The irlba Package

Bryan W. Lewis
blewis@illposed.net,

adapted from the work of:
Jim Baglama (University of Rhode Island)
and Lothar Reichel (Kent State University).

May 27, 2011

1 Introduction

The `irlba` package provides a fast way to compute partial singular value decompositions (SVD) of large matrices. It is an R implementation of the *implicitly restarted Lanczos bidiagonalization algorithm* of Jim Baglama and Lothar Reichel[1]. The `irlba` package source code is maintained at http://rforge.net/irlba/. The web homepage for the `irlba` package is http://illposed.net/irlba.html. An introductory example using the Netflix prize data set may be found at the web link http://goo.gl/fRech.

The `irlba` package works with regular dense real- and complex-valued R matrices and sparse matrices (provided by the `Matrix` package). It also includes a simple way to work with other matrix classes including `big.matrix` from the `bigmemory` package and others. The `irlba` is both faster and more memory efficient than the usual R `svd` function for computing a few singular vectors and corresponding singular values of a matrix. It may be used to compute a partial SVD corresponding to either the largest or smallest singular values of a matrix. The package exhibits good speed improvements when used with versions of R compiled using high performance linear algebra libraries. Performance is good even with the reference R linear algebra libraries.

We summarize the algorithm and provide a few examples. A much more detailed description and discussion of the algorithm may be found in the cited Baglama-Reichel reference.

[1]Restarted Block Lanczos Bidiagonalization Methods (with L. Reichel) Numerical Algorithms, 43 (2006), pp. 251-272

2 The SVD and Partial SVD

The singular value decomposition of the matrix $A \in \mathbf{R}^{\ell \times n}, \ell \geq n$ may be defined as:

$$A = \sum_{j=1}^{n} \sigma_j u_j v_j^T, \qquad v_j^T v_k = u_j^T u_k = \begin{cases} 1 & \text{if } j = k, \\ 0 & \text{o.w.,} \end{cases}$$

where $u_j \in \mathbf{R}^{\ell}$, $v_j \in \mathbf{R}^n$, $j = 1, 2, \ldots, n$, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. Let $1 \leq k < n$. We define the partial SVD of A to be:

$$A_k \quad := \quad \sum_{j=1}^{k} \sigma_j u_j v_j^T$$

The following simple example shows how to use `irlba` to compute the five largest singular values and corresponding singular vectors of a 5000×5000 matrix. We compare to the usual R `svd` function and report timings for our test machine, an 8-CPU core, 2.0 GHz AMD Opteron server with 16 GB RAM, using R version 2.13.0 compiled with the high performance AMD ACML core math libraries.

```
> library('irlba')
> A <- matrix(rnorm(5000*5000), 5000)
> t1 <- proc.time()
> L <- irlba(A, nu=5, nv=5)
> print(proc.time() - t1)
   user  system elapsed
 41.640   0.470  36.985
> gc()
          used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  137098   7.4     350000  18.7   350000  18.7
Vcells 25180235 192.2 52881183 403.5 52881005 403.5
```

Now, compare with the standard `svd` function:

```
> t1 <- proc.time()
> S <- svd(A, nu=5, nv=5)
> print(proc.time() - t1)
   user  system elapsed
616.035   4.396 187.371
> gc()
          used  (Mb) gc trigger  (Mb)  max used  (Mb)
Ncells  137109   7.4     350000   18.7    350000   18.7
Vcells 25235234 192.6 168397903 1284.8 200272760 1528.0
```

```
# Compare the singular values computed by each method:
> sqrt (crossprod(S$d[1:5]-L$d)/crossprod(S$d[1:5]))
            [,1]
[1,] 1.56029e-12
```

The `irlba` method uses less than one tenth total CPU time as the `svd` method in this example, less than one fifth the total run time, and about one fourth the peak memory.

2.1 Differences with `svd`

The `irlba` function is designed to compute a *partial* singular value decomposition. It is largely compatible with the usual R `svd` function but there are some differences. In particular:

1. The `irlba` function only computes the number of singular values corresponding to the `nu` and `nv` parameters. For example, if 5 singular vectors are desired (`nu=nv=5`), then only the five corresponding singular values are computed. The standard R `svd` function always returns the *total* set of singular values for the matrix, regardless of how many singular vectors are specified.

2. The `irlba` function is an iterative method until either a tolerance or maximum number of iterations is reached. There exists pathological problems for which `irlba` does not converge (see the references for more information). Such problems are not likely to be encountered, but the method will fail with an error after the iteration limit is reached in those cases.

Watch out for the first difference noted above.

2.2 Computing the Smallest Singular Values

The `irlba` function may be used to compute either the largest or smallest singular values (and corresponding singular vectors) of a matrix. The default is to compute the largest singular values. Use the `sigma='ss'` option to compute the smallest values, illustrated below:

```
L <- irlba(A, nu=5, nv=5, sigma='ss')
```

Harmonic Ritz vectors are used by default to augment the Lanczos process when the smallest singular values are desired. See the reference for a discussion of the Lanczos process augmentation strategy.

2.3 User-defined Matrix Operations

The `irlba` function includes a provision for specifying custom matrix operators. Using this feature, `irlba` may be used with the `big.matrix` class from the `bigmemory/bigalgebra` packages, or to compute the partial SVD of matrix-free linear operators, for example.

User-defined matrix operations are specified using the optional `matmul` parameter. If defined, it must be a function that takes three arguments as follows:

```
matmul <- function (A, B, transpose)
{
  if(transpose) return(t(A) %*% B)
  return(A %*% B)
}
```

Replace the above transpose and matrix multiply operations with ones appropriate to your matrix class.

3 A Quick Summary of the IRLBA Method

3.1 Partial Lanczos Bidiagonalization

Start with a given vector p_1. Compute m steps of the Lanczos process:

$$\begin{aligned} AP_m &= Q_m B_m \\ A^T Q_m &= P_m B_m^T + r_m e_m^T, \end{aligned}$$

$B_m \in \mathbf{R}^{m \times m}$, $P_m \in \mathbf{R}^{n \times m}$, $Q_m \in \mathbf{R}^{\ell \times m}$,

$P_m^T P_m = Q_m^T Q_m = I_m$,

$r_m \in \mathbf{R}^n$, $P_m^T r_m = 0$,

$P_m = [p_1, p_2, \ldots, p_m]$.

3.2 Approximating Partial SVD with A Partial Lanczos bidiagonalization

$$\begin{aligned} A^T A P_m &= A^T Q_m B_m \\ &= P_m B_m^T B_m + r_m e_m^T B_m, \end{aligned}$$

$$
\begin{aligned}
AA^TQ_m &= AP_mB_m^T + Ar_me_m^T, \\
&= Q_mB_mB_m^T + Ar_me_m^T.
\end{aligned}
$$

Compute the SVD of B_m:

$$
B_m = \sum_{j=1}^{m} \sigma_j^B u_j^B \left(v_j^B\right)^T .
$$

$$
\left(\text{i.e., } B_m v_j^B = \sigma_j^B u_j^B, \text{ and } B_m^T u_j^b = \sigma_j^B v_j^B.\right)
$$

Define: $\tilde{\sigma}_j := \sigma_j^B, \qquad \tilde{u}_j := Q_m u_j^B, \qquad \tilde{v}_j := P_m v_j^B.$

Then:

$$
\begin{aligned}
A\tilde{v}_j &= AP_mv_j^B \\
&= Q_mB_mv_j^B \\
&= \sigma_j^BQ_mu_j^B \\
&= \tilde{\sigma}_j\tilde{u}_j,
\end{aligned}
$$

and

$$
\begin{aligned}
A^T\tilde{u}_j &= A^TQ_mu_j^B \\
&= P_mB_m^Tu_j^B + r_me_m^Tu_j^B \\
&= \sigma_j^BP_mv_j^B + r_me_m^Tu_j^B \\
&= \tilde{\sigma}_j\tilde{v}_j + r_me_m^Tu_j^B.
\end{aligned}
$$

The part in red above represents the error with respect to the exact SVD. The IRLBA strategy is to iteratively reduce the norm of that error term by augmenting and restarting.

Here is the overall method:

1. Compute the Lanczos process up to step m.
2. Compute $k < m$ approximate singular vectors.
3. Orthogonalize against the approximate singular vectors to get a new starting vector.
4. Continue the Lanczos process with the new starting vector for m more steps.
5. Check for convergence tolerance and exit if met.
6. GOTO 1.

3.3 Sketch of the augmented process...

$$
\begin{aligned}
\bar{P}_{k+1} &:= [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_k, p_{m+1}], \\
A\bar{P}_{k+1} &= [\tilde{\sigma}_1 \tilde{u}_1, \tilde{\sigma}_1 \tilde{u}_2, \ldots, \tilde{\sigma}_k \tilde{u}_k, A p_{m+1}]
\end{aligned}
$$

Orthogonalize Ap_{m+1} against $\{\tilde{u}_j\}_{j=1}^k$: $Ap_{m+1} = \sum_{j=1}^k \rho_j \tilde{u}_j + r_k$.

$$
\begin{aligned}
\bar{Q}_{k+1} &:= [\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_k, r_k/\|r_k\|], \\
\bar{B}_{k+1} &:= \begin{bmatrix} \tilde{\sigma}_1 & & & \rho_1 \\ & \tilde{\sigma}_2 & & \rho_2 \\ & & \ddots & \rho_k \\ & & & \|r_k\| \end{bmatrix}.
\end{aligned}
$$

$$
A\bar{P}_{k+1} = \bar{Q}_{k+1}\bar{B}_{k+1}.
$$