New Eigen Solver Technology in LS-DYNA

Roger G. Grimes[1], François-Henry Rouet[1]

[1]Ansys, Inc.

1 Introduction

The Linear Algebra Team of Ansys LST has added two new eigen solver technologies for the standard vibration analysis problem of structural mechanics. The first, LOBPCG, is based on iterative solution technology to reduce the cost of the direct solution used by the default eigen solver Lanczos. The second, Fast Lanczos, is a new innovative implementation of the Block Shift and Invert Lanczos algorithm targeting the computation of thousands of eigenmodes with less accuracy for the Noise-Vibration-Harshness (NVH) application.

2 LOBPCG

2.1 The LOBPCG algorithm

Shift-and-invert Lanczos requires "exact" solves with $K - \sigma M$ to maintain the three-term Lanczos recurrence relation at the heart of the algorithm. This is also true of nonsymmetric solvers that rely on the Arnoldi relation, e.g., the Implicitly Restarted Arnoldi Method implemented in ARPACK [1,2]. Preconditioned eigensolvers are algorithms that do not require an exact solve; instead, they only rely on an approximate solution, or *preconditioner*. Obviously, the closer the preconditioner is to an exact solve, the better the chances of fast convergence are.

LOBPCG (Locally Optimal Block Preconditioned Conjugate Gradient) is a preconditioned eigensolver for Hermitian problems based on a Rayleigh quotient minimization technique. The Rayleigh quotient

$$\lambda(u) = \frac{u^T K\, u}{u^T M u}$$

is minimized by eigenvectors corresponding to the smallest eigenvalue, and the value of the Rayleigh quotient is the smallest eigenvalue. The definition can be extended to a block of (smallest) eigenmodes. LOBPCG performs the minimization over a subspace span(U, W, P) where U are the current eigenvector approximations, W are the preconditioned residuals, and P are search directions inspired by the Conjugate Gradients algorithm. Once the subspace is built and is M-orthonormalized, LOBPCG performs a Rayleigh-Ritz procedure, i.e., it builds a projected eigenproblem

$$\widehat{K}\,\widehat{X} = \widehat{M}\,\widehat{X}\,\widehat{\Lambda}$$

where $\widehat{K} = [U\ P\ W]^T\ K\ [U\ P\ W]$ and $\widehat{M} = [U\ P\ W]^T\ M\ [U\ P\ W]$. This is a small eigenvalue problem of size 3m x 3m with m the number of requested modes. The eigenvalues of this small eigenproblem (so-called Ritz values) are the candidate eigenvalues for the original problem, and the eigenvectors are used to build new search directions and new candidate eigenvectors for the original problem. The process is repeated until the residuals fall under a user-given tolerance.

To compute large numbers of modes, it is recommended to compute them in blocks and use shifts, just like with Lanczos. However, the difference here is that Lanzcos has access to the inertia of the shifted matrix; this allows Lanczos to count precisely eigenvalues left and right of the shift, and to guarantee that no mode is missed. LOBPCG can potentially miss some modes in some complicated situations, although this has not happened in our testing.

2.2 Implementation in LS-DYNA

LOBPCG was released in LS-DYNA R11 for SMP and LS-DYNA R13 for MPP. One of the most important implementation aspects is the choice of the preconditioner. Our choice is the Block Low-Rank (BLR) mode of MUMPS. MUMPS (MUltifrontal Massively Parallel Solver) [4] is a sparse direct solver with a preconditioning mode based on low-rank approximations [5]. Dense matrices that appear throughout the factorization process are compressed (i.e., represented as a product of smaller matrices) following a dropping criterion chosen by the user. When the dropping criterion is close to machine precision, the factorization is almost exact; when one increases the dropping criterion, one gets an

approximate factorization that uses less memory and is faster to compute and use. As we show in the experimental section, there is a sweet spot between accuracy and performance.

The choice of this preconditioner (an accelerated direct solver) is motivated by the fact that our mechanical models are particularly challenging for iterative solvers and preconditioners. Preconditioners like diagonal scaling or simple incomplete factorizations almost never lead to convergence for our models. Multigrid solvers can be more successful, but they are notoriously hard to parametrize and require providing lots of information about the problem. MUMPS-BLR is fully algebraic (requiring only the matrices and vectors) and easy to use. The BLR mode comes with strong approximation guarantees [6].

In the MPP, the matrices and blocks of vectors are distributed (respectively by columns and rows). The preconditioner uses the MPP (or Hybrid Parallel) mode of MUMPS. Matrix-vector products with K and M are performed with a careful implementation that reduces communications and does not use any global vectors. Orthogonalization of blocks of vectors is done using the Cholesky QR algorithm:

- $A = X^T X$
- $R = \text{Cholesky}(A)$ (replicated on all processors)
- $Q = XR^{-1}$

This is very efficient in MPP but is prone to numerical instability; in case of loss of orthogonality, we revert to a more careful algorithm. Finally, solving the projected eigenproblem (Rayleigh-Ritz procedure) is done in a replicated way by all the processors; this is cheap and reduces communications.

2.3 Results

2.3.1 Tuning the Block Low-Rank preconditioner

Fig. 1 shows experiments for a 9 million element electric pick-up model. This is a typical car model, with multiple kinds of elements (solids, shells, beams...), over 30 material models (metals, glass, foams...), and many types of constraints and boundary conditions (rigid bodies, contacts, spotwelds...). These models are known to be particularly difficult for iterative solvers. The figure shows performance as a function of the BLR compression threshold epsilon when computing 50 modes with a single shift (factorization). The bars show the three distinct phases of the preconditioner (analysis, factorization, and triangular solve) and the rest of the LOBPCG algorithm. Setting the tolerance to 10^{-6} provides the best performance; we get a speed-up of 1.8x compared to using an exact solve at each iteration ("FR", full rank). Pushing the compression threshold further leads to lost convergence. Based on our testing, 10^{-6} seems to be the sweet spot for a wide range of problems.

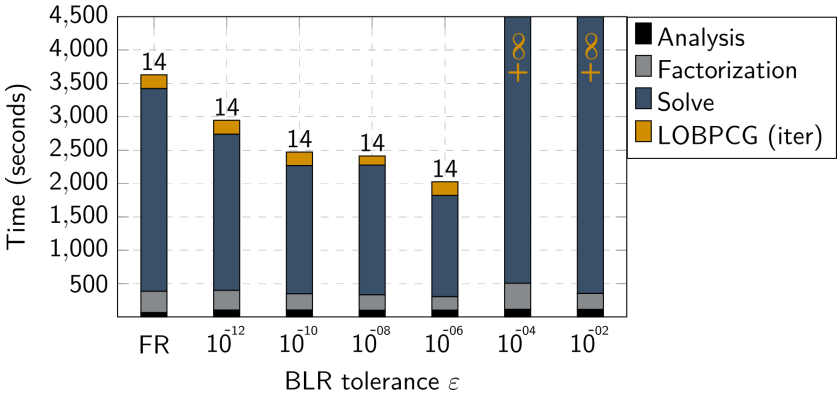


Fig.1: LOBPCG performance as a function of the BLR compression threshold, computing 50 modes.

2.3.2 Scalability

Fig. 2 shows a strong scaling experiment for the same model. We get a speed-up of about 2.3 when going from 32 to 256 cores. The limiting factor is the triangular solve (blue bars in the figure).

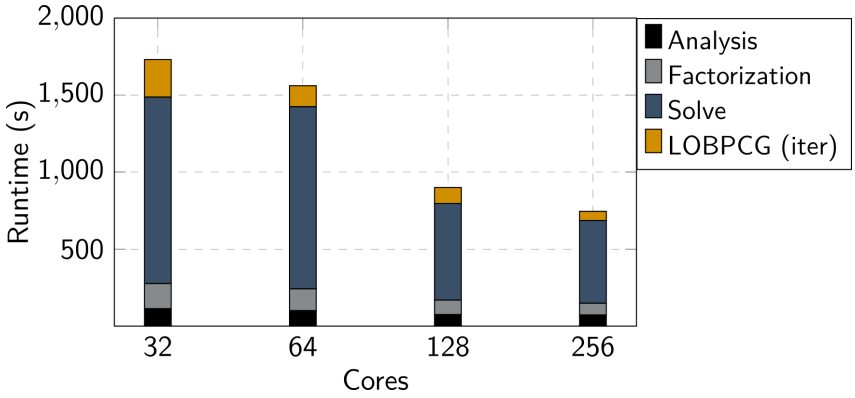


Fig.2: LOBPCG strong scaling experiment, computing 50 modes.

2.3.3 Comparison against Block Lanczos

Fig. 3 shows a comparison of LOBPCG and Block Lanczos for two different problems. The results are very problem dependent. LOBPCG is usually fast for small numbers of modes thanks to the lower startup cost (BLR factorization for LOBPCG, exact factorization for Lanczos) but the cutoff point where Lanczos becomes faster than LOBPCG, if it exists, depends on several parameters, like how much acceleration BLR compression provides, convergence speed, numbers of shifts... There is no clear winner.

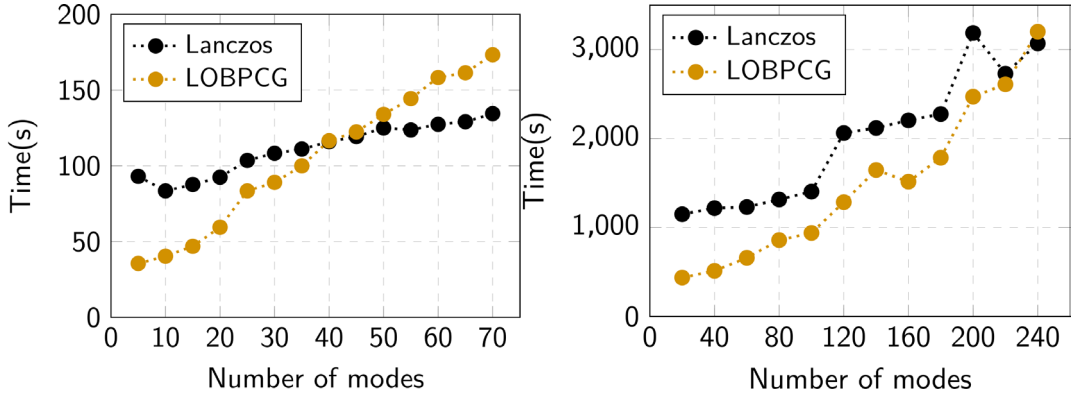


Fig.3: LOBPCG vs Lanczos. Left: 7M dof car model; right: 12M dof car battery pack assembly.

2.3.4 "Fast" mode for NVH-type calculations

The two main parameters of the algorithms are the convergence tolerance of LOBPCG and the dropping criterion (compression threshold) of the preconditioner. For analysts who want to compute large numbers of modes but only require a few digits of accuracy, these two tolerances can be increased to improve speed at the cost of accuracy. The example in Table 1 is a 2.8 million element model of an electric sedan car (body in white with battery packs) for which we compute the 2,000 lowest modes. We relax the BLR compression threshold to 1e-5 and the LOBPCG convergence criterion to 1e-8. The results show that we get a 2X speed-up compared to the default settings. The eigenvectors and eigenvalues we get have less than 0.01% error across the spectrum.

Another interesting observation is that contrary to the results in the previous sections, the preconditioner does not clearly dominate the total cost. When computing many modes, maintaining orthogonality of the (large) trial subspace becomes the bottleneck. This is what constitutes most of the "LOBPCG" column of the table.

Mode	Analysis(s)	Factorization(s)	Solve(s)	LOBPCG(s)	Total time(s)	#iterations
Default	Default	1042.1	5702.3	4448.7	11220.0	659
"Fast"	"Fast"	920.0	2517.5	2376.1	5840.8	231

Table 1: Standard mode vs. "Fast" mode for an NVH-type calculation (2000 modes).

2.4 Future work

Our implementation of LOBPCG is still being improved. We want to look at better orthogonalization algorithms. We also want to look at reducing data movement between our distribution of the matrices and vectors and the internal distribution that MUMPS uses. Finally, we want to consider nonsymmetric versions of the algorithm.

3 Fast Lanczos

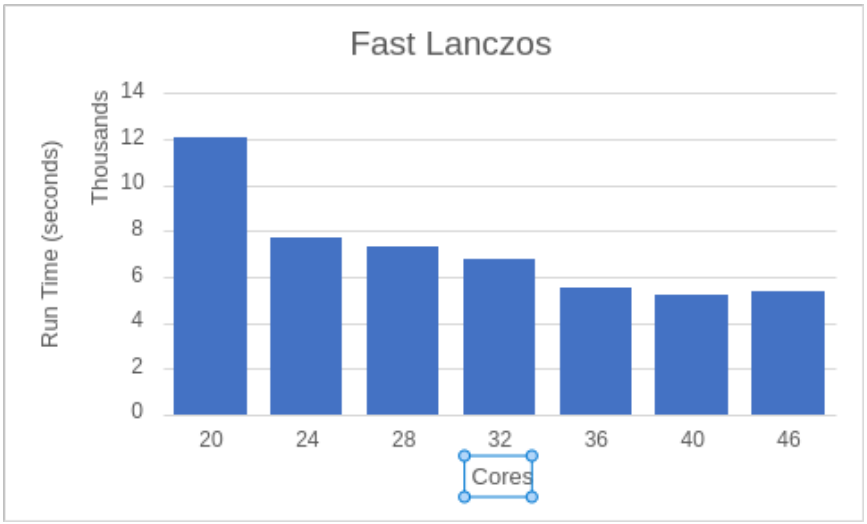
The lead eigen solver technologist at Ansys LST does not like the AMLS algorithm used by many of our competitors to compute thousands of eigenmodes for the NVH application. The accuracy of the AMLS computed eigenvalues deep into the spectrum degrade at each step of the algorithm to the point of almost a complete loss of significance when compared to the more accurately computed eigenmodes from the Block Shift and Invert Lanczos algorithm. The approximation at each step of the AMLS algorithm causes the eigenvalues to increase leading to this degradation. Furthermore, there is very little parallel speed-up available in the algorithm leading to only SMP implementations that use only a small number of CPUs on current computational systems.

After several years of experimentation and testing we have developed an implementation of the Block Shift and Invert Lanczos algorithm for this application which we call Fast Lanczos. Our development goals were to

1. Keep the accuracy of the computed eigenvalues to a 1% relative error even deep into the spectrum.
2. Have an efficient and scalable implementation for distributed memory computing.
3. Match or beat the performance of the competitor's s/w based on AMLS.
4. Execute on standard well equipped workstations, that is, not dependent on high-end computational clusters.

We need to emphasize the last point. The development and testing of the Fast Lanczos eigen solver has all been based on a Intel dual socket installation with two XEON Platinum 8260 chips providing 48 cores. It has 1.58 Tbytes of RAM and a 5.6 Tbytes SSD based Raid system for disk storage. We were told that this workstation cost $33K in 2021, far less than the cost of the automobiles it will be used to design.

Using the same 9 million element electric pick-up model used for the LOBPCG performance testing in the Section 2,3 we computed 1200 eigenmodes using a range of MPI ranks.



This chart shows the scalability of the Fast Lanczos implementation. The algorithm depends on the sparse direct multifrontal solver (MF2) used in both LS-DYNA and Ansys MAPDL which is highly

scalable. The orthogonality computations required in the Fast Lanczos algorithm are also highly scalable. We expect to be able to hold the factorization and Krylov space in memory. With the 9 million element model the memory requirement in 0.8 Tbytes. We estimate we can go to 15 million elements on this computer. We have compared the eigenmodes computed with Fast Lanczos with those computed by the more accurate Lanczos implementation. We are achieving our goal of 1% relative accuracy with higher accuracy in most areas of the spectrum.

We have anecdotal evidence that Fast Lanczos is competitive with the competition software based on the AMLS algorithm. We would welcome production models that customers of Ls-DYNA are using for NVH modeling with both LS-DYNA and competitors software so we can do a fair and equitable comparison of the technologies.

The implementation depends on user parameters for Lanczos algorithm such as recurrence block size, the number of block iterations per shift, and the value of the first shift. The conference presentation and keyword manual will provide information on usage.

4 Summary

Both LOBPCG and Fast Lanczos will be in the production release R15 of LS-DYNA via the *CONTROL_IMPLICIT_EIGENVALUE keyword. As both implementations are actively being developed at the time of the writing of this paper we advise contacting the authors at Roger.Grimes@ansys.com (Fast Lanczos) or Francois-Henry.Rouet@ansys.com (LOBPCG) for usage information. And, of course, to expose the authors to your application needs.

5 Literature

[1] Sorensen, D. C., “Implicit application of polynomial filters in a k-step Arnoldi method”, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 357–385.

[2] Lehoucq, R. B., Sorensen, D. C., and Yang, C., “ARPACK Users’ Guide”, Society for Industrial and Applied Mathematics, 1998.

[3] Knyazev, A. V., “Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method”, SIAM Journal on Scientific Computing, 23 (2001), pp. 517–541.

[4] Amestoy, P. R., Duff, I. S., L’Excellent, J.-Y., and Koster, J., “A fully asynchronous multifrontal solver using distributed dynamic scheduling”, SIAM Journal on Matrix Analysis and Applications (SIMAX), 23 (2001), pp. 15–41.

[5] Amestoy, P. R., Buttari, A., L’Excellent, J.-Y. and Mary, T., “Performance and scalability of the block low-rank multifrontal factorization on multicore architectures”, ACM Transactions on Mathematical Software (TOMS), 45 (2019), pp. 1–26.

[6] Higham, N. J. and Mary, T., “Solving block low-rank linear systems by LU factorization is numerically stable”, IMA Journal of Numerical Analysis, 42 (2022), pp. 951–980.