Package 'irlba'

October 18, 2017

Type Package

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

Version 2.3.1

Date 2017-10-18

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

Suggests PMA

License GPL-3

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

RoxygenNote 6.0.1

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 2017-10-18 20:15:08 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 work or maxit" 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 may also try setting `fastpath=FALSE`.
- "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

Baglama, James, and Lothar Reichel. "Augmented implicitly restarted Lanczos bidiagonalization methods." SIAM Journal on Scientific Computing 27.1 (2005): 19-42.

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.
col_scale <- sqrt(apply(A, 2, crossprod))
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

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

`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. The value of `scale` determines how column scaling is performed (after centering). If `scale` is a numeric vector with length equal to the number of columns of `x`, then each column of `x` is divided by the corresponding value from `scale`. If `scale` is `TRUE` then scaling is done by dividing the (centered) columns of `x` by their standard deviations if `center=TRUE`, and the root mean square otherwise. If `scale` is `FALSE`, no scaling is done. See `scale` for more details.
`...`	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)
```

`ssvd` *Sparse regularized low-rank matrix approximation.*

Description

Estimate an $\ell 1$-penalized singular value or principal components decomposition (SVD or PCA) that introduces sparsity in the right singular vectors based on the fast and memory-efficient sPCA-rSVD algorithm of Haipeng Shen and Jianhua Huang.

Usage

```
ssvd(x, k = 1, n = 2, maxit = 500, tol = 0.001, center = FALSE,
  scale. = FALSE, alpha = 0, tsvd = NULL, ...)
```

Arguments

x	A numeric real- or complex-valued matrix or real-valued sparse matrix.
k	Matrix rank of the computed decomposition (see the Details section below).
n	Number of nonzero components in the right singular vectors. If k > 1, then a single value of n specifies the number of nonzero components in each regularized right singular vector. Or, specify a vector of length k indicating the number of desired nonzero components in each returned vector. See the examples.
maxit	Maximum number of soft-thresholding iterations.
tol	Convergence is determined when $\|U_j - U_{j-1}\|_F < tol$, where U_j is the matrix of estimated left regularized singular vectors at iteration j.
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. Use center=TRUE to perform a regularized sparse PCA.
scale.	a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. Alternatively, a vector of length equal the number of columns of x can be supplied. The value of scale determines how column scaling is performed (after centering). If scale is a numeric vector with length equal to the number of columns of x, then each column of x is divided by the corresponding value from scale. If scale is TRUE then scaling is done by dividing the (centered) columns of x by their standard deviations if center=TRUE, and the root mean square otherwise. If scale is FALSE, no scaling is done. See scale for more details.
alpha	Optional scalar regularization parameter between zero and one (see Details below).
tsvd	Optional initial rank-k truncated SVD or PCA (skips computation if supplied).
...	Additional arguments passed to irlba.

Details

The ssvd function implements a version of an algorithm by Shen and Huang that computes a penalized SVD or PCA that introduces sparsity in the right singular vectors by solving a penalized least squares problem. The algorithm in the rank 1 case finds vectors u, w that minimize

$$\|x - uw^T\|_F^2 + \lambda\|w\|_1$$

such that $\|u\| = 1$, and then sets $v = w/\|w\|$ and $d = u^T xv$; see the referenced paper for details. The penalty λ is implicitly determined from the specified desired number of nonzero values n. Higher rank output is determined similarly but using a sequence of λ values determined to maintain the desired number of nonzero elements in each column of v specified by n. Unlike standard SVD or PCA, the columns of the returned v when k > 1 may not be orthogonal.

Value

A list containing the following components:

- u regularized left singular vectors with orthonormal columns

- d regularized upper-triangluar projection matrix so that `x %*% v == u %*% d`
- v regularized, sparse right singular vectors with columns of unit norm
- center, scale the centering and scaling used, if any
- lambda the per-column regularization parameter found to obtain the desired sparsity
- iter number of soft thresholding iterations
- n value of input parameter `n`
- alpha value of input parameter `alpha`

Note

Our `ssvd` implementation of the Shen-Huang method makes the following choices:

1. The l1 penalty is the only available penalty function. Other penalties may appear in the future.
2. Given a desired number of nonzero elements in `v`, value(s) for the λ penalty are determined to achieve the sparsity goal subject to the parameter `alpha`.
3. An experimental block implementation is used for results with rank greater than 1 (when `k > 1`) instead of the deflation method described in the reference.
4. The choice of a penalty lambda associated with a given number of desired nonzero components is not unique. The `alpha` parameter, a scalar between zero and one, selects any possible value of lambda that produces the desired number of nonzero entries. The default `alpha = 0` selects a penalized solution with largest corresponding value of `d` in the 1-d case. Think of `alpha` as fine-tuning of the penalty.
5. Our method returns an upper-triangular matrix `d` when `k > 1` so that `x %*% v == u %*% d`. Non-zero elements above the diagonal result from non-orthogonality of the `v` matrix, providing a simple interpretation of cumulative information, or explained variance in the PCA case, via the singular value decomposition of `d %*% t(v)`.

What if you have no idea for values of the argument `n` (the desired sparsity)? The reference describes a cross-validation and an ad-hoc approach; neither of which are in the package yet. Both are prohibitively computationally expensive for matrices with a huge number of columns. A future version of this package will include a revised approach to automatically selecting a reasonable sparsity constraint.

Compare with the similar but more general functions `SPC` and `PMD` in the `PMA` package by Daniela M. Witten, Robert Tibshirani, Sam Gross, and Balasubramanian Narasimhan. The `PMD` function can compute low-rank regularized matrix decompositions with sparsity penalties on both the `u` and `v` vectors. The `ssvd` function is similar to the PMD(*, L1) method invocation of `PMD` or alternatively the `SPC` function. Although less general than `PMD`(*), the `ssvd` function can be faster and more memory efficient for the basic sparse PCA problem. See the examples below for more information.

(* Note that the s4vd package by Martin Sill and Sebastian Kaiser, https://cran.r-project.org/package=s4vd, includes a fast optimized version of a closely related algorithm by Shen, Huang, and Marron, that penalizes both `u` and `v`.)

References

- Shen, Haipeng, and Jianhua Z. Huang. "Sparse principal component analysis via regularized low rank matrix approximation." Journal of multivariate analysis 99.6 (2008): 1015-1034.

- Witten, Tibshirani and Hastie (2009) A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. _Biostatistics_ 10(3): 515-534.

Examples

```
set.seed(1)
u <- matrix(rnorm(200), ncol=1)
v <- matrix(c(runif(50, min=0.1), rep(0,250)), ncol=1)
u <- u / drop(sqrt(crossprod(u)))
v <- v / drop(sqrt(crossprod(v)))
x <- u %*% t(v) + 0.001 * matrix(rnorm(200*300), ncol=300)
s <- ssvd(x, n=50)
table(actual=v[, 1] != 0, estimated=s$v[, 1] != 0)
oldpar <- par(mfrow=c(2, 1))
plot(u, cex=2, main="u (black circles), Estimated u (blue discs)")
points(s$u, pch=19, col=4)
plot(v, cex=2, main="v (black circles), Estimated v (blue discs)")
points(s$v, pch=19, col=4)

# Compare with SPC from the PMA package, regularizing only the v vector and choosing
# the regularization constraint `sum(abs(s$v))` computed above by ssvd
# (they find the about same solution in this "sparse SVD" case):
if (requireNamespace("PMA", quietly = TRUE)) {
  p <- PMA::SPC(x, sumabsv=sum(abs(s$v)), center=FALSE)
  table(actual=v[, 1] != 0, estimated=p$v[, 1] != 0)
  # compare optimized values
  print(c(ssvd=s$d, SPC=p$d))

  # Same example, but computing a "sparse PCA", again about the same results:
  sp <- ssvd(x, n=50, center=TRUE)
  pp <- PMA::SPC(x, sumabsv=sum(abs(sp$v)), center=TRUE)
  print(c(ssvd=sp$d, SPC=pp$d))
}


# Let's consider a trivial rank-2 example (k=2) with noise. Like the
# last example, we know the exact number of nonzero elements in each
# solution vector of the noise-free matrix. Note the application of
# different sparsity constraints on each column of the estimated v.
set.seed(1)
u <- qr.Q(qr(matrix(rnorm(400), ncol=2)))
v <- matrix(0, ncol=2, nrow=300)
v[sample(300, 15), 1] <- runif(15, min=0.1)
v[sample(300, 50), 2] <- runif(50, min=0.1)
v <- qr.Q(qr(v))
x <- u %*% (c(2, 1) * t(v)) + .001 * matrix(rnorm(200 * 300), 200)
s <- ssvd(x, k=2, n=colSums(v != 0))

# Compare actual and estimated vectors:
table(actual=v[, 1] != 0, estimated=s$v[, 1] != 0)
table(actual=v[, 2] != 0, estimated=s$v[, 2] != 0)
```

```
plot(v[, 1], cex=2, main="True v1 (black circles), Estimated v1 (blue discs)")
points(s$v[, 1], pch=19, col=4)
plot(v[, 2], cex=2, main="True v2 (black circles), Estimated v2 (blue discs)")
points(s$v[, 2], pch=19, col=4)
par(oldpar)
```

`summary.irlba_prcomp` *Summary method for truncated pca objects computed by* `prcomp_irlba`.

Description

Summary method for truncated pca objects computed by `prcomp_irlba`.

Usage

```
## S3 method for class 'irlba_prcomp'
summary(object, ...)
```

Arguments

`object`	An object returned by `prcomp_irlba`.
`...`	Optional arguments passed to `summary`.

`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, tol = 1e-05, it = 100L, extra = min(10L, dim(x) - k),
  center = NULL, Q = NULL, return.Q = FALSE)
```

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).
tol	stop iteration when the largest absolute relative change in estimated singular values from one iteration to the next falls below this value.
it	maximum number of algorithm iterations.
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.
return.Q	if TRUE return the Q matrix for restarting (see examples).

Details

Also see an alternate implementation (rsvd) of this method by N. Benjamin Erichson in the https://cran.r-project.org/package=rsvd package.

Value

Returns a list with entries:

d: k approximate singular values

u: k approximate left singular vectors

v: k approximate right singular vectors

mprod: total number of matrix products carried out

Q: optional subspace matrix (when return.Q=TRUE)

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, rsvd in the rsvd package

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