Andreas Stathopoulos

Iterative validation: a scheme for improving the reliability and performance of eigenvalue block iterative solvers

Department of Computer Science
College of William and Mary
Williamsburg
VA 23187-8795
andreas@cs.wm.edu
James, R. McCombs

Computationally intensive applications depend increasingly on Krylov iterative methods to solve large, sparse, eigenvalue problems. If the matrix can be factored the benefits are twofold; First, using iterative methods in shift and invert mode very fast convergence can be achieve toward eigenvalues close to a specified shift. Second, it allows for calculating the matrix inertia for predicting the exact number of eigenvalues within an interval and thus provides assurance that no required eigenpairs have been missed.

In this work, we are interested in cases where the matrix is too dense or too large to factor, and thus the above validation of results is not possible. In such cases, the required eigenvalues are usually tightly clustered or multiple, causing Krylov methods to converge slowly and often miss eigenpairs. Several methods such as (Jacobi-)Davidson, LOBPCG and EIGIFP exploit preconditioning to improve convergence and robustness of eigenvalue iterative codes. Still, however, no assurance is provided that no eigenvalues are missed.

Block Krylov methods work on an orthonormal set of vectors simultaneously instead of one vector. Provided that the initial set of vectors are not deficient in the direction of required eigenvectors, block methods identify all clustered or multiple eigenvalues within the size of the block. Yet, eigenvalues can still be missed beyond the block size. Moreover, knowledge about the problem is required to decide the appropriate block size. Computationally, block methods usually require more floating point operations than single vector methods but they are more cache efficient. For certain block sizes execution time is improved, but such a choice is complicated as it affects the numerics of the problem.

Alternatively, a large number of (even multiple) eigenvalues can be obtained through a stable form of deflation called locking. When an eigenvalue λ_1 converges, all subsequent computations are performed orthogonally to the corresponding eigenvector x_1. This guarantees that the method will converge to a different eigenpair, but not necessarily the next required one. In practice, locking does not miss eigenvalues and it even identifies multiplicities if a tolerance close to machine precision is used. Even though in theory single vector Krylov

methods cannot obtain multiplicities, floating point arithmetic introduces noise in the direction of the invariant subspace that is gradually amplified. In the presence of a very high multiplicity or a high number of multiplicities block methods are more effective. Similarly, with high tolerances, block methods are still robust, while locking may miss or yield inaccurate eigenpairs. Typically, single vector methods with locking are preferred because of their efficiency, reverting to small block sizes when the problem is known to have multiplicities.

To improve confidence on obtained results, eigenvalue practitioners traditionally restart the iterative method with a new random initial guess, locking against all previously computed eigenvectors. If the new eigenvalue is in the required range, the computation must be continued. However, it is neither efficient nor robust to rely on a single vector method to obtain highly clustered or multiple eigenvalues, especially with lower accuracies.

We propose a new technique, which we call *iterative validation*, that acts as a wrapper calling another eigenvalue block iterative method repeatedly until no missed eigenvalues can be identified. The inner method can be any block iterative method, such as block Lanczos or subspace iteration, that implements locking against an externally provided set of vectors. We describe iterative validation assuming the lower `nev` eigenpairs of a symmetric matrix are needed.

1. The user calls the inner block method as (s)he would normally do, specifying the smallest necessary block size (usually one) to obtain efficiently most of the required eigenpairs. After completion, if requested, our validation technique starts from step 2.

2. Using error bounds on the `nev` Ritz values $\lambda_1, \ldots, \lambda_{nev}$, we identify the largest numerical multiplicity obtained thus far, say m.

3. We set the block size equal to $m+1$, and call back the inner block method with $m+1$ random initial guesses, seeking the smallest eigenvalue of the matrix with all previously computed eigenvectors locked.

4. After convergence, we report all $m+1$ lowest eigenpairs in the block $(\mu_1, z_1), \ldots, (\mu_{m+1}, z_{m+1})$. We compute the error bound of μ_1: ϵ_1.

5. If $\mu_1 - \epsilon_1 > \lambda_{nev}$, report the original λ_i eigenvalues and stop.

6. If $\mu_1 < \lambda_{nev}$, an eigenvalue was missed. Insert μ_1 among the eigenvalues λ_i, and dispense with the eigenpair of λ_{nev} [1]. Consider $(z_2, \ldots, z_{m+1})$ as initial guesses. The rest will be random vectors. Go to step 2.

7. if $\lambda_j < \mu_1 - \epsilon_1 < \lambda_{j+1}$, return to the user with a note that the provided tolerance was not sufficient to resolve an eigenvalue between λ_{j+1} and μ_1.

[1] If space is available, it is better to keep the eigenpair λ_{nev} as it will help convergence in the next validation step.

The above technique has several advantages. First, it provides a relatively unobtrusive way to validate results, dramatically improving robustness. If the original code would miss eigenvalues, the additional expense is more than justified. Typically, the validation will be run once for each new class of problems, and switched off afterwards. Second, it provides a dynamic way to fine tune the block size without wasting all previous effort. Third, assuming a multiplicity or a cluster of m eigenvalues, a block size of m would be slower for many required eigenvalues that do not belong to the cluster or the multiplicity. Iterative validation with an original block size of 1 finds first the easier part of the spectrum and it only uses the block size m at the clusters that need it.

We have implemented iterative validation as a Matlab wrapper for irbleigs and lobpcg eigenvalue codes, and as a C wrapper for our block Jacobi-Davidson.