicity violations, the relative damping factor in iteration k is $1/(1+\alpha^(\kappa-k))$.

`resid.tol` A nonnegative scalar < 1 that determines whether a residual change condition is violated. The residual change condition is violated whenever $||x_k + 1 - G(x_k+1)|| > ||x_k - G(x_k)||(1 + resid.tol^k)$. Default value of resid.tol is `0.95`.

`convtype` This can equal either "param" or "objfn". When set to "param", convergence is determined by the criterion: $||x_k - G(x_k)|| \leq tol$. When set to "objfn", convergence is determined by the objective function-based criterion: $|L(x[k+1]) - L(x[k])| < tol$.

`intermed` A logical variable indicating whether or not the intermediate results of iterations should be returned. If set to `TRUE`, the function will return a matrix where each row corresponds to parameters at each iteration, along with the corresponding value of the objective function in the first column. This option is inactive when objfn is not specified. Default is `FALSE`.

Value

A list with the following components:

`par`	Parameter, $x*$ that are the fixed-point of G such that $x* = G(x*)$, if convergence is successful.
`value.objfn`	The value of the objective function L at termination.
`fpevals`	Number of times the fixed-point function `fixptfn` was evaluated.
`objfevals`	Number of times the objective function `objfn` was evaluated.
`convergence`	An integer code indicating type of convergence. `0` indicates successful convergence, whereas `1` denotes failure to converge.
`objfn.track`	A vector containing the value of the objective function at each iteration.
`p.intermed`	A matrix where each row corresponds to parameters at each iteration, along with the corresponding value of the objective function (in the first column). This

object is returned only when the control parameter `intermed` is set to `TRUE`. It is not returned when `objfn` is not specified.

Author(s)

Nicholas Henderson and Ravi Varadhan

References

Henderson, N.C. and Varadhan, R. (2019) *Damped Anderson acceleration with restarts and monotonicity control for accelerating EM and EM-like algorithms*, *Journal of Computational and Graphical Statistics, Vol. 28(4), 834-846.* doi: 10.1080/10618600.2019.1594835

See Also

`fpiter`

Examples

```
n <- 2000
npars <- 25
true.beta <- .5*rt(npars, df=2) + 2
XX <- matrix(rnorm(n*npars), nrow=n, ncol=npars)
yy <- ProbitSimulate(true.beta, XX)
max.iter <- 1000
beta.init <- rep(0.0, npars)

# Estimating Probit model with DAAREM acceleration
aa.probit <- daarem(par=beta.init, fixptfn = ProbitUpdate, objfn = ProbitLogLik,
                    X=XX, y=yy, control=list(maxiter=max.iter))

plot(aa.probit$objfn, type="b", xlab="Iterations", ylab="log-likelihood")


# Compare with estimating Probit model using the EM algorithm

max.iter <- 25000  # need more iterations for EM convergence
beta.init <- rep(0.0, npars)

em.probit <- fpiter(par=beta.init, fixptfn = ProbitUpdate, objfn = ProbitLogLik,
                    X=XX, y=yy, control=list(maxiter=max.iter))
c(aa.probit$fpevals, em.probit$fpevals)
c(aa.probit$value, em.probit$value)


# Accelerating using SQUAREM if the SQUAREM package is loaded
# library(SQUAREM)
# max.iter <- 5000
# sq.probit <- squarem(par=beta.init, fixptfn=ProbitUpdate, objfn=ProbitLogLik,
#                      X=XX, y=yy, control=list(maxiter=max.iter))

# print( c(aa.probit$fpevals, em.probit$fpevals, sq.probit$fpevals) )
```

```
# print( c(aa.probit$value, em.probit$value, sq.probit$value) )
# print( c(aa.probit$objfeval, em.probit$objfeval, sq.probit$objfeval) )
```

---

fpiter *Fixed-Point Iteration Scheme*

Description

A function to implement the fixed-point iteration algorithm. This includes monotone, contraction mappings including EM and MM algorithms

Usage

```
fpiter(par, fixptfn, objfn=NULL, control=list( ), ...)
```

Arguments

par	A vector of parameters denoting the initial guess for the fixed-point iteration.
fixptfn	A vector function, F that denotes the fixed-point mapping. This function is the most essential input in the package. It should accept a parameter vector as input and should return a parameter vector of same length. This function defines the fixed-point iteration: $x_{k+1} = F(x_k)$. In the case of EM algorithm, F defines a single E and M step.
objfn	This is a scalar function, L, that denotes a "merit" function which attains its local minimum at the fixed-point of F. This function should accept a parameter vector as input and should return a scalar value. In the EM algorithm, the merit function L is the log-likelihood. In some problems, a natural merit function may not exist, in which case the algorithm works with only fixptfn. The merit function function objfn does not have to be specified, even when a natural merit function is available, especially when its computation is expensive.
control	A list of control parameters to pass on to the algorithm. Full names of control list elements must be specified, otherwise, user-specifications are ignored. See *Details* below.
...	Arguments passed to fixptfn and objfn.

Details

control is list of control parameters for the algorithm.

control = list(tol = 1.e-07, maxiter = 1500, trace = FALSE)

tol A small, positive scalar that determines when iterations should be terminated. Iteration is terminated when $||x_k - F(x_k)|| \leq tol$. Default is 1.e-07.

maxiter An integer denoting the maximum limit on the number of evaluations of fixptfn, F. Default is 1500.

trace A logical variable denoting whether some of the intermediate results of iterations should be displayed to the user. Default is FALSE.

Value

A list with the following components:

par	Parameter,$x*$ that are the fixed-point of F such that $x* = F(x*)$, if convergence is successful.
value.objfn	The value of the objective function L at termination.
fpevals	Number of times the fixed-point function fixptfn was evaluated.
objfevals	Number of times the objective function objfn was evaluated.
convergence	An integer code indicating type of convergence. 0 indicates successful convergence, whereas 1 denotes failure to converge.

See Also

daarem

Examples

```
### Generate outcomes from a probit regression model
n <- 1000
npars <- 5
true.beta <- .5*rt(npars, df=2) + 1
XX <- matrix(rnorm(n*npars), nrow=n, ncol=npars)
yy <- ProbitSimulate(true.beta, XX)
max.iter <- 1000
beta.init <- rep(0.0, npars)

### EM algorithm for estimating parameters from probit regression

em.probit <- fpiter(par=beta.init, fixptfn = ProbitUpdate, X=XX, y=yy,
                    control=list(maxiter=max.iter))
```

ProbitLogLik *Probit Regression Log-Likelihood Function*

Description

Given a design matrix and vector of binary responses, this function evaluates the log-likelihood function for the Probit regression model.

Usage

```
ProbitLogLik(beta.hat, X, y)
```

Arguments

beta.hat	A vector of length p. The current estimates of the regression parameters.
X	The n x p design matrix for the Probit regression model.
y	Vector of length n containing binary outcomes (either 0 or 1).

Value

A scalar - the value of the log-likelihood at beta.hat.

Author(s)

Nicholas Henderson

See Also

ProbitSimulate, ProbitUpdate

Examples

```
n <- 200
npars <- 5
true.beta <- .5*rt(npars, df=2) + 2
XX <- matrix(rnorm(n*npars), nrow=n, ncol=npars)
yy <- ProbitSimulate(true.beta, XX)

initial.beta <- rep(0.0, npars)
ll <- ProbitLogLik(initial.beta, XX, yy)
```

ProbitSimulate *Simulate Data from a Probit Regression Model*

Description

Function to simulate data from a Probit regression model. User provides a design matrix and a vector of regression coefficients. Output is a vector of 0/1 responses.

Usage

```
ProbitSimulate(beta.vec, X)
```

Arguments

beta.vec	A vector of length p containing the true regression coefficients of the Probit regression model to be simulated from.
X	An n x p design matrix for the Probit regression model to be simulated from.

Value

A vector of length n containing binary outcomes (i.e., 0 or 1).

Author(s)

Nicholas Henderson

See Also

ProbitUpdate, ProbitLogLik

Examples

```
n <- 200
npars <- 5
true.beta <- .5*rt(npars, df=2) + 2
XX <- matrix(rnorm(n*npars), nrow=n, ncol=npars)
yy <- ProbitSimulate(true.beta, XX)
```

ProbitUpdate *EM Algorithm Update for Probit Regression*

Description

Function performs an EM update (both the E and M steps) of the parameters for a Probit regression model.

Usage

```
ProbitUpdate(beta.hat, X, y)
```

Arguments

beta.hat	A vector of length p. The current estimates of the regression parameters.
X	The n x p design matrix for the Probit regression model.
y	Vector of length n containing binary outcomes (either 0 or 1).

Value

A vector of length p - the updated parameter values.

Author(s)

Nicholas Henderson

See Also

ProbitSimulate, ProbitLogLik

Examples

```
n <- 200
npars <- 5
true.beta <- .5*rt(npars, df=2) + 2
XX <- matrix(rnorm(n*npars), nrow=n, ncol=npars)
yy <- ProbitSimulate(true.beta, XX)

initial.beta <- rep(0.0, npars)
new.beta <- ProbitUpdate(initial.beta, XX, yy)
```

Index