Package 'SQUAREM'

October 21, 2020

Version 2020.5

Date 2020-10-21

Title Squared Extrapolation Methods for Accelerating EM-Like Monotone Algorithms

Description Algorithms for accelerating the convergence of slow, monotone sequences from smooth, contraction mapping such as the EM algorithm. It can be used to accelerate any smooth, linearly convergent acceleration scheme. 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(``SQUAREM''). Refer to the J Stat Software article: <doi:10.18637/jss.v092.i07>.

Depends R (>= 3.0)

Suggests setRNG

LazyLoad yes

License GPL (>= 2)

Author Ravi Varadhan

Maintainer Ravi Varadhan <ravi.varadhan@jhu.edu>

URL https://coah.jhu.edu/people/Faculty_personal_Pages/Varadhan.html

Repository CRAN

NeedsCompilation no

Date/Publication 2020-10-21 21:00:02 UTC

R topics documented:

`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.

References

R Varadhan and C Roland (2008), Simple and globally convergent numerical schemes for accelerating the convergence of any EM algorithm, *Scandinavian Journal of Statistics*, 35:335-353.

C Roland, R Varadhan, and CE Frangakis (2007), Squared polynomial extrapolation methods with cycling: an application to the positron emission tomography problem, *Numerical Algorithms*, 44:159-172.

See Also

`squarem`

Examples

```
##############################################################################
# Example 1:  EM algorithm for Poisson mixture estimation
poissmix.em <- function(p,y) {
# The fixed point mapping giving a single E and M step of the EM algorithm
#
pnew <- rep(NA,3)
i <- 0:(length(y)-1)
zi <- p[1]*exp(-p[2])*p[2]^i / (p[1]*exp(-p[2])*p[2]^i + (1 - p[1])*exp(-p[3])*p[3]^i)
pnew[1] <- sum(y*zi)/sum(y)
pnew[2] <- sum(y*i*zi)/sum(y*zi)
pnew[3] <- sum(y*i*(1-zi))/sum(y*(1-zi))
p <- pnew
return(pnew)
}

poissmix.loglik <- function(p,y) {
# Objective function whose local minimum is a fixed point \
# negative log-likelihood of 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) )
}
```

```
# Real 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
tol <- 1.e-08

# Use a preset seed so the example is reproducable.
require("setRNG")
old.seed <- setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion",
    seed=54321))

p0 <- c(runif(1),runif(2,0,4))  # random starting value

# Basic EM algorithm
pf1 <- fpiter(p=p0, y=y, fixptfn=poissmix.em, objfn=poissmix.loglik, control=list(tol=tol))


##############################################################################
# Example 2:  Accelerating the convergence of power method iteration for
# finding the dominant eigenvector of a matrix

power.method <- function(x, A) {

# Defines one iteration of the power method
# x = starting guess for dominant eigenvector
# A = a square matrix

ax <- as.numeric(A %*% x)
f <- ax / sqrt(as.numeric(crossprod(ax)))
f
}

# Finding the dominant eigenvector of the Bodewig matrix
# This is a famous matrix for which power method has trouble converging
# See, for example, Sidi, Ford, and Smith (SIAM Review, 1988)
#
# Note: there are two eigenvalues that are equally dominant,
#  but have opposite signs.
# Sometimes the power method finds the eigenvector corresponding to the
# large positive eigenvalue, but other times it finds the eigenvector
# corresponding to the large negative eigenvalue
b <- c(2, 1, 3, 4, 1,  -3,   1,   5,  3,   1,   6,  -2,  4,   5,  -2,  -1)
bodewig.mat <- matrix(b,4,4)
eigen(bodewig.mat)

p0 <- rnorm(4)

# Standard power method iteration
ans1 <- fpiter(p0, fixptfn=power.method, A=bodewig.mat)
# re-scaling the eigenvector so that it has unit length
ans1$par <- ans1$par / sqrt(sum(ans1$par^2))
ans1
```

`squarem` *Squared Extrapolation Methods for Accelerating Slowly-Convergent Fixed-Point Iterations*

Description

Globally-convergent, partially monotone, acceleration schemes for accelerating the convergence of *any* smooth, monotone, slowly-converging contraction mapping. It can be used to accelerate the convergence of a wide variety of iterations including the expectation-maximization (EM) algorithms and its variants, majorization-minimization (MM) algorithm, power method for dominant eigenvalue-eigenvector, Google's page-rank algorithm, and multi-dimensional scaling.

Usage

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

Arguments

`par`	A vector of parameters denoting the initial guess for the fixed-point.
`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 maximum or minimum at the fixed -point of F. Also see the control parameter `minimize` which determines whether the objective function is minimized or maximized. The objective function should accept a parameter vector as input and should return a scalar value. In the EM algorithm, the merit function L is the either log-likelihood or its negative. In someproblems, 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 specifing 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

The function `squarem` is a general-purpose algorithm for accelerating the convergence of *any* slowly-convergent (smooth) fixed-point iteration. Full names of

Default values of `control` are: `K=1`, `method=3`, `minimize=TRUE`, `square=TRUE`, `step.min0=1`, `step.max0=1`, `mstep=4`, `objfn.inc=1`, `kr=1`, `tol=1e-07`, `maxiter=1500`, `trace=FALSE`, `intermed=FALSE`.

K An integer denoting the order of the SQUAREM scheme. Default is 1, which is a first-order scheme developed in Varadhan and Roland (2008). Our experience is that first-order schemes are adequate for most problems. K=2,3 may provide greater speed in some problems, although they are less reliable than the first-order schemes.

method Either an integer or a character variable that denotes the particular SQUAREM scheme to be used. When K=1, method should be an integer, either 1, 2, or 3. These correspond to the 3 schemes discussed in Varadhan and Roland (2008). Default is method=3. When K > 1, method should be a character string, either ''RRE'' or ''MPE''. These correspond to the reduced-rank extrapolation or squared minimal=polynomial extrapolation (See Roland, Varadhan, and Frangakis (2007)). Default is "RRE".

minimize A logical variable. By default it is set to TRUE for minimization of the objective function. If the objective function is to be maximized, which is commonly the case for likelihood estimation, it should be changed to FALSE.

square A logical variable indicating whether or not a squared extrapolation scheme should be used. Our experience is that the squared extrapolation schemes are faster and more stable than the unsquared schemes. Hence, we have set the default as TRUE.

step.min0 A scalar denoting the minimum steplength taken by a SQUAREM algorithm. Default is 1. For contractive fixed-point iterations (e.g. EM and MM), this defualt works well. In problems where an eigenvalue of the Jacobian of F is outside of the interval $(0, 1)$, step.min0 should be less than 1 or even negative in some cases.

step.max0 A positive-valued scalar denoting the initial value of the maximum steplength taken by a SQUAREM algorithm. Default is 1. When the steplength computed by SQUAREM exceeds step.max0, the steplength is set equal to step.max0, but then step.max0 is increased by a factor of mstep.

mstep A scalar greater than 1. When the steplength computed by SQUAREM exceeds step.max0, the steplength is set equal to step.max0, but step.max0 is increased by a factor of mstep. Default is 4.

objfn.inc A non-negative scalar that dictates the degree of non-montonicity. Default is 1. Set objfn.inc = 0 to obtain monotone convergence. Setting objfn.inc = Inf gives a non-monotone scheme. In-between values result in partially-monotone convergence.

kr A non-negative scalar that dictates the degree of non-montonicity. Default is 1. Set kr = 0 to obtain monotone convergence. Setting kr = Inf gives a non-monotone scheme. In-between values result in partially-monotone convergence. This parameter is only used when objfn is not specified by user.

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.

intermed A logical variable denoting whether 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 log-likelihood value. 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 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.
`p.inter`	A matrix where each row corresponds to parameters at each iteration, along with the corresponding log-likelihood value. This object is returned only when the control parameter `intermed` is set to `TRUE`. It is not returned when `objfn` is not specified.

References

R Varadhan and C Roland (2008), Simple and globally convergent numerical schemes for accelerating the convergence of any EM algorithm, *Scandinavian Journal of Statistics*, 35:335-353.

C Roland, R Varadhan, and CE Frangakis (2007), Squared polynomial extrapolation methods with cycling: an application to the positron emission tomography problem, *Numerical Algorithms*, 44:159-172.

Y Du and R Varadhan (2020), SQUAREM: An R package for off-the-shelf acceleration of EM, MM, and other EM-like monotone algorithms, *Journal of Statistical Software*, 92(7): 1-41. <doi:10.18637/jss.v092.i07>

See Also

`fpiter`

Examples

```
##############################################################################
# Also see the vignette by typing:
#  vignette("SQUAREM", all=FALSE)
#
# Example 1:  EM algorithm for Poisson mixture estimation

poissmix.em <- function(p,y) {
# The fixed point mapping giving a single E and M step of the EM algorithm
#
pnew <- rep(NA,3)
i <- 0:(length(y)-1)
zi <- p[1]*exp(-p[2])*p[2]^i / (p[1]*exp(-p[2])*p[2]^i + (1 - p[1])*exp(-p[3])*p[3]^i)
pnew[1] <- sum(y*zi)/sum(y)
pnew[2] <- sum(y*i*zi)/sum(y*zi)
pnew[3] <- sum(y*i*(1-zi))/sum(y*(1-zi))
p <- pnew
return(pnew)
```

```
}

poissmix.loglik <- function(p,y) {
# Objective function whose local minimum is a fixed point
# negative log-likelihood of 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) )
}

# Real 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
tol <- 1.e-08

# Use a preset seed so the example is reproducable.
require("setRNG")
old.seed <- setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion",
    seed=54321))

p0 <- c(runif(1),runif(2,0,4))  # random starting value

# Basic EM algorithm
pf1 <- fpiter(p=p0, y=y, fixptfn=poissmix.em, objfn=poissmix.loglik, control=list(tol=tol))

# First-order SQUAREM algorithm with SqS3 method
pf2 <- squarem(par=p0, y=y, fixptfn=poissmix.em, objfn=poissmix.loglik,
               control=list(tol=tol))

# First-order SQUAREM algorithm with SqS2 method
pf3 <- squarem(par=p0, y=y, fixptfn=poissmix.em, objfn=poissmix.loglik,
               control=list(method=2, tol=tol))

# First-order SQUAREM algorithm with SqS3 method; non-monotone
# Note: the objective function is not evaluated when objfn.inc = Inf
pf4 <- squarem(par=p0,y=y, fixptfn=poissmix.em,
               control=list(tol=tol, objfn.inc=Inf))

# First-order SQUAREM algorithm with SqS3 method;
#  objective function is not specified
pf5 <- squarem(par=p0,y=y, fixptfn=poissmix.em, control=list(tol=tol, kr=0.1))

# Second-order (K=2) SQUAREM algorithm with SqRRE
pf6 <- squarem(par=p0, y=y, fixptfn=poissmix.em, objfn=poissmix.loglik,
               control=list (K=2, tol=tol))

# Second-order SQUAREM algorithm with SqRRE; objective function is not specified
pf7 <- squarem(par=p0, y=y, fixptfn=poissmix.em, control=list(K=2, tol=tol))

# Comparison of converged parameter estimates
par.mat <- rbind(pf1$par, pf2$par, pf3$par, pf4$par, pf5$par, pf6$par, pf7$par)
par.mat
```

```
# Compare objective function values
# (note: `NA's indicate that \code{objfn} was not specified)
c(pf1$value, pf2$value, pf3$value, pf4$value,
  pf5$value, pf6$value, pf7$value)

# Compare number of fixed-point evaluations
c(pf1$fpeval, pf2$fpeval, pf3$fpeval, pf4$fpeval,
  pf5$fpeval, pf6$fpeval, pf7$fpeval)

# Compare mumber of objective function evaluations
# (note: `0' indicate that \code{objfn} was not specified)
c(pf1$objfeval, pf2$objfeval, pf3$objfeval, pf4$objfeval,
  pf5$objfeval, pf6$objfeval, pf7$objfeval)

#################################################################
# Example 2: Same as above (i.e. Poisson mixture)
# but now showing how to "maximize" the log-likelihood

poissmix.loglik.max <- function(p,y) {
# Objective function which is to be *maximized*
# Log-likelihood of 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) )
}

# Maximizing the log-likelihood
# Note:  the control parameter `minimize' is set to FALSE
#
pf.max <- squarem(par=p0, y=y, fixptfn=poissmix.em, objfn=poissmix.loglik.max,
               control=list(tol=tol, minimize=FALSE))
pf.max

#############################################################################
# Example 3:  Accelerating the convergence of power method iteration
#  for finding the dominant eigenvector of a matrix

power.method <- function(x, A) {
# Defines one iteration of the power method
# x = starting guess for dominant eigenvector
# A = a square matrix
ax <- as.numeric(A %*% x)
f <- ax / sqrt(as.numeric(crossprod(ax)))
f
}

# Finding the dominant eigenvector of the Bodewig matrix
b <- c(2, 1, 3, 4, 1,  -3,   1,   5,  3,   1,   6,  -2,  4,   5,  -2,  -1)
bodewig.mat <- matrix(b,4,4)
eigen(bodewig.mat)
```

```
p0 <- rnorm(4)

# Standard power method iteration
ans1 <- fpiter(p0, fixptfn=power.method, A=bodewig.mat)
# re-scaling the eigenvector so that it has unit length
ans1$par <- ans1$par / sqrt(sum(ans1$par^2))
ans1

# First-order SQUAREM with default settings
ans2 <- squarem(p0, fixptfn=power.method, A=bodewig.mat, control=list(K=1))
ans2$par <- ans2$par / sqrt(sum(ans2$par^2))
ans2

# First-order SQUAREM with a smaller step.min0
# Convergence is dramatically faster now!
ans3 <- squarem(p0, fixptfn=power.method, A=bodewig.mat, control=list(step.min0 = 0.5))
ans3$par <- ans3$par / sqrt(sum(ans3$par^2))
ans3

# Second-order SQUAREM
ans4 <- squarem(p0, fixptfn=power.method, A=bodewig.mat, control=list(K=2, method="rre"))
ans4$par <- ans4$par / sqrt(sum(ans4$par^2))
ans4
```

Index