Package 'irlba'

May 17, 2017

Type Package

Title Fast Truncated Singular Value Decomposition and Principal Components Analysis for Large Dense and Sparse Matrices

Version 2.2.1

Date 2017-05-16

Description Fast and memory efficient methods for truncated singular value decomposition and principal components analysis of large sparse and dense matrices.

Depends Matrix

LinkingTo Matrix

Imports stats, methods

License GPL-3

BugReports https://github.com/bwlewis/irlba/issues

RoxygenNote 5.0.1

NeedsCompilation yes

Author Jim Baglama [aut, cph],
Lothar Reichel [aut, cph],
B. W. Lewis [aut, cre, cph]

Maintainer B. W. Lewis <blewis@illposed.net>

Repository CRAN

Date/Publication 2017-05-17 07:28:01 UTC

R topics documented:

irlba *Find a few approximate singular values and corresponding singular vectors of a matrix.*

Description

The augmented implicitly restarted Lanczos bidiagonalization algorithm (IRLBA) finds a few approximate largest (or, optionally, smallest) singular values and corresponding singular vectors of a sparse or dense matrix using a method of Baglama and Reichel. It is a fast and memory-efficient way to compute a partial SVD.

Usage

```
irlba(A, nv = 5, nu = nv, maxit = 100, work = nv + 7, reorth = TRUE,
  tol = 1e-05, v = NULL, right_only = FALSE, verbose = FALSE,
  scale = NULL, center = NULL, shift = NULL, mult = NULL,
  fastpath = TRUE, svtol = tol, smallest = FALSE, ...)
```

Arguments

A	numeric real- or complex-valued matrix or real-valued sparse matrix.
nv	number of right singular vectors to estimate.
nu	number of left singular vectors to estimate (defaults to nv).
maxit	maximum number of iterations.
work	working subspace dimension, larger values can speed convergence at the cost of more memory use.
reorth	if TRUE, apply full reorthogonalization to both SVD bases, otherwise only apply reorthogonalization to the right SVD basis vectors; the latter case is cheaper per iteration but, overall, may require more iterations for convergence. Automatically TRUE when fastpath=TRUE (see below).
tol	convergence is determined when $\|A^T U - VS\| < tol\|A\|$, and when the maximum relative change in estimated singular values from one iteration to the next is less than svtol = tol (see svtol below), where the spectral norm ‖A‖ is approximated by the largest estimated singular value, and U, V, S are the matrices corresponding to the estimated left and right singular vectors, and diagonal matrix of estimated singular values, respectively.
v	optional starting vector or output from a previous run of irlba used to restart the algorithm from where it left off (see the notes).
right_only	logical value indicating return only the right singular vectors (TRUE) or both sets of vectors (FALSE). The right_only option can be cheaper to compute and use much less memory when nrow(A) >> ncol(A) but note that right_only = TRUE sets fastpath = FALSE (only use this option for really large problems that run out of memory and when nrow(A) >> ncol(A)).
verbose	logical value that when TRUE prints status messages during the computation.

scale	optional column scaling vector whose values divide each column of A; must be as long as the number of columns of A (see notes).
center	optional column centering vector whose values are subtracted from each column of A; must be as long as the number of columns of A and may not be used together with the deflation options below (see notes).
shift	optional shift value (square matrices only, see notes).
mult	DEPRECATED optional custom matrix multiplication function (default is %*%, see notes).
fastpath	try a fast C algorithm implementation if possible; set fastpath=FALSE to use the reference R implementation. See the notes for more details.
svtol	additional stopping tolerance on maximum allowed absolute relative change across each estimated singular value between iterations. The default value of this parameter is to set it to tol. You can set svtol=Inf to effectively disable this stopping criterion. Setting svtol=Inf allows the method to terminate on the first Lanczos iteration if it finds an invariant subspace, but with less certainty that the converged subspace is the desired one. (It may, for instance, miss some of the largest singular values in difficult problems.)
smallest	set smallest=TRUE to estimate the smallest singular values and associated singular vectors. WARNING: this option is somewhat experimental, and may produce poor estimates for ill-conditioned matrices.
...	optional additional arguments used to support experimental and deprecated features.

Value

Returns a list with entries:

d: max(nu, nv) approximate singular values

u: nu approximate left singular vectors (only when right_only=FALSE)

v: nv approximate right singular vectors

iter: The number of Lanczos iterations carried out

mprod: The total number of matrix vector products carried out

Note

The syntax of irlba partially follows svd, with an important exception. The usual R svd function always returns a complete set of singular values, even if the number of singular vectors nu or nv is set less than the maximum. The irlba function returns a number of estimated singular values equal to the maximum of the number of specified singular vectors nu and nv.

Use the optional scale parameter to implicitly scale each column of the matrix A by the values in the scale vector, computing the truncated SVD of the column-scaled sweep(A, 2, scale, FUN=`/`), or equivalently, A %*% diag(1 / scale), without explicitly forming the scaled matrix. scale must be a non-zero vector of length equal to the number of columns of A.

Use the optional center parameter to implicitly subtract the values in the center vector from each column of A, computing the truncated SVD of sweep(A, 2, center, FUN=`-`), without explicitly

forming the centered matrix. `center` must be a vector of length equal to the number of columns of `A`. This option may be used to efficiently compute principal components without explicitly forming the centered matrix (which can, importantly, preserve sparsity in the matrix). See the examples.

The optional `shift` scalar valued argument applies only to square matrices; use it to estimate the partial svd of `A + diag(shift, nrow(A), nrow(A))` (without explicitly forming the shifted matrix).

(Deprecated) Specify an optional alternative matrix multiplication operator in the `mult` parameter. `mult` must be a function of two arguments, and must handle both cases where one argument is a vector and the other a matrix. This option is deprecated and will be removed in a future version. The new preferred method simply uses R itself to define a custom matrix class with your user-defined matrix multiplication operator. See the examples.

Use the `v` option to supply a starting vector for the iterative method. A random vector is used by default (precede with `set.seed()` for reproducibility). Optionally set `v` to the output of a previous run of `irlba` to restart the method, adding additional singular values/vectors without recomputing the solution subspace. See the examples.

The function may generate the following warnings:

- "did not converge–results might be invalid!; try increasing maxit or fastpath=FALSE" means that the algorithm didn't converge – this is potentially a serious problem and the returned results may not be valid. `irlba` reports a warning here instead of an error so that you can inspect whatever is returned. If this happens, carefully heed the warning and inspect the result.
- "You're computing a large percentage of total singular values, standard svd might work better!" `irlba` is designed to efficiently compute a few of the largest singular values and associated singular vectors of a matrix. The standard `svd` function will be more efficient for computing large numbers of singular values than `irlba`.
- "convergence criterion below machine epsilon" means that the product of `tol` and the largest estimated singular value is really small and the normal convergence criterion is only met up to round off error.

The function might return an error for several reasons including a situation when the starting vector `v` is near the null space of the matrix. In that case, try a different `v`.

The `fastpath=TRUE` option only supports real-valued matrices and sparse matrices of type `dgCMatrix` (for now). Other problems fall back to the reference R implementation.

References

Augmented Implicitly Restarted Lanczos Bidiagonalization Methods, J. Baglama and L. Reichel, SIAM J. Sci. Comput. 2005.

See Also

`svd`, `prcomp`, `partial_eigen`, `svdr`

Examples

```
set.seed(1)
```

```
A <- matrix(runif(400), nrow=20)
S <- irlba(A, 3)
S$d

# Compare with svd
svd(A)$d[1:3]

# Restart the algorithm to compute more singular values
# (starting with an existing solution S)
S1 <- irlba(A, 5, v=S)

# Estimate smallest singular values
irlba(A, 3, smallest=TRUE)$d

#Compare with
tail(svd(A)$d, 3)

# Principal components (see also prcomp_irlba)
P <- irlba(A, nv=1, center=colMeans(A))

# Compare with prcomp and prcomp_irlba (might vary up to sign)
cbind(P$v,
      prcomp(A)$rotation[, 1],
      prcomp_irlba(A)$rotation[, 1])

# A custom matrix multiplication function that scales the columns of A
# (cf the scale option). This function scales the columns of A to unit norm.
# This approach is deprecated (see below for a bettwe way to do this).
col_scale <- sqrt(apply(A, 2, crossprod))
mult <- function(x, y)
        {
          # check if x is a  vector
          if (is.vector(x))
          {
            return((x %*% y) / col_scale)
          }
          # else x is the matrix
          x %*% (y / col_scale)
        }
irlba(A, 3, mult=mult)$d

# Compare with:
svd(sweep(A, 2, col_scale, FUN=`/`))$d[1:3]

# Compare with the new recommended approach:
setClass("scaled_matrix", contains="matrix", slots=c(scale="numeric"))
setMethod("%*%", signature(x="scaled_matrix", y="numeric"),
   function(x ,y) x@.Data %*% (y / x@scale))
setMethod("%*%", signature(x="numeric", y="scaled_matrix"),
   function(x ,y) (x %*% y@.Data) / y@scale)
a <- new("scaled_matrix", A, scale=col_scale)
irlba(a, 3)$d
```

`partial_eigen` *Find a few approximate largest eigenvalues and corresponding eigenvectors of a symmetric matrix.*

Description

Use `partial_eigen` to estimate a subset of the largest (most positive) eigenvalues and corresponding eigenvectors of a symmetric dense or sparse real-valued matrix.

Usage

```
partial_eigen(x, n = 5, symmetric = TRUE, ...)
```

Arguments

`x`	numeric real-valued dense or sparse matrix.
`n`	number of largest eigenvalues and corresponding eigenvectors to compute.
`symmetric`	`TRUE` indicates `x` is a symmetric matrix (the default); specify `symmetric=FALSE` to compute the largest eigenvalues and corresponding eigenvectors of `t(x) %*% x` instead.
`...`	optional additional parameters passed to the `irlba` function.

Value

Returns a list with entries:

- values n approximate largest eigenvalues
- vectors n approximate corresponding eigenvectors

Note

Specify `symmetric=FALSE` to compute the largest `n` eigenvalues and corresponding eigenvectors of the symmetric matrix cross-product `t(x) %*% x`.

This function uses the `irlba` function under the hood. See `?irlba` for description of additional options, especially the `tol` parameter.

See the RSpectra package https://cran.r-project.org/package=RSpectra for more comprehensive partial eigenvalue decomposition.

References

Augmented Implicitly Restarted Lanczos Bidiagonalization Methods, J. Baglama and L. Reichel, SIAM J. Sci. Comput. 2005.

See Also

`eigen`, `irlba`

Examples

```
set.seed(1)
# Construct a symmetric matrix with some positive and negative eigenvalues:
V <- qr.Q(qr(matrix(runif(100),nrow=10)))
x <- V %*% diag(c(10, -9, 8, -7, 6, -5, 4, -3, 2, -1)) %*% t(V)
partial_eigen(x, 3)$values

# Compare with eigen
eigen(x)$values[1:3]

# Use symmetric=FALSE to compute the eigenvalues of t(x) %*% x for general
# matrices x:
x <- matrix(rnorm(100), 10)
partial_eigen(x, 3, symmetric=FALSE)$values
eigen(crossprod(x))$values
```

`prcomp_irlba` *Principal Components Analysis*

Description

Efficient computation of a truncated principal components analysis of a given data matrix using an implicitly restarted Lanczos method from the `irlba` package.

Usage

```
prcomp_irlba(x, n = 3, retx = TRUE, center = TRUE, scale. = FALSE, ...)
```

Arguments

`x`	a numeric or complex matrix (or data frame) which provides the data for the principal components analysis.
`n`	integer number of principal component vectors to return, must be less than `min(dim(x))`.
`retx`	a logical value indicating whether the rotated variables should be returned.
`center`	a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a centering vector of length equal the number of columns of `x` can be supplied.
`scale.`	a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is `FALSE` for consistency with S, but scaling is often advisable. Alternatively, a vector of length equal the number of columns of `x` can be supplied.
`...`	additional arguments passed to `irlba`.

Value

A list with class "prcomp" containing the following components:

- sdev the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix).
- rotation the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).
- x if `retx` is `TRUE` the value of the rotated data (the centred (and scaled if requested) data multiplied by the `rotation` matrix) is returned. Hence, `cov(x)` is the diagonal matrix `diag(sdev^2)`.
- center, scale the centering and scaling used, or `FALSE`.

Note

The signs of the columns of the rotation matrix are arbitrary, and so may differ between different programs for PCA, and even between different builds of R.

NOTE DIFFERENCES WITH THE DEFAULT `prcomp` FUNCTION! The `tol` truncation argument found in `prcomp` is not supported. In place of the truncation tolerance in the original function, the `prcomp_irlba` function has the argument `n` explicitly giving the number of principal components to return. A warning is generated if the argument `tol` is used, which is interpreted differently between the two functions.

See Also

`prcomp`

Examples

```
set.seed(1)
x  <- matrix(rnorm(200), nrow=20)
p1 <- prcomp_irlba(x, n=3)
summary(p1)

# Compare with
p2 <- prcomp(x, tol=0.7)
summary(p2)
```

`svdr` *Find a few approximate largest singular values and corresponding singular vectors of a matrix.*

Description

The randomized method for truncated SVD by P. G. Martinsson and colleagues finds a few approximate largest singular values and corresponding singular vectors of a sparse or dense matrix. It is a fast and memory-efficient way to compute a partial SVD, similar in performance for many problems to irlba. The svdr method is a block method and may produce more accurate estimations with less work for problems with clustered large singular values (see the examples). In other problems, irlba may exhibit faster convergence.

Usage

```
svdr(x, k, it = 3, extra = 10, center = NULL, Q = NULL)
```

Arguments

x	numeric real- or complex-valued matrix or real-valued sparse matrix.
k	dimension of subspace to estimate (number of approximate singular values to compute).
it	fixed number of algorithm iterations, larger values improve accuracy.
extra	number of extra vectors of dimension ncol(x), larger values generally improve accuracy (with increased computational cost).
center	optional column centering vector whose values are implicitly subtracted from each column of A without explicitly forming the centered matrix (preserving sparsity). Optionally specify center=TRUE as shorthand for center=colMeans(x). Use for efficient principal components computation.
Q	optional initial random matrix, defaults to a matrix of size ncol(x) by k + extra with entries sampled from a normal random distribution.

Value

Returns a list with entries:

d: k approximate singular values

u: k approximate left singular vectors

v: k approximate right singular vectors

mprod: The total number of matrix vector products carried out

References

Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions N. Halko, P. G. Martinsson, J. Tropp. Sep. 2009.

See Also

irlba, svd

Examples

```
set.seed(1)

A <- matrix(runif(400), nrow=20)
svdr(A, 3)$d

# Compare with svd
svd(A)$d[1:3]

# Compare with irlba
irlba(A, 3)$d

## Not run:
# A problem with clustered large singular values where svdr out-performs irlba.
tprolate <- function(n, w=0.25)
{
  a <- rep(0, n)
  a[1] <- 2 * w
  a[2:n] <- sin( 2 * pi * w * (1:(n-1)) ) / ( pi * (1:(n-1)) )
  toeplitz(a)
}

x <- tprolate(512)
set.seed(1)
tL <- system.time(L <- irlba(x, 20))
tR <- system.time(R <- svdr(x, 20))
S <- svd(x)
plot(S$d)
data.frame(time=c(tL[3], tR[3]),
           error=sqrt(c(crossprod(L$d - S$d[1:20]), crossprod(R$d - S$d[1:20]))),
           row.names=c("IRLBA", "Randomized SVD"))

# But, here is a similar problem with clustered singular values where svdr
# doesn't out-perform irlba as easily...clusters of singular values are,
# in general, very hard to deal with!
# (This example based on https://github.com/bwlewis/irlba/issues/16.)
set.seed(1)
s <- svd(matrix(rnorm(200 * 200), 200))
x <- s$u %*% (c(exp(-(1:100)^0.3) * 1e-12 + 1, rep(0.5, 100)) * t(s$v))
tL <- system.time(L <- irlba(x, 5))
tR <- system.time(R <- svdr(x, 5))
S <- svd(x)
plot(S$d)
data.frame(time=c(tL[3], tR[3]),
           error=sqrt(c(crossprod(L$d - S$d[1:5]), crossprod(R$d - S$d[1:5]))),
           row.names=c("IRLBA", "Randomized SVD"))

## End(Not run)
```

Index