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 and Krylov-like iterative methods to solve large, and often sparse, eigenvalue problems. Some of the most widely used methods include subspace iteration, the Lanczos and the Arnoldi methods. If the matrix can be factored the benefits are twofold; First, using the above methods in shift and invert mode (e.g., the shift-invert Arnoldi) very fast convergence can be achieved toward eigenvalues close to a specified shift. Second, calculating the matrix inertia through an additional factorization allows an exact prediction of the number of eigenvalues within an interval and thus 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, it is also common that the required eigenvalues are tightly clustered or even multiple, causing Krylov methods to converge slowly and often miss eigenpairs. Several methods such as Davidson, 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 just 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, there is no assurance that eigenvalues are not missed beyond the block size. Moreover, knowledge about the problem is required to decide on 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 they improve overall execution time, but such a choice is complicated as it is affected by the numerics of the problem.

Alternatively, a large number of (even multiple) eigenvalues can be obtained through a stable form of deflation called locking. Assume for example a symmetric matrix where the lowest `nev` eigenvalues are required. An iterative method is

run until the lowest eigenvalue λ_1 converges. Then, the corresponding eigenvector x_1 is placed in a special locked array, and all computations in the iterative subspace are performed orthogonally to x_1. This guarantees that the method will converge to a different eigenpair, but not necessarily the next lowest 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 more than one eigenvector per multiplicity, 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 single vector methods with locking are usually not as effective as block methods. Similarly, with high tolerances, block methods still provide a very robust choice, 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 results obtained from some calculation, eigenvalue practitioners traditionally restart the iterative method with a new random initial guess, locking against all previously computed eigenvectors. If the new eigenvalue is inside the required range, the computation has to be continued to obtain the missed eigenvalues. 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. The iterative validation can be described easier assuming the lower `nev` eigenpairs of a symmetric matrix are needed.

1. The first time through, 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 of the inner method, 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, looking for the smallest eigenvalue of the matrix that is deflated from all previously computed eigenvalues. For numerical stability, locking is used for deflation.
4. When the inner method converges, we report not only the first but 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 as the required eigenvalues and stop.
6. If $\mu_1 < \lambda_{nev}$, an eigenvalue was missed. Insert μ_1 in its proper place 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 filled with random vectors. Go back to step 2.
7. if $\lambda_j < \mu_1 - \epsilon_1 < \lambda_{j+1}$, return to the user the above eigenvalues noting that the provided tolerance was not sufficient to resolve an eigenvalue between λ_{j+1} and μ_1.

The above technique has several advantages. First and foremost, it provides a relatively unobtrusive way to validate results, thus dramatically improving robustness. If the original code would miss eigenvalues, the additional user specified 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 other required 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** code. In one extreme example demonstrating robustness, irbleigs with block size of 33 was not able to obtain all 74 multiplicities of matrix BCSSTK16. Using irbleigs with block size of 5, followed by iterative validation with a block size of no more than 33 found all the multiplicities.

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