Implementations of locally optimal block preconditioned conjugate gradient (LOBPCG) eigensolvers

Group seminar

Nicolas Venkovic

Technical University of Munich
Group of Computational Mathematics

April 16, 2025

Outline

Methods based on optimization

Extremal generalized eigenvalue problem

- **Generalized eigenvalue problem**:

$$\text{Find } (x, \lambda) \text{ such that } Ax = \lambda Bx$$

A is symmetric (or Hermitian), and B is symmetric positive definite (SPD)

- When only a few extremal eigenvalues are wanted and A and B are very large and/or applied matrix-free, it is customary to resort to iterative eigensolvers:
  - methods based on projection (Krylov)
  - inherently preconditioned methods (Jacobi-Davidson)
  - methods based on contour integration and polynomial filters (FEAST)
  - methods based on constrained optimization (**LOBPCG**)
- **Characterization of the extremal generalized eigenpair**:

We are looking for an extremal (min or max) generalized eigenpair (λ, x).

Since B is SPD, it admits a Cholesky decomposition $B = LL^T$.

Let the generalized Rayleigh quotient of (A, B) be given by

$$\rho(x) = \frac{x^T Ax}{x^T Bx} \quad \text{for} \quad x^T Bx > 0$$

Characterization of the extremal generalized eigenpair

Making the substitution $y := L^T x$, the generalized Rayleigh quotient becomes the standard Rayleigh quotient of $L^{-1}AL^{-T}$:

$$\rho = \frac{x^T A x}{x^T B x} = \frac{(L^{-T}y)^T A (L^{-T}y)}{(L^{-T}y)^T B (L^{-T}y)} = \frac{y^T L^{-1} A L^{-T} y}{y^T y}$$

Since $L^{-1}AL^{-T}$ is symmetric, the Courant-Fischer theorem implies that the extremum of ρ is the extremal eigenvalue λ of $L^{-1}AL^{-T}$.

Then, since we have

$$\begin{aligned} L^{-1}AL^{-T}y &= \lambda y \\ L^{-1}AL^{-T}L^T x &= \lambda L^T x \\ Ax &= \lambda L L^T x \\ Ax &= \lambda B x \end{aligned}$$

the extremum value λ of $\rho(x)$ is the extremal eigenvalue of the generalized eigenvalue problem $Ax = \lambda Bx$, achieved with the general eigenvector x.

- Finding an extremal generalized eigen-pair of (A, B) is equivalent to an optimization problem of the generalized Rayleigh quotient $\frac{x^T A x}{x^T B x}$.

Approach by steepest descent

- Approaches based on the optimization of the quotient $\rho(x) = \frac{x^T Ax}{x^T Bx}$ generate a sequence $x_0, x_1, \ldots$ of approximate eigenvectors based on a given recurrence formula.
- Note that the gradient of $\rho(x)$ is given by:

$$\nabla\rho(x) = \frac{2}{x^T Bx}(Ax - \rho(x)Bx) = \frac{2}{x^T Bx}r(x) \propto r(x)$$

  where $r(x) = Ax - \rho(x)Bx$ is the generalized eigen-residual.
- Then, the steepest descent iteration is of the form

$$x_{i+1} = x_i - \alpha_i \nabla\rho(x_i)$$

  where α_i is a step size, optimally chosen to minimize $\rho(x_{i+1})$.

  In other words, the iterate x_{i+1} is searched in a subspace of $\mathrm{span}\{x_i, r_i\}$ where $r_i = Ax_i - \rho(x_i)Bx_i$.
- In order to accelerate convergence, a preconditioner T may be chosen and applied to r_i, in which case the next iterate is searched in $\mathrm{span}\{x_i, z_i\}$ where $z_i = Tr_i$.

From steepest descent to locally optimal conjugate gradient

- **Limitations of steepest descent**:
  - Slow convergence, especially for ill-conditioned problems
  - Limited to 2-dimensional search space, i.e., $x_{i+1} \in \text{span}\{x_i, Tr_i\}$

Locally optimal preconditioned conjugate gradient (LOPCG) method

The recurrence formula for LOPCG is

$$x_{i+1} = \alpha_i x_i + \beta_i p_i + \gamma_i T r_i$$

where p_i is a search direction, A-conjugate to the previous direction p_{i-1}.

- LOPCG is called "Conjugate Gradient" because:
  1. It uses conjugate directions p_i similar to the CG algorithm
  2. It relies on a recurrence formula similar to that of the CG algorithm
- "Locally Optimal" refers to the fact that the search space of each iterate is composed of three directions, namely

$$\text{span}\{x_i, p_i, Tr_i\} = \text{span}\{x_i, x_{i-1}, Tr_i\}$$

- The iterate x_{i+1} is formed by Rayleigh Ritz projection to optimize $\rho(x_{i+1})$.

Rayleigh-Ritz projection of generalized eigenvalue problems

- Rayleigh-Ritz projections are a means to find an optimal iterate in a given search space, e.g., $\mathrm{span}\{x_i, x_{i-1}, Tr_i\}$ in the case of LOPCG.
- Rayleigh-Ritz projection of the generalized eigenvalue problem in a search space $\mathcal{R}(V)$ for some full rank $V \in \mathbb{R}^{n\times m}$ with $m \leq n$:

$$\text{Find } (x_{i+1}, \lambda) \in \mathcal{R}(V) \times \mathbb{R} \text{ s.t. } (Ax_{i+1} - \lambda Bx_{i+1}) \perp \mathcal{R}(V).$$

Rayleigh-Ritz procedure with respect to $\mathcal{R}(V)$

$$\begin{aligned} \text{RR} : \mathbb{S}^n_* \times \mathbb{S}^n_{++} \times \mathbb{R}^{n\times m}_* &\to \mathbb{R}^m \times \mathbb{R} \\ (A, B, V) &\mapsto (\hat{x}, \lambda) \text{ s.t. } \boxed{V^T AV\hat{x} = V^T BV\hat{x}\,\lambda} \end{aligned}$$

where λ is an extremal **eigenvalue** of the projected problem with the corresponding eigenvector $\hat{x}$.

Then, the Rayleigh-Ritz vector is formed by $x_{i+1} := V\hat{x}$.

By convention, $\hat{x}^T V^T BV\hat{x} = 1$ so that $x_{i+1}^T Bx_{i+1} = 1$ and $x_{i+1}^T Ax_{i+1} = \theta$.

Early LOPCG iteration

The early form of LOPCG iteration is given by:

Early_LOPCG(A, B, x_{-1}, T^{-1}):

$(\hat{x}_0, \lambda_0) \hookleftarrow \mathsf{RR}(A, B, x_{-1})$

$x_0 := x_{-1}\hat{x}_0$; $r_0 := Ax_0 - Bx_0\lambda_0$

for $i = 0, 1, \ldots$ **do**

$z_i := Tr_i$

if $i == 0$ **then** $V_{i+1} := [x_i, z_i]$ **else** $V_{i+1} := [x_i, z_i, x_{i-1}]$

$(\hat{x}_{i+1}, \lambda_{i+1}) \hookleftarrow \mathsf{RR}(A, B, V_{i+1})$

$x_{i+1} := V_{i+1}\hat{x}_{i+1}$; $r_{i+1} := Ax_{i+1} - Bx_{i+1}\lambda_{i+1}$

Early development of LOBPCG iterations

Locally optimal block preconditioned conjugate gradient

- We now search for $X \in \mathbb{R}^{n \times k}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$ for $k \leq n$ s.t.
$$AX = BX\Lambda$$
where $X^T BX = I_k$ and $\lambda_1, \ldots, \lambda_k$ are extremal generalized eigenvalues of the pencil (A, B).
- When $k \ll n$ and n is very large and/or the application of the operators A and B is matrix-free, it is customary to resort to iterative eigensolvers.
- The least dominant generalized eigenvectors in the columns of X such that $AX = BX\Lambda$ can be defined as the minimizer of the trace$(X^T AX)$ subjected to the constraint $X^T BX = I_k$.
- Knyazev (2001) introduced LOBPCG by extending LOPCG to a block version, which simultaneously produces iterates for the approximation of multiple dominant eigen-pairs.
- Given an initial iterate X_0, LOBPCG generates a sequence of iterates $X_1, X_2, \ldots$ which, in their earliest form, are obtained by Rayleigh-Ritz projection in locally optimal subspaces $\mathcal{R}([X_i, TR_i, X_{i-1}])$, where the columns of R_i are eigen-residuals, and T is a preconditioner.

Knyazev, A. V. (2001). Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM journal on scientific computing, 23(2), 517-541.

Rayleigh-Ritz projection of generalized eigenvalue problems

- Rayleigh-Ritz projections are an essential step of the definition of iterates generated by LOBPCG.
- Rayleigh-Ritz projection of the generalized eigenvalue problem in $\mathcal{R}(V)$ for some full rank $V \in \mathbb{R}_*^{n\times m}$ with $m \leq n$:

$$\text{Find } (Y, \Theta) \in \mathcal{R}(V) \times \mathbb{R}_*^{k\times k} \text{ s.t. } (AY - BY\Theta) \perp \mathcal{R}(V)$$

where $k \leq m \leq n$ and Θ is diagonal.

Rayleigh-Ritz procedure with respect to $\mathcal{R}(V)$

$$\text{RR} : \mathbb{S}_*^n \times \mathbb{S}_{++}^n \times \mathbb{R}_*^{n\times m} \times \mathbb{N}^+ \to \mathbb{R}_*^{m\times k} \times \mathbb{R}_*^{k\times k}$$

$$(A, B, V, k) \mapsto (\hat{Y}, \Theta) \text{ s.t. } \boxed{V^T AV\hat{Y} = V^T BV\hat{Y}\Theta}$$

where $\Theta = \operatorname{diag}(\theta_1, \ldots, \theta_k)$ consists of $k \leq m \leq n$ **extremal eigenvalues** of the projected problem with corresponding eigenvectors in the columns of $\hat{Y}$.

Then, Rayleigh-Ritz vectors are formed by the columns of $Y := V\hat{Y}$.

By convention, $\hat{Y}^T V^T BV\hat{Y} = I_k$ so that $Y^T BY = I_k$ and $Y^T AY = \Theta$.

Definition of Early_LOBPCG iterations

- In their earliest form, LOBPCG iterations are defined by Knyazev (2001) as Rayleigh-Ritz projections w.r.t. $\mathcal{R}([X_i, Z_i, X_{i-1}])$. We refer to these as

Early_LOBPCG(A, B, X_{-1}, T^{-1}, k):

$\triangleright\, X_{-1} \in \mathbb{R}_*^{n\times m}$, $k \le m \le n$

$(\hat{X}_0, \Lambda_0) \hookleftarrow \mathsf{RR}(A, B, X_{-1}, m)$

$X_0 := X_{-1}\hat{X}_0$; $R_0 := AX_0 - BX_0\Lambda_0$

for $i = 0, 1, \ldots$ **do**

  $Z_i := T^{-1}R_i$

  if $i == 0$ **then** $V_{i+1} := [X_i, Z_i]$ **else** $V_{i+1} := [X_i, Z_i, X_{i-1}]$

  $(\hat{X}_{i+1}, \Lambda_{i+1}) \hookleftarrow \mathsf{RR}(A, B, V_{i+1}, m)$

  $X_{i+1} := V_{i+1}\hat{X}_{i+1}$; $R_{i+1} := AX_{i+1} - BX_{i+1}\Lambda_{i+1}$

- As Early_LOBPCG converges, $[X_i, Z_i, X_{i-1}]$ becomes ill-conditioned which, when relying on finite arithmetic, leads to error propagation through the Rayleigh-Ritz procedure, eventually making the method unstable.

Knyazev, A. V. (2001). Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM journal on scientific computing, 23(2), 517-541.

Basic_LOBPCG iterations (Knyazev, 2001)

Definition of Basic_LOBPCG iterations

- To circumvent the stability issue of Early_LOBPCG, Knyazev (2001) proposes to improve the conditioning of the matrices $V_2, V_3, \ldots$, while preserving their ranges. We refer to these iterations as Basic_LOBPCG.
- We denote the variables of Basic_LOBPCG by $\widetilde{X}_i, \widetilde{Z}_i, \ldots$. Then,
  - The iterates for $i = 0$ are the same as in Early_LOBPCG:

$$(\hat{\widetilde{X}}_0, \widetilde{\Lambda}_0) \leftarrow \mathsf{RR}(A, B, X_{-1}, m) \implies \hat{\widetilde{X}}_0 = \hat{X}_0,\ \widetilde{\Lambda}_0 = \Lambda_0$$

$$\widetilde{X}_0 := X_{-1}\hat{\widetilde{X}}_0 \implies \widetilde{X}_0 = X_0$$

$$\widetilde{R}_0 := A\widetilde{X}_0 - B\widetilde{X}_0\widetilde{\Lambda}_0\,;\ \widetilde{Z}_0 := T^{-1}R_0 \implies \widetilde{R}_0 = R_0,\ \widetilde{Z}_0 = Z_0$$

  - The iterates for $i = 1$ are also the same as in Early_LOBPCG:

$$\widetilde{V}_1 := [\widetilde{X}_0, \widetilde{Z}_0] \implies \widetilde{V}_1 = V_1$$

$$(\hat{\widetilde{X}}_1, \widetilde{\Lambda}_1) \leftarrow \mathsf{RR}(A, B, \widetilde{V}_1, m) \implies \hat{\widetilde{X}}_1 = \hat{X}_1,\ \widetilde{\Lambda}_1 = \Lambda_1$$

$$\widetilde{X}_1 := \widetilde{V}_1\hat{\widetilde{X}}_1 = \widetilde{X}_0\hat{\widetilde{X}}_{1|\widetilde{X}_0} + \widetilde{Z}_0\hat{\widetilde{X}}_{1|\widetilde{Z}_0} \implies \widetilde{X}_1 = X_1 = X_0\hat{X}_{1|X_0} + Z_0\hat{X}_{1|Z_0}$$

$$\widetilde{R}_1 := A\widetilde{X}_1 - B\widetilde{X}_1\widetilde{\Lambda}_1\,;\ \widetilde{Z}_1 := T^{-1}\widetilde{R}_1 \implies \widetilde{R}_1 = R_1,\ \widetilde{Z}_1 = Z_1$$

Knyazev, A. V. (2001). Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM journal on scientific computing, 23(2), 517-541.

Definition of Basic_LOBPCG iterations, cont'd$_1$

- Then, blocks of search directions are introduced, and used to define $\widetilde{V}_{i+1}$. For $i = 1$, this is done by

$$\begin{aligned}
\widetilde{P}_1 := \widetilde{Z}_0 \hat{\widetilde{X}}_{1|\widetilde{Z}_0} \implies \widetilde{P}_1 &= \widetilde{X}_1 - \widetilde{X}_0 \hat{\widetilde{X}}_{1|\widetilde{X}_0} = X_1 - X_0 \hat{X}_{1|X_0} \\
\widetilde{V}_2 := [\widetilde{X}_1, \widetilde{Z}_1, \widetilde{P}_1] \implies \mathcal{R}(\widetilde{V}_2) &= \mathcal{R}([\widetilde{X}_1, \widetilde{Z}_1, \widetilde{P}_1]) \\
&= \mathcal{R}([X_1, Z_1, X_1 - X_0 \hat{X}_{1|X_0}]) \\
&= \mathcal{R}([X_1, Z_1, X_0]) \\
&= \mathcal{R}(V_2)
\end{aligned}$$

- Consequently, the iterates for $i = 2$ are such that

$$\begin{aligned}
(\hat{\widetilde{X}}_2, \widetilde{\Lambda}_2) \leftarrow \mathsf{RR}(A, B, \widetilde{V}_2, m) &\implies \widetilde{X}_2 = X_2,\ \widetilde{\Lambda}_2 = \Lambda_2 \\
\widetilde{R}_2 := A\widetilde{X}_2 - B\widetilde{X}_2\widetilde{\Lambda}_2\,;\ \widetilde{Z}_2 := T^{-1}\widetilde{R}_2 &\implies \widetilde{R}_2 = R_2,\ \widetilde{Z}_2 = Z_2 \\
\widetilde{P}_2 := \widetilde{Z}_1 \hat{\widetilde{X}}_{2|\widetilde{Z}_1} + \widetilde{P}_1 \hat{\widetilde{X}}_{2|\widetilde{P}_1} &\implies \widetilde{P}_2 = \widetilde{X}_2 - \widetilde{X}_1 \hat{\widetilde{X}}_{2|\widetilde{X}_1} \\
&= X_2 - X_1 \hat{\widetilde{X}}_{2|\widetilde{X}_1}
\end{aligned}$$

Knyazev, A. V. (2001). Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM journal on scientific computing, 23(2), 517-541.

Definition of Basic_LOBPCG iterations, cont'd$_2$

- Then, the iterates for $i = 3$ are such that

$$\widetilde{V}_3 := [\widetilde{X}_2, \widetilde{Z}_2, \widetilde{P}_2] \implies \mathcal{R}(\widetilde{V}_3) = \mathcal{R}([\widetilde{X}_2, \widetilde{Z}_2, \widetilde{P}_2])$$
$$= \mathcal{R}([X_2, Z_2, X_2 - X_2\hat{\widetilde{X}}_{3|\widetilde{X}_0}])$$
$$= \mathcal{R}([X_2, Z_2, X_1]) = \mathcal{R}(V_3)$$

$$(\hat{\widetilde{X}}_3, \widetilde{\Lambda}_3) \leftarrow \mathsf{RR}(A, B, \widetilde{V}_3, m) \implies \widetilde{X}_3 = X_3,\ \widetilde{\Lambda}_3 = \Lambda_3$$

$$\vdots$$

- In general, for all $i > 0$, we have

$$\widetilde{P}_{i+1} := \widetilde{Z}_i\hat{\widetilde{X}}_{i+1|\widetilde{Z}_i} + \widetilde{P}_i\hat{\widetilde{X}}_{i+1|\widetilde{P}_i} = X_{i+1} - X_i\hat{\widetilde{X}}_{i+1|\widetilde{X}_i}$$

  so that $\mathcal{R}(\widetilde{V}_{i+1}) = \mathcal{R}(V_{i+1})$ which, in turn, implies $\widetilde{X}_{i+1} = X_{i+1}$, $\widetilde{\Lambda}_{i+1} = \Lambda_{i+1}$, $\widetilde{R}_{i+1} = R_{i+1}$ and $\widetilde{Z}_{i+1} = Z_{i+1}$.
- So, in exact arithmetic, the iterates of Basic_LOBPCG are equivalent to those of Early_LOBPCG, the difference being that, when nearing convergence, $\widetilde{V}_{i+1}$ is presumably better conditioned than V_{i+1}.

Knyazev, A. V. (2001). Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM journal on scientific computing, 23(2), 517-541.

Definition of Basic_LOBPCG iterations, cont'd$_3$

- Dropping the $\widetilde{\bullet}$ notation, a pseudocode for the Basic_LOBPCG iterations is given as follows:

Basic_LOBPCG(A, B, X_{-1}, T^{-1}, k):

$\triangleright\ X_{-1} \in \mathbb{R}_*^{n\times m},\ k \le m \le n$

$(\hat{X}_0, \Lambda_0) \hookleftarrow \mathsf{RR}(A, B, X_{-1}, m)$

$X_0 := X_{-1}\hat{X}_0\ ;\ R_0 := AX_0 - BX_0\Lambda_0$

for $i = 0, 1, \ldots$ **do**

  $Z_i := T^{-1}R_i$

  if $i == 0$ **then** $V_{i+1} := [X_i, Z_i]$ **else** $V_{i+1} := [X_i, Z_i, P_i]$

  $(\hat{X}_{i+1}, \Lambda_{i+1}) \hookleftarrow \mathsf{RR}(A, B, V_{i+1}, m)$

  $X_{i+1} := V_{i+1}\hat{X}_{i+1};\ R_{i+1} := AX_{i+1} - BX_{i+1}\Lambda_{i+1}$

  if $i == 0$ **then** $P_{i+1} := Z_i\hat{X}_{i+1|Z_i}$ **else** $P_{i+1} := Z_i\hat{X}_{i+1|Z_i} + P_i\hat{X}_{i+1|P_i}$

- In some implementations, the iterate X_{i+1} is updated as

$$X_{i+1} := P_{i+1} + X_i\hat{X}_{i+1|X_i},$$

which is equivalent to setting $X_{i+1} := V_{i+1}\hat{X}_{i+1}$.

Rayleigh-Ritz procedure in Basic_LOBPCG

- Consider the reduced eigenvalue problems solved in the Rayleigh-Ritz procedure of Basic_LOBPCG.
  For $i > 1$, the following matrices need to be assembled:

$$V_{i+1}^T A V_{i+1} = \begin{bmatrix} X_i^T A X_i & X_i^T A Z_i & X_i^T A P_i \\ . & Z_i^T A Z_i & Z_i^T A P_i \\ . & . & P_i^T A P_i \end{bmatrix}$$

and

$$V_{i+1}^T B V_{i+1} = \begin{bmatrix} X_i^T B X_i & X_i^T B Z_i & X_i^T B P_i \\ . & Z_i^T B Z_i & Z_i^T B P_i \\ . & . & P_i^T B P_i \end{bmatrix}$$

where, by construction/convention, we have:

$$X_i^T A X_i = \Lambda_i \text{ and } X_i^T B X_i = I_m.$$

Rayleigh-Ritz procedure in Basic_LOBPCG, cont'd

Unless the basis V_{i+1} is orthogonalized, the remaining blocks

$$X_i^T AZ_i \ , \ X_i^T AP_i \ , \ Z_i^T AZ_i \ , \ Z_i^T AP_i, \ P_i^T AP_i$$
$$\text{and } X_i^T BZ_i \ , \ X_i^T BP_i \ , \ Z_i^T BZ_i \ , \ Z_i^T BP_i, \ P_i^T BP_i.$$

do not have any specific structures.

- We observed that, in practical implementations, which rely on implicit updates of the products AX, BX, AP and BP, the stability of LOBPCG is enhanced by explicitly computing $X^T AX$ rather than assuming that $X^T AX = \Lambda$ stands in finite precision.

Implicit product updates in Basic_LOBPCG

- From the fact that $P_1 = Z_0 \hat{X}_{1|Z_0}$ we can compute the products AP_1 and BP_1 from AZ_0 and BZ_0 as follows:

$$AP_1 := AZ_0 \hat{X}_{1|Z_0} \text{ and } BP_1 := BZ_0 \hat{X}_{1|Z_0}.$$

- From the fact that $X_1 = X_0 \hat{X}_{1|X_0} + Z_0 \hat{X}_{1|Z_0}$, the products AX_1 and BX_1 can be formed from AX_0, AP_1, BX_0 and BP_1 as follows:

$$AX_1 := AX_0 \hat{X}_{1|X_0} + AP_1 \text{ and } BX_1 := BX_0 \hat{X}_{1|X_0} + BP_1.$$

- For $i > 0$, from the fact that $P_{i+1} = Z_i \hat{X}_{i+1|Z_i} + P_i \hat{X}_{i+1|P_i}$, the AP_{i+1} and BP_{i+1} can be calculated as follows from AZ_i, AP_i, BZ_i and BP_i:

$$AP_{i+1} := AZ_i \hat{X}_{i+1|Z_i} + AP_i \hat{X}_{i+1|P_i} \text{ and } BP_{i+1} := BZ_i \hat{X}_{i+1|Z_i} + BP_i \hat{X}_{i+1|P_i}.$$

- For $i > 0$, from the fact that $X_{i+1} = P_{i+1} + X_i \hat{X}_{i+1|X_i}$, AX_{i+1} and BX_{i+1} can be calculated as follows from AP_{i+1}, AX_i, BP_{i+1} and BX_{i+1}:

$$AX_{i+1} := AP_{i+1} + AX_i \hat{X}_{i+1|X_i} \text{ and } BX_{i+1} := BP_{i+1} + BX_i \hat{X}_{i+1|X_i}.$$

Implementation of Basic_LOBPCG iterations

- Making use of the relations and implicit product updates presented, the implementation of Basic_LOBPCG iterations takes the following form:

Algorithm 1 `Basic_LOBPCG`$(A,\ B,\ X,\ T^{-1},\ k) \mapsto (X, \Lambda)$

Allocate memory for $Z, P \in \mathbb{R}^{n\times m}$ ▷ $X \in \mathbb{R}^{n\times m}_{*}$, $k \le m \le n$
Allocate memory for $AX, AZ, AP \in \mathbb{R}^{n\times m}$
Allocate memory for $BX, BZ, BP \in \mathbb{R}^{n\times m}$
Compute AX, BX
$(\hat{X}, \Lambda) \leftarrow \mathtt{RR}(X, AX, BX, m)$
$X := X\hat{X}$
$[AX, BX] := [AX\hat{X}, BX\hat{X}]$ ▷ implicit product updates
$Z := AX - BX\Lambda$
for $j = 0, 1, \ldots$ **do**
    $Z := T^{-1}Z$
    Compute AZ, BZ
    if $i == 0$ **then**
        $(\hat{X}, \Lambda) \leftarrow \mathtt{RR}(X, Z, AZ, BZ, \Lambda, m)$ ▷ $\hat{X} = [\hat{X}_X^T, \hat{X}_Z^T]^T$
        $P := Z\hat{X}_Z$
        $[AP, BP] := [AZ\hat{X}_Z, BZ\hat{X}_Z]$ ▷ implicit product updates
    else
        $(\hat{X}, \Lambda) \leftarrow \mathtt{RR}(X, Z, P, AZ, AP, BZ, BP, \Lambda, m)$ ▷ $\hat{X} = [\hat{X}_X^T, \hat{X}_Z^T, \hat{X}_P^T]^T$
        $P := Z\hat{X}_Z + P\hat{X}_P$
        $[AP, BP] := [AZ\hat{X}_Z, BZ\hat{X}_Z] + [AP\hat{X}_P, BP\hat{X}_P]$ ▷ implicit product updates
    $X := P + X\hat{X}_X$
    $[AX, BX] := [AP, BP] + [AX\hat{X}_X, BX\hat{X}_X]$ ▷ implicit product updates
    $Z := AX - BX\Lambda$

Implementation of Basic_LOBPCG iterations, cont'd$_1$

- The Rayleigh-Ritz procedures of `Basic_LOBPCG` are as follows:

Algorithm 2 RR$(X,\ AX,\ BX)$

1: ▷ $X \in \mathbb{R}_*^{n\times m},\ m \le n$
2: Compute $X^TAX, X^TBX \in \mathbb{R}^{m\times m}$
3: Solve for $\hat{X} \in \mathbb{R}_*^{m\times m}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ s.t. $G_A\hat{X} = G_B\hat{X}\Lambda$ and $\hat{X}^TG_B\hat{X} = I_m$ where $G_A := X^TAX$ and $G_B := X^TBX$.
4: **return** $\hat{X}, \Lambda$

Algorithm 3 RR$(X,\ Z,\ AX,\ AZ,\ BZ)$

1: ▷ $[X, Z] \in \mathbb{R}_*^{n\times 2m},\ 2m \le n$
2: Compute $X^TAX, X^TAZ, Z^TAZ, X^TBZ, Z^TBZ \in \mathbb{R}^{m\times m}$
3: Solve for $\hat{X} \in \mathbb{R}_*^{2m\times m}$ and $\Theta = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ s.t. $G_A\hat{X} = G_B\hat{X}\Lambda$ and $\hat{X}^TG_B\hat{X} = I_m$ where $G_A := \begin{bmatrix} X^TAX & X^TAZ \\ . & Z^TAZ \end{bmatrix}$, $G_B := \begin{bmatrix} I_m & X^TBZ \\ . & Z^TBZ \end{bmatrix}$ and $\lambda_1, \ldots, \lambda_m$ are extremal generalized eigenvalues of (G_A, G_B).
4: **return** $\hat{X}, \Lambda$

Implementation of Basic_LOBPCG iterations, cont'd$_2$

Algorithm 4 RR(X, Z, P, AX, AZ, AP, BZ, BP)

1: ▷ $[X, Z, P] \in \mathbb{R}^{n\times 3m}_*$, $3m \leq n$

2: Compute $X^TAX, X^TAZ, X^TAP, Z^TAZ, Z^TAP, P^TAP \in \mathbb{R}^{m\times m}$

3: Compute $X^TBZ, X^TBP, Z^TBZ, Z^TBP, P^TBP \in \mathbb{R}^{m\times m}$

4: Solve for $\hat{X} \in \mathbb{R}^{3m\times m}_*$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ s.t. $G_A\hat{X} = G_B\hat{X}\Lambda$ and $\hat{X}^TG_B\hat{X} = I_m$

where $G_A := \begin{bmatrix} X^TAX & X^TAZ & X^TAP \\ . & Z^TAZ & Z^TAP \\ . & . & P^TAP \end{bmatrix}$, $G_B := \begin{bmatrix} I_m & X^TBZ & X^TBP \\ . & Z^TBZ & Z^TBP \\ . & . & P^TBP \end{bmatrix}$ and

$\lambda_1, \ldots, \lambda_m$ are extremal generalized eigenvalues of (G_A, G_B).

5: **return** $\hat{X}, \Lambda$

Sources of instability in Basic_LOBPCG iterations

The instability of Basic_LOBPCG was showcased and related to the ill-conditioning of $V_{i+1}^T B V_{i+1}$. This was explained as follows in the works of Hetmaniuk and Lehoucq (2006), Knyazev et al. (2007) and Duersch (2015):

1. RR(A, B, V_{i+1}, m) needs to solve for $(\hat{X}, \Lambda)$ in the reduced equation

$$V_{i+1}^T A V_{i+1} \hat{X} = V_{i+1}^T B V_{i+1} \hat{X} \Lambda.$$

   This is done by computing the Cholesky decomposition $LL^T = V_{i+1}^T B V_{i+1}$ before solving for $(\hat{Y}, \Lambda)$ in the standard symmetric eigenvalue problem

$$L^{-1} V_{i+1}^T A V_{i+1} L^{-T} \hat{Y} = \hat{Y} \Lambda$$

   and letting $\hat{X} := L^{-T} \hat{Y}$.

   When $V_{i+1}^T B V_{i+1}$ is ill-conditioned, the Cholesky factorization may fail or lead to significant round-off error upon factor deployment.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.
Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.
Duersch, J. A. (2015). High Efficiency Spectral Analysis and BLAS-3 Randomized QRCP with Low-Rank Approximations. University of California, Berkeley.

Sources of instability in Basic_LOBPCG iterations, cont'd$_1$

2. As we denote the generalized eigenvalues of (A, B) and $(V^T AV, V^T BV)$ for some $V \in \mathbb{R}_*^{n\times m}$ with $m \leq n$ by $\lambda_1 \leq \cdots \leq \lambda_n$ and $\hat{\lambda}_1 \leq \cdots \leq \hat{\lambda}_m$, respectively, assuming A is positive definite, Parlett (1998, see Theorem 11.10.1) states that

$$|\lambda_i - \hat{\lambda}_i| \leq \frac{\|AV - BVL^{-1}V^T AVL^{-T}\|_{B^{-1}}}{\sqrt{\hat{\lambda}_1}}$$

where LL^T is the Cholesky decomposition of $V^T BV$.

Consequently, the error $|\lambda_i - \hat{\lambda}_i|$ of a Ritz value $\hat{\lambda}_i$ increases as the basis in the columns of V departs from B-orthonormality.

Parlett, B. N. (1998). The symmetric eigenvalue problem. Society for Industrial and Applied Mathematics.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Duersch, J. A. (2015). High Efficiency Spectral Analysis and BLAS-3 Randomized QRCP with Low-Rank Approximations. University of California, Berkeley.

Sources of instability in Basic_LOBPCG iterations, cont'd$_2$

- Additionally, the following observations were made:
  - The Gram matrix $V_{i+1}^T B V_{i+1}$ can become ill-conditioned irrespective of the conditioning of B.
  - When the number k of approximated eigenvectors is large, $V_{i+1}^T B V_{i+1}$ can become ill-conditioned before any eigenvector is accurately approximated.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Duersch, J. A. (2015). High Efficiency Spectral Analysis and BLAS-3 Randomized QRCP with Low-Rank Approximations. University of California, Berkeley.

More stable LOBPCG iterations

- Alternative LOBPCG iterations were introduced to fix the stability issues of Basic_LOBPCG:
  - BLOPEX was developed as an alternative implementation since 2005, and became the standard with adoption through Hypre and PETSc (i.e., SLEPc), see Knyazev et al. (2007).
    - In BLOPEX, Knyazev et al. (2007) B-orthogonalize the Z_i and P_i blocks of V_{i+1} independently, but not between themselves. The method is shown to work for specific examples.
  - Hetmaniuk and Lehoucq (2006) investigate the choice of basis used in the Rayleigh-Ritz procedure and its effect on stability with more details.
    - Hetmaniuk and Lehoucq (2006) argue that the strategy adopted in BLOPEX to handle ill-conditioned $V_{i+1}^T B V_{i+1}$ matrices has no theoretical justication, and it does not solve the problem when $[X_i, Z_i]$ is ill-conditioned.
    - In order to improve the stability of Basic_LOBPCG, they propose to B-orthonormalize V_{i+1} at each iteration. We call this Ortho_LOBPCG.
  - Duersch et al. (2018) improve the performance of Ortho_LOBPCG, and showcase its better stability in comparison to BLOPEX.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Ortho_LOBPCG iterations (Hetmaniuk and Lehoucq, 2006)

Definition of Ortho_LOBPCG iterations

- LOBPCG is made more robust by making V_{i+1} B-orthonormal, i.e., by making sure that $V_1^T B V_1 = I_{2m}$ and $V_{i+1}^T B V_{i+1} = I_{3m}$ for $i = 1, 2, \ldots$
- To do so, Hetmaniuk and Lehoucq (2006), rely on a generic procedure

$$\begin{aligned} \text{Ortho} : \mathbb{R}_*^{n\times p} \times \mathbb{S}_{++}^n \times \mathbb{R}_*^{n\times q} &\to \mathbb{R}_*^{n\times p} \\ (Z, B, W) &\mapsto V \end{aligned}$$

  such that $V^T B V = I_p$, $V^T B W = 0_{p\times q}$ and $\mathcal{R}(Z) \subseteq \mathcal{R}(V)$.
- The following observations are made to define Ortho_LOBPCG iterations:
  - Let $Z_0 \hookleftarrow \text{Ortho}(T^{-1}R_0, B, X_0)$ and $V_1 := [X_0, Z_0]$, so that $V_1^T B V_1 = I_{2m}$ and $\hat{X}_1^T \hat{X}_1 = \hat{X}_{1|X_0}^T \hat{X}_{1|X_0} + \hat{X}_{1|Z_0}^T \hat{X}_{1|Z_0} = I_m$.
  - Now, if we were to let $P_1 := Z_0 \hat{X}_{1|Z_0}$ as in Basic_LOBPCG, we would have

$$P_1^T B X_1 = (Z_0 \hat{X}_{1|Z_0})^T B V_1 \hat{X}_1 = \hat{X}_{1|Z_0}^T Z_0^T B [X_0, Z_0] \hat{X}_1 = \hat{X}_{1|Z_0}^T \hat{X}_{1|Z_0} \neq I_m$$

    so that P_1 also needs to be formed by B-orthonormalization against X_1.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Definition of Ortho_LOBPCG iterations, cont'd$_1$

Note however that letting $P_1 \leftarrow \text{Ortho}(Z_0\hat{X}_{1|Z_0}, B, X_1)$ is equivalent to

$$\begin{aligned}\hat{Y}_1 &\leftarrow \text{Ortho}([0_{m\times m}, \hat{X}_{1|Z_0}]^T, V_1^T B V_1, \hat{X}_1)\\ P_1 &:= V_1\hat{Y}_1\end{aligned}$$

as we have

1. $\hat{Y}_1^T V_1^T B V_1 \hat{Y}_1 = I_m \implies (V_1\hat{Y}_1)^T B V_1 \hat{Y}_1 = P_1^T B P_1 = I_m$
2. $\hat{Y}_1^T V_1^T B V_1 \hat{X}_1 = 0_{m\times m} \implies (V_1\hat{Y}_1)^T B X_1 = P_1^T B X_1 = 0_{m\times m}$
3. $\mathcal{R}([0_{m\times m}, \hat{X}_{1|Z_0}]^T) \subseteq \mathcal{R}(\hat{Y}_1) \implies \mathcal{R}(V_1[0_{m\times m}, \hat{X}_{1|Z_0}]^T) \subseteq \mathcal{R}(V_1\hat{Y}_1)$
   $\implies \mathcal{R}(Z_0\hat{X}_{1|Z_0}) \subseteq \mathcal{R}(P_1)$

Moreover, remember that $V_1^T B V_1 = I_{2m}$.
Consequently, P_1 may be constructed as follows:

$$\begin{aligned}\hat{Y}_1 &\leftarrow \text{Ortho}([0_{m\times m}, \hat{X}_{1|Z_0}]^T, I_{2m}, \hat{X}_1)\\ P_1 &:= V_1\hat{Y}_1\end{aligned}$$

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Definition of Ortho_LOBPCG iterations, cont'd$_2$

- Then, as we let $Z_1 \leftarrow\!\shortmid \text{Ortho}(T^{-1}R_1, B, [X_1, P_1])$ and $V_2 := [X_1, Z_1, P_1]$, we have $V_2^T B V_2 = I_{3m}$ and

$$\hat{X}_2^T \hat{X}_2 = \hat{X}_{2|X_1}^T \hat{X}_{2|X_1} + \hat{X}_{2|Z_1}^T \hat{X}_{2|Z_1} + \hat{X}_{2|P_1}^T \hat{X}_{2|P_1} = I_m.$$

Now, if we were to let $P_2 := Z_1 \hat{X}_{2|Z_1} + P_1 \hat{X}_{2|P_1}$, we still would have

$$\begin{aligned} P_2^T B X_2 &= (Z_1 \hat{X}_{2|Z_1} + P_1 \hat{X}_{2|P_1})^T B V_2 \hat{X}_2 \\ &= \hat{X}_{2|Z_1}^T Z_1^T B V_2 \hat{X}_2 + \hat{X}_{2|P_1}^T P_1^T B V_2 \hat{X}_2 \\ &= \hat{X}_{2|Z_1}^T Z_1^T B Z_1 \hat{X}_{2|Z_1} + \hat{X}_{2|P_1}^T P_1^T B P_1 \hat{X}_{2|P_1} \\ &= \hat{X}_{2|Z_1}^T \hat{X}_{2|Z_1} + \hat{X}_{2|P_1}^T \hat{X}_{2|P_1} \\ &\neq I_m \end{aligned}$$

so that P_2 needs to be formed by B-orthonormalization against X_2.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Definition of Ortho_LOBPCG iterations, cont'd$_3$

Similarly as before, letting $P_2 \leftarrow\!\shortmid \text{Ortho}(Z_1\hat{X}_{2|Z_1} + P_1\hat{X}_{2|P_1}, B, X_2)$ is equivalent to

$$\hat{Y}_2 \leftarrow\!\shortmid \text{Ortho}([0_{m\times m}, \hat{X}_{2|Z_1}^T, \hat{X}_{2|P_1}^T]^T, V_2^T B V_2, \hat{X}_2)$$
$$P_2 := V_2\hat{Y}_2$$

as we have

1. $\hat{Y}_2^T V_2^T B V_2 \hat{Y}_2 = I_m \implies (V_2\hat{Y}_2)^T B V_2 \hat{Y}_2 = P_2^T B P_2 = I_m$
2. $\hat{Y}_2^T V_2^T B V_2 \hat{X}_2 = 0_{m\times m} \implies (V_2\hat{Y}_2)^T B X_2 = P_2^T B X_2 = 0_{m\times m}$
3. $\mathcal{R}([0_{m\times m}, \hat{X}_{2|Z_1}^T, \hat{X}_{2|P_1}^T]^T) \subseteq \mathcal{R}(\hat{Y}_2) \implies \mathcal{R}(V_2[0_{m\times m}, \hat{X}_{2|Z_1}^T, \hat{X}_{2|P_1}^T]^T) \subseteq \mathcal{R}(V_2\hat{Y}_2)$
   $\implies \mathcal{R}(Z_1\hat{X}_{2|Z_1} + P_1\hat{X}_{2|P_1}) \subseteq \mathcal{R}(P_2)$

Moreover, remember that $V_2^T B V_2 = I_{3m}$.
Consequently, P_2 is best built by

$$\hat{Y}_2 \leftarrow\!\shortmid \text{Ortho}([0_{m\times m}, \hat{X}_{2|Z_1}^T, \hat{X}_{2|P_1}^T]^T, I_{3m}, \hat{X}_2)$$
$$P_2 := V_2\hat{Y}_2$$

- Subsequent iterates are defined similarly.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Definition of Ortho_LOBPCG iterations, cont'd$_4$

- In summary, Ortho_LOBPCG iterations are defined as follows:

Ortho_LOBPCG(A, B, X_{-1}, T^{-1}, k):

$\triangleright\, X_{-1} \in \mathbb{R}_*^{n\times m}$, $k \le m \le n$

$(\hat{X}_0, \Lambda_0) \hookleftarrow \mathsf{RR}(A, B, X_{-1}, m)$

$X_0 := X_{-1}\hat{X}_0$; $R_0 := AX_0 - BX_0\Lambda_0$

for $i = 0, 1, \ldots$ **do**

  if $i == 0$ **then** $Z_i \hookleftarrow \mathsf{Ortho}(T^{-1}R_i, B, X_i)$ **else** $Z_i \hookleftarrow \mathsf{Ortho}(T^{-1}R_i, B, [X_i, P_i])$

  if $i == 0$ **then** $V_{i+1} := [X_i, Z_i]$ **else** $V_{i+1} := [X_i, Z_i, P_i]$

  $(\hat{X}_{i+1}, \Lambda_{i+1}) \hookleftarrow \mathsf{RR}(A, B, V_{i+1}, m)$

  $X_{i+1} := V_{i+1}\hat{X}_{i+1}$; $R_{i+1} := AX_{i+1} - BX_{i+1}\Lambda_{i+1}$

  if $i == 0$ **then**

    $\hat{Y}_{i+1} \hookleftarrow \mathsf{Ortho}([0_{m\times m}, \hat{X}_{i+1|Z_i}^T]^T, I_{2m}, \hat{X}_{i+1})$

  else

    $\hat{Y}_{i+1} \hookleftarrow \mathsf{Ortho}([0_{m\times m}, \hat{X}_{i+1|Z_i}^T, \hat{X}_{i+1|P_i}^T]^T, I_{3m}, \hat{X}_{i+1})$

  $P_{i+1} := V_{i+1}\hat{Y}_{i+1}$

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Implicit product updates in Ortho_LOBPCG

- From the fact that $X_1 = X_0\hat{X}_{1|X_0} + Z_0\hat{X}_{1|Z_0}$, the products AX_1 and BX_1 can still be formed from AX_0, AZ_0, BX_0 and BZ_0 as follows:

$$AX_1 := AX_0\hat{X}_{1|X_0} + AZ_0\hat{X}_{1|Z_0} \text{ and } BX_1 := BX_0\hat{X}_{1|X_0} + BZ_0\hat{X}_{1|Z_0}.$$

- Similarly, from the fact that $P_1 = X_0\hat{Y}_{1|X_0} + Z_0\hat{Y}_{1|Z_0}$, we have:

$$AP_1 := AX_0\hat{Y}_{1|X_0} + AZ_0\hat{Y}_{1|Z_0} \text{ and } BP_1 := BX_0\hat{Y}_{1|X_0} + BZ_0\hat{Y}_{1|Z_0}.$$

- For $i > 0$, from the fact that $X_{i+1} = X_i\hat{X}_{i+1|X_i} + Z_i\hat{X}_{i+1|Z_i} + P_i\hat{X}_{i+1|P_i}$, AX_{i+1} and BX_{i+1} can be calculated as follows from AX_i, AZ_i, AP_i, BX_i, BZ_i and BP_i:

$$\begin{aligned} AX_{i+1} &:= AX_i\hat{X}_{i+1|X_i} + AZ_i\hat{X}_{i+1|Z_i} + AP_i\hat{X}_{i+1|P_i} \\ \text{and } BX_{i+1} &:= BX_i\hat{X}_{i+1|X_i} + BZ_i\hat{X}_{i+1|Z_i} + BP_i\hat{X}_{i+1|P_i}. \end{aligned}$$

- Similarly, AP_{i+1} and BP_{i+1} can be calculated as follows:

$$\begin{aligned} AP_{i+1} &:= AX_i\hat{Y}_{i+1|X_i} + AZ_i\hat{Y}_{i+1|Z_i} + AP_i\hat{Y}_{i+1|P_i} \\ \text{and } BP_{i+1} &:= BX_i\hat{Y}_{i+1|X_i} + BZ_i\hat{Y}_{i+1|Z_i} + BP_i\hat{Y}_{i+1|P_i}. \end{aligned}$$

Implementation of Ortho_LOBPCG iterations

- Making use of the relations and implicit product updates presented, the implementation of Ortho_LOBPCG iterations takes the following form:

Algorithm 5 `Ortho_LOBPCG`$(A,\ B,\ X,\ T^{-1},\ k) \mapsto (X, \Lambda)$

Allocate memory for $Z, P, W \in \mathbb{R}^{n\times m}$ ▷ $X \in \mathbb{R}_*^{n\times m}$, $k \leq m \leq n$
Allocate memory for $AX, AZ, AP \in \mathbb{R}^{n\times m}$
Allocate memory for $BX, BZ, BP \in \mathbb{R}^{n\times m}$
Compute AX, BX
$(\hat{X}, \Lambda) \leftarrow$ `RR`(X, AX, BX)
$[X, AX, BX] := [X\hat{X}, AX\hat{X}, BX\hat{X}]$ ▷ implicit product updates
$Z := AX - BX\Lambda$
for $j = 0, 1, \ldots$ **do**
  if $j == 0$ **then** $Z \leftarrow$ Ortho$(T^{-1}Z, B, X)$ **else** $Z \leftarrow$ Ortho$(T^{-1}Z, B, [X, P])$
  Compute AZ, BZ
  if $i == 0$ **then**
    $(\hat{X}, \Lambda) \leftarrow$ `RR`(X, Z, AX, AZ) ▷ $\hat{X} = [\hat{X}_X^T, \hat{X}_Z^T]^T$
    $\hat{Y} \leftarrow$ Ortho$([0_{m\times m}, \hat{X}_Z^T]^T, I_{2m}, \hat{X})$
    $[AP, BP] := [AX\hat{Y}_X, BX\hat{Y}_X] + [AZ\hat{Y}_Z, BZ\hat{Y}_Z]$ ▷ implicit product updates
    $[AX, BX] := [AX\hat{X}_X, BX\hat{X}_X] + [AZ\hat{X}_Z, BZ\hat{X}_Z]$ ▷ implicit product updates
    $W := X\hat{X}_X + Z\hat{X}_Z$; $P := X\hat{Y}_X + Z\hat{Y}_Z$; $X := W$
  else
    $(\hat{X}, \Lambda) \leftarrow$ `RR`(X, Z, P, AX, AZ, AP) ▷ $\hat{X} = [\hat{X}_X^T, \hat{X}_Z^T, \hat{X}_P^T]^T$
    $\hat{Y} \leftarrow$ Ortho$([0_{m\times m}, \hat{X}_Z^T, \hat{X}_P^T]^T, I_{3m}, \hat{X})$
    $[W, AX] := [AX\hat{Y}_X, AX\hat{X}_X] + [AZ\hat{Y}_Z, AZ\hat{X}_Z] + [AP\hat{Y}_P, AP\hat{X}_P]$; $AP := W$ ▷ implicit product updates
    $[W, BX] := [BX\hat{Y}_X, BX\hat{X}_X] + [BZ\hat{Y}_Z, BZ\hat{X}_Z] + [BP\hat{Y}_P, BP\hat{X}_P]$; $BP := W$ ▷ implicit product updates
    $W := X\hat{X}_X + Z\hat{X}_Z + P\hat{X}_P$; $P := X\hat{Y}_X + Z\hat{Y}_Z + P\hat{Y}_P$; $X := W$
  $Z := AX - BX\Lambda$

Implementation of Ortho_LOBPCG iterations, cont'd

- The Rayleigh-Ritz procedure $\texttt{RR}(X,\, AX,\, BX,\, k)$ is the same as before. The other Rayleigh-Ritz procedures in `Basic_LOBPCG` are

Algorithm 6 $\texttt{RR}(X,\, Z,\, AX,\, AZ)$

1: ▷ $[X, Z] \in \mathbb{R}_*^{n\times 2m}$, $2m \le n$
2: Compute $X^TAX, X^TAZ, Z^TAZ \in \mathbb{R}^{m\times m}$
3: Solve for $\hat{X} \in \mathbb{R}_*^{2m\times m}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ s.t. $G_A\hat{X} = \hat{X}\Theta$ and $\hat{X}^T\hat{X} = I_m$ where $G_A := \begin{bmatrix} X^TAX & X^TAZ \\ . & Z^TAZ \end{bmatrix}$ and $\lambda_1, \ldots, \lambda_m$ are extremal eigenvalues of G_A.
4: **return** $\hat{X}, \Lambda$

Algorithm 7 $\texttt{RR}(X,\, Z,\, P,\, AX,\, AZ,\, AP)$

1: ▷ $[X, Z, P] \in \mathbb{R}_*^{n\times 3m}$, $3m \le n$
2: Compute $X^TAX, X^TAZ, X^TAP, Z^TAZ, Z^TAP, P^TAP \in \mathbb{R}^{m\times m}$
3: Solve for $\hat{X} \in \mathbb{R}_*^{3m\times m}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ s.t. $G_A\hat{X} = \hat{X}\Lambda$ and $\hat{X}^T\hat{X} = I_m$ where $G_A := \begin{bmatrix} X^TAX & X^TAZ & X^TAP \\ . & Z^TAZ & Z^TAP \\ . & . & P^TAP \end{bmatrix}$ and $\lambda_1, \ldots, \lambda_m$ are extremal eigenvalues of G_A.
4: **return** $\hat{X}, \Lambda$

Choice of B-ortonormalization procedure

- The following orthogonalization procedures need be implemented in order to deploy Ortho_LOBPCG iterations:

$$
\begin{aligned}
Z &\leftarrow \text{Ortho}(Z, B, X) \\
Z &\leftarrow \text{Ortho}(Z, B, [X, P]) \\
\hat{X} &\leftarrow \text{Ortho}([0_{m\times m}, \hat{X}_Z^T]^T, I_{2m}, \hat{X}) \\
\hat{X} &\leftarrow \text{Ortho}([0_{m\times m}, \hat{X}_Z^T, \hat{X}_P^T]^T, I_{3m}, \hat{X})
\end{aligned}
$$

Different methods can be used for this purpose:

- In Hetmaniuk & Lehoucq (2006) and Duersch et al. (2018): SVD-based B-orthogonalization using SVQB from Stathopoulos & Wu (2002), which is cache-efficient, highly stable, with low synchronization cost.
- Householder-QR, which is highly stable, but difficult to implement for $B \neq I_n$.
- Gram-Schmidt procedures, which can be stable, but less efficient than SVQB.
- Cholesky-QR, which is not very stable.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Stathopoulos, A., & Wu, K. (2002). A block orthogonalization procedure with constant synchronization requirements. SIAM Journal on Scientific Computing, 23(6), 2165-2182.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Definition of the Ortho procedure

- In order to B-orthogonalize $U \in \mathbb{R}_*^{n\times p}$ against $V \in \mathbb{R}_*^{n\times q}$, the Ortho procedure is defined as follows using the SVQB method from Stathopoulos & Wu (2002):

Ortho(U, B, V):

$\triangleright U \in \mathbb{R}_*^{n\times p}, V \in \mathbb{R}_*^{n\times q}$, $p, q \leq n$
do
  $U := U - V(V^T BU)$
  do
    $U := \text{SVQB}(B, U)$
  while $\frac{\|U^T BU - I_p\|}{\|BU\|\|U\|} < \tau_{ortho}$
while $\frac{\|V^T BU\|}{\|BV\|\|U\|} < \tau_{ortho}$
return U

SVQB(U, B):

$\triangleright U \in \mathbb{R}_*^{n\times p}$, $p \leq n$
$D := (\text{diag}(U^T BU))^{-1/2}$
Solve for eigen-pairs Z, Θ of $DU^T BUD$ such that $DU^T BUDZ = Z\Theta$
$\theta_{max} := \max_i |\Theta_{ii}|$
for $i = 1, \ldots, p$ **do**
  if $\Theta_{ii} < \tau\, \theta_{max}$ **then** $\Theta_{ii} := \tau\, \theta_{max}$
return $UDZ\Theta^{-1/2}$

where τ_{ortho} and τ are set to modest multiples of the machine precision.

Stathopoulos, A., & Wu, K. (2002). A block orthogonalization procedure with constant synchronization requirements. SIAM Journal on Scientific Computing, 23(6), 2165-2182.

BLOPEX_LOBPCG iterations (Knyazev et al., 2007)

Definition of BLOPEX_LOBPCG iterations

- BLOPEX_LOBPCG iterations are summarized as follows:

BLOPEX_LOBPCG(A, B, X_{-1}, T^{-1}, k):

$\triangleright\, X_{-1} \in \mathbb{R}_*^{n\times m}$, $k \le m \le n$

B-orthogonalize X_{-1}: $L \hookleftarrow \mathrm{chol}(X_{-1}^T B X_{-1})$; $X_{-1} := X_{-1} L^{-T}$

$(\hat{X}_0, \Lambda_0) \hookleftarrow \mathrm{RR}(A, B, X_{-1}, m)$; $X_0 := X_{-1}\hat{X}_0$

for $i = 0, 1, \ldots$ **do**

  $R_i := AX_i - BX_i\Lambda_i$; $Z_i := T^{-1}R_i$

  B-orthogonalize Z_i: $L \hookleftarrow \mathrm{chol}(Z_i^T B Z_i)$; $Z_i := Z_i L^{-T}$

  if $i == 0$ **then** $V_{i+1} := [X_i, Z_i]$

  else

    B-orthogonalize P_i: $L \hookleftarrow \mathrm{chol}(P_i^T B P_i)$; $P_i := P_i L^{-T}$

    $V_{i+1} := [X_i, Z_i, P_i]$

  $(\hat{X}_{i+1}, \Lambda_{i+1}) \hookleftarrow \mathrm{RR}(A, B, V_{i+1}, m)$

  if $i == 0$ **then** $P_{i+1} := Z_i \hat{X}_{i+1|Z_i}$ **else** $P_{i+1} := Z_i \hat{X}_{i+1|Z_i} + P_i \hat{X}_{i+1|P_i}$

  $X_{i+1} := X_i \hat{X}_{i+1|X_i} + P_{i+1}$

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Implementation of BLOPEX_LOBPCG iterations

- Making use of implicit product updates and other relations, BLOPEX_LOBPCG iterations can be implemented as follows:

Algorithm 8 BLOPEX_LOBPCG$(A,\ B,\ X,\ T^{-1},\ k) \mapsto (X, \Lambda)$

1: Allocate memory for $Z, P \in \mathbb{R}^{n\times m}$ ▷ $X \in \mathbb{R}_*^{n\times m}$, $k \le m \le n$
2: Allocate memory for $AX, AZ, AP \in \mathbb{R}^{n\times m}$
3: Allocate memory for $BX, BZ, BP \in \mathbb{R}^{n\times m}$
4: Compute BX
5: B-orthogonalize X: $L \leftarrow \text{chol}(X^T BX)$; $X := XL^{-T}$; $BX := BXL^{-T}$ ▷ $LL^T = X^T BX$
6: Compute AX
7: $(\hat{X}, \Lambda) \leftarrow \text{RR}(X, AX, k)$
8: $[X, AX, BX] := [X\hat{X}, AX\hat{X}, BX\hat{X}]$ ▷ implicit product updates
9: **for** $j = 0, 1, \ldots$ **do**
10: $Z := AX - BX\Lambda$; $Z := T^{-1}Z$
11: Compute BZ
12: B-orthogonalize Z: $L \leftarrow \text{chol}(Z^T BZ)$; $Z := ZL^{-T}$; $BZ := BZL^{-T}$ ▷ $LL^T = Z^T BZ$
13: Compute AZ
14: **if** $i == 0$ **then**
15: $(\hat{X}, \Lambda) \leftarrow \text{RR\_BLOPEX}(X, Z, AX, AZ, BZ)$ ▷ $\hat{X} = [\hat{X}_X^T, \hat{X}_Z^T]^T$
16: $[P, AP, BP] := [Z\hat{X}_Z, AZ\hat{X}_Z, BZ\hat{X}_Z]$ ▷ implicit product updates
17: **else**
18: B-orthogonalize P: $L \leftarrow \text{chol}(P^T BP)$; $P := PL^{-T}$; $BP := BPL^{-T}$ ▷ $LL^T = P^T BP$
19: $(\hat{X}, \Lambda) \leftarrow \text{RR\_BLOPEX}(X, Z, P, AX, AZ, AP, BZ, BP)$ ▷ $\hat{X} = [\hat{X}_X^T, \hat{X}_Z^T, \hat{X}_P^T]^T$
20: $[P, AP, BP] := [Z\hat{X}_Z, AZ\hat{X}_Z, BZ\hat{X}_Z] + [P\hat{X}_P, AP\hat{X}_P, BP\hat{X}_P]$ ▷ implicit product updates
21: $[X, AX, BX] := [X\hat{X}_X, AX\hat{X}_X, BX\hat{X}_X] + [P, AP, BP]$ ▷ implicit product updates

Implementation of BLOPEX_LOBPCG iterations, cont'd$_1$

- The Rayleigh-Ritz procedures of `BLOPEX_LOBPCG` are as follows:

Algorithm 9 `RR`$(X,\ AX)$

1: ▷ $X \in \mathbb{R}_*^{n\times m},\ m \le n$
2: Compute $X^TAX \in \mathbb{R}^{m\times m}$
3: Solve for $\hat{X} \in \mathbb{R}_*^{m\times m}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ s.t. $G_A\hat{X} = \hat{X}\Lambda$ and $\hat{X}^T\hat{X} = I_m$ where $G_A := X^TAX$.
4: **return** $\hat{X}, \Lambda$

Algorithm 10 `RR_BLOPEX`$(X,\ Z,\ AX,\ AZ,\ BZ)$

1: ▷ $[X, Z] \in \mathbb{R}_*^{n\times 2m},\ 2m \le n$
2: Compute $X^TAX, X^TAZ, Z^TAZ, X^TBZ \in \mathbb{R}^{m\times m}$
3: Solve for $\hat{X} \in \mathbb{R}_*^{2m\times m}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ s.t. $G_A\hat{X} = G_B\hat{X}\Lambda$ and $\hat{X}^TG_B\hat{X} = I_m$ where $G_A := \begin{bmatrix} X^TAX & X^TAZ \\ . & Z^TAZ \end{bmatrix}$, $G_B := \begin{bmatrix} I_m & X^TBZ \\ . & I_m \end{bmatrix}$ and $\lambda_1, \ldots, \lambda_m$ are extremal generalized eigenvalues of (G_A, G_B).
4: **return** $\hat{X}, \Lambda$

Implementation of BLOPEX_LOBPCG iterations, cont'd$_2$

Algorithm 11 RR_BLOPEX(X, Z, P, AX, AZ, AP, BZ, BP)

1: ▷ $[X, Z, P] \in \mathbb{R}_*^{n\times 3m}$, $3m \le n$

2: Compute $X^TAX, X^TAZ, X^TAP, Z^TAZ, Z^TAP, P^TAP \in \mathbb{R}^{m\times m}$

3: Compute $X^TBZ, X^TBP, Z^TBP \in \mathbb{R}^{m\times m}$

4: Solve for $\hat{X} \in \mathbb{R}_*^{3m\times m}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ s.t. $G_A\hat{X} = G_B\hat{X}\Lambda$ and $\hat{X}^TG_B\hat{X} = I_m$

where $G_A := \begin{bmatrix} X^TAX & X^TAZ & X^TAP \\ . & Z^TAZ & Z^TAP \\ . & . & P^TAP \end{bmatrix}$, $G_B := \begin{bmatrix} I_m & X^TBZ & X^TBP \\ . & I_m & Z^TBP \\ . & . & I_m \end{bmatrix}$ and

$\lambda_1, \ldots, \lambda_m$ are extremal generalized eigenvalues of (G_A, G_B).

5: **return** $\hat{X}, \Lambda$

Skip_ortho_LOBPCG iterations (Duersch et al., 2018)

Definition of Skip_ortho_LOBPCG iterations

- Duersch et al. (2018) point out that the B-orthonormalization of Z_i against $[X_i, P_i]$ represents a significant cost of Ortho_LOBPCG iterations.
- However, the B-orthonormalization of Z_i is not always necessary for a stable implementation of LOBPCG.
- Consequently, they propose to skip parts of the B-orthonormalization:
  1. Start by Ortho_LOBPCG iterations, without the B-orthonormalization of Z_i.
     - As seen with Ortho_LOBPCG iterations, the B-orthonormalization of P_{i+1} against X_{i+1} can be done implicitly, at relatively low-cost. Indeed, we saw that

$$\hat{Y}_{i+1} \leftarrow \text{Ortho}([0_{m\times m}, \hat{X}^T_{i+1|Z_i}, \hat{X}^T_{i+1|P_i}]^T, V^T_{i+1}BV_{i+1}, \hat{X}_{i+1})$$
$$P_{i+1} := V_{i+1}\hat{Y}_{i+1}$$

       is equivalent to $P_{i+1} \leftarrow \text{Ortho}(Z_i\hat{X}_{i+1|Z_i} + P_i\hat{X}_{i+1|P_i}, B, X_{i+1})$ where, if Z_i is B-orthonormal w.r.t. $[X_i, P_i]$, we have $V^T_{i+1}BV_{i+1} = I_{3m}$.
     - For this reason, the B-orthonormalization of P_{i+1} against X_{i+1} is never skipped by Duersch et al. (2018).
       Then, as long as the B-orthonormalization of Z_i w.r.t. $[X_i, P_i]$ is skipped, the Gram matrix $V^T_{i+1}BV_{i+1}$ is not identity, and must be accounted for a proper implicit B-orthonormalization of P_{i+1} against X_{i+1}.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Definition of Skip_ortho_LOBPCG iterations, cont'd$_1$

2. For every Rayleigh-Ritz procedure, as long as V_{i+1} is not B-orthonormal, solving the reduced eigenvalue problem requires to factorize the Gram matrix $V_{i+1}^T B V_{i+1}$.
   - Duersch et al. (2018) investigate the conditioning of those factors, and then decide when to trigger the B-orthonormalization of Z_i w.r.t. $[X_i, P_i]$.
     To do so, the reduced eigenvalue problem is scaled as follows:

     1. $D := \text{diag}(V_{i+1}^T B V_{i+1})^{-1/2}$
     2. Compute Cholesky decomposition $LL^T = DV_{i+1}^T B V_{i+1} D$
     3. Solve for small eigenpairs in $(\Lambda, \hat{X})$ s.t. $L^{-1} D V_{i+1}^T B V_{i+1} D L^{-T} \hat{X} = \hat{X}\Lambda$
     4. Form Rayleigh-Ritz vectors as $X := V_{i+1} D L^{-T} \hat{X}$

3. Duersch et al. (2018) set a criterion on the conditioning of L to decide whether to keep the basis V_{i+1} as is, or not.
   - Since 3 triangular solves need be applied to form the Rayleigh-Ritz vectors, Duersch et al. (2018) check if $\text{cond}(L)^{-3}$ is greater than a modest multiple of machine precision. If so, the procedure is maintained as is.
     Otherwise, the Rayleigh-Ritz procedure is aborted and, from that point on, Z_i is always B-orthonormalized w.r.t. $[X_i, P_i]$.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Definition of Skip_ortho_LOBPCG iterations, cont'd$_2$

- We refer to these iterations as Skip_ortho_LOBPCG defined as follows:

Skip_ortho_LOBPCG(A, B, X_{-1}, T^{-1}, k, τ_{skip}):

skipOrtho := True ▷ $X_{-1} \in \mathbb{R}_*^{n\times m}$, $k \le m \le n$

$(\hat{X}_0, \Lambda_0) \leftarrow\!\shortmid \mathrm{RR}(A, B, X_{-1}, m)$

$X_0 := X_{-1}\hat{X}_0$; $R_0 := AX_0 - BX_0\Lambda_0$

for $i = 0, 1, \ldots$ **do**

  $Z_i := T^{-1}R_i$

  if !skipOrtho

    if $i == 0$ **then** $Z_i \leftarrow\!\shortmid \mathrm{Ortho}(Z_i, B, X_i)$ **else** $Z_i \leftarrow\!\shortmid \mathrm{Ortho}(Z_i, B, [X_i, P_i])$

  if $i == 0$ **then** $V_{i+1} := [X_i, Z_i]$ **else** $V_{i+1} := [X_i, Z_i, P_i]$

  $(\hat{X}_{i+1}, \Lambda_{i+1}) \leftarrow\!\shortmid \mathrm{RR}(A, B, V_{i+1}, m)$ ▷ L is a by-product s.t. $LL^T = DV_{i+1}^T BV_{i+1}D$

  if !skipOrtho **then if** $\mathrm{cond}(L)^{-3} < \tau_{skip}$ **then** skipOrtho := False ; restart i-th iteration

  $X_{i+1} := V_{i+1}\hat{X}_{i+1}$; $R_{i+1} := AX_{i+1} - BX_{i+1}\Lambda_{i+1}$

  if $i == 0$ **then**

    $\hat{Y}_{i+1} \leftarrow\!\shortmid \mathrm{Ortho}([0_{m\times m}, \hat{X}_{i+1|Z_i}^T]^T, V_{i+1}^T BV_{i+1}, \hat{X}_{i+1})$

  else

    $\hat{Y}_{i+1} \leftarrow\!\shortmid \mathrm{Ortho}([0_{m\times m}, \hat{X}_{i+1|Z_i}^T, \hat{X}_{i+1|P_i}^T]^T, V_{i+1}^T BV_{i+1}, \hat{X}_{i+1})$

  $P_{i+1} := V_{i+1}\hat{Y}_{i+1}$

Implementation of Skip_ortho_LOBPCG iterations

Algorithm 12 `Skip_ortho_LOBPCG`$(A,\ B,\ X,\ T^{-1},\ k,\ \tau_{skip}) \mapsto (X, \Lambda)$

1: Allocate memory for $Z, P \in \mathbb{R}^{n\times m}$ ▷ $X \in \mathbb{R}_*^{n\times m}$, $k \le m \le n$
2: Allocate memory for $AX, AZ, AP, BX, BZ, BP \in \mathbb{R}^{n\times m}$
3: skipOrtho := True
4: Compute AX, BX
5: $(\hat{X}, \Lambda) \leftarrow$ `RR`(X, AX, BX)
6: $[X, AX, BX] := [X\hat{X}, AX\hat{X}, BX\hat{X}]$ ▷ implicit product updates
7: $Z := AX - BX\Lambda$
8: **for** $j = 0, 1, \ldots$ **do**
9: $\quad Z := T^{-1}Z$
10: $\quad$ **if** not skipOrtho **then**
11: $\quad\quad$ **if** $j == 0$ **then** $Z \leftarrow$ Ortho(Z, B, X) **else** $Z \leftarrow$ Ortho$(Z, B, [X, P])$
12: $\quad$ Compute AZ, BZ
13: $\quad$ **if** skipOrtho **then**
14: $\quad\quad$ **if** $i == 0$ **then** $(\hat{X}, \Lambda) \leftarrow$ `RR`(X, Z, AX, AZ, BZ) **else** $(\hat{X}, \Lambda) \leftarrow$ `RR`$(X, Z, P, AX, AZ, AP, BZ, BP)$
15: $\quad$ **else**
16: $\quad\quad$ **if** $i == 0$ **then** $(\hat{X}, \Lambda) \leftarrow$ `RR`(X, Z, AX, AZ) **else** $(\hat{X}, \Lambda) \leftarrow$ `RR`(X, Z, P, AX, AZ, AP)
17: $\quad$ **if** skipOrtho **then if** cond$(L)^{-3} < \tau_{skip}$ **then** skipOrtho := False ; go to 13
18: $\quad$ **if** $i == 0$ **then**
19: $\quad\quad [AX, BX] := [AX\hat{X}_X, BX\hat{X}_X] + [AZ\hat{X}_Z, BZ\hat{X}_Z]$ ▷ implicit product updates
20: $\quad\quad \hat{Y} \leftarrow$ Ortho$([0_{m\times m}, \hat{X}_Z^T]^T, V_{i+1}^T B V_{i+1}, \hat{X})$
21: $\quad\quad [AP, BP] := [AX\hat{Y}_X, BX\hat{Y}_X] + [AZ\hat{Y}_Z, BZ\hat{Y}_Z]$ ▷ implicit product updates
22: $\quad\quad AZ := [X, Z]\hat{X}$; $P := [X, Z]\hat{Y}$; $X := AZ$
23: $\quad$ **else**
24: $\quad\quad [AX, BX] := [AX\hat{X}_X, BX\hat{X}_X] + [AZ\hat{X}_Z, BZ\hat{X}_Z] + [AP\hat{X}_P, BP\hat{X}_P]$ ▷ implicit product updates
25: $\quad\quad \hat{Y} \leftarrow$ Ortho$([0_{m\times m}, \hat{X}_Z^T, \hat{X}_P^T]^T, V_{i+1}^T B V_{i+1}, \hat{X})$
26: $\quad\quad [AP, BP] := [AX\hat{Y}_X, BX\hat{Y}_X] + [AZ\hat{Y}_Z, BZ\hat{Y}_Z] + [AP\hat{Y}_P, BP\hat{Y}_P]$ ▷ implicit product updates
27: $\quad\quad AZ := [X, Z, P]\hat{X}$; $P := [X, Z, P]\hat{Y}$; $X := AZ$
28: $\quad Z := AX - BX\Lambda$

Improvement of stability

- Even though Ortho_LOBPCG is more stable than BLOPEX_LOBPCG, Duersch et al. (2018) further improve the stability of Ortho_LOBPCG by:
  - Basis truncation:

    The B-orthogonalization of Z_i against $[X_i, P_i]$ using the SVQB procedure can sometimes yield a poorly conditioned V_{i+1}, even when $B := I_n$.

    To circumvent this issue, the basis generated by the SVQB procedure is truncated by discarding contributions below a round-off error threshold to represent the Gram matrix $Z^T BZ$, as previously done in Sathopoulos & Wu (2002) and Yamamoto et al. (2015).

    However, in order to deploy basis truncation, one needs to accommodate for Ortho$(Z, B, [X, P])$ to return a variable number of B-othonormalized preconditioned residuals, unless artificially maintaining the rank of $Z^T BZ$'s decomposition to a prescribed level.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Stathopoulos, A., & Wu, K. (2002). A block orthogonalization procedure with constant synchronization requirements. SIAM Journal on Scientific Computing, 23(6), 2165-2182.

Yamamoto, Y., Nakatsukasa, Y., Yanagisawa, Y., & Fukaya, T. (2015). Roundoff error analysis of the CholeskyQR2 algorithm. Electron. Trans. Numer. Anal, 44(01), 306-326.

Improvement of stability, cont'd$_1$

- Even though Ortho_LOBPCG is more stable than BLOPEX_LOBPCG, Duersch et al. (2018) further improve the stability of Ortho_LOBPCG by:
  - Improved basis selection strategy:

    Changes can also made to the way $[X_i, P_i]$ is implicitly B-orthogonalized, in order to improve both stability and performance.

    Let us consider the case when V is B-orthonormal, so that the Raleigh-Ritz procedure relies on solving the eigendecomposition $\hat{Z}\Theta\hat{Z}^T = V^T AV$ with the block structure given by

$$\hat{Z} = [\hat{X}, \hat{X}_\perp], \ \ \Theta = \text{diag}(\Lambda, \Lambda_\perp) \text{ where } \hat{X} = \begin{bmatrix} \hat{X}_X \\ \hat{X}_Z \\ \hat{X}_P \end{bmatrix} \text{ and } \hat{X}_\perp = \begin{bmatrix} \hat{X}_{\perp|X} \\ \hat{X}_{\perp|Z} \\ \hat{X}_{\perp|P} \end{bmatrix}.$$

    Then, the B-orthonormalization of $[X, P]$ is done implicitly by letting $P := V\hat{Y}$, in which $\hat{Y}$ is such that $\hat{Y}^T\hat{Y} = I_m$, $\hat{Y}^T\hat{X} = 0_{m\times m}$ and

$$\mathcal{R}([0_{m\times m}, \hat{X}_Z^T, \hat{X}_P^T]^T) \subseteq \mathcal{R}(\hat{Y}).$$

    This can be achieved by letting $\hat{Y}$ be a normalized $(I_{3m} - \hat{X}\hat{X}^T) \begin{bmatrix} 0_{m\times m} \\ \hat{X}_Z \\ \hat{X}_P \end{bmatrix}$.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Improvement of stability, cont'd$_2$

- Even though Ortho_LOBPCG is more stable than BLOPEX_LOBPCG, Duersch et al. (2018) further improve the stability of Ortho_LOBPCG by:
  - Improved basis selection strategy, cont'd:

    But since we have $\hat{X}\hat{X}^T + \hat{X}_\perp \hat{X}\perp^T = I_3 m$, we can actually focus on

$$\begin{aligned}\hat{X}_\perp \hat{X}_\perp^T \begin{bmatrix} 0_{m\times m} \\ \hat{X}_Z \\ \hat{X}_P \end{bmatrix} &= \hat{X}_\perp [\hat{X}_{\perp|X}^T, \hat{X}_{\perp|Z}^T, \hat{X}_{\perp|P}^T] \begin{bmatrix} 0_{m\times m} \\ \hat{X}_Z \\ \hat{X}_P \end{bmatrix} \\ &= \hat{X}_\perp \left(\hat{X}_{\perp|Z}^T \hat{X}_Z + \hat{X}_{\perp|P}^T \hat{X}_P \right) \\ &= -\hat{X}_\perp \hat{X}_{\perp|X}^T \hat{X}_X\end{aligned}$$

    where $\hat{X}_\perp$ is orthonormal and $\hat{X}_{\perp|X}^T \in \mathbb{R}^{2m\times m}$, so that it suffices to let

$$\hat{Y} := \hat{X}_\perp Q_{\perp|X}^T$$

    where $Q_{\perp|X}^T$ is the Q-factor of a QR decomposition $Q_{\perp|X}^T L_{\perp|X}^T$ of $X_{\perp|X}^T$.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Improvement of convergence detection

- Most commonly, the convergence of an approximate eigenpair (λ_i, x_i) is determined by a stopping criterion of the form

$$\frac{\|r_i\|}{|\lambda_i|\|x_i\|_B} = \frac{\|Ax_i - \lambda_i Bx_i\|}{|\lambda_i|\|x_i\|_B} \leq \tau.$$

However, as explained in Duersch et al. (2018), difficulties can occur when diagnosing convergence of iterative eigensolvers with this criterion:

1. For generalized problems, i.e., when $B \neq I_n$, the lack of scaling invariance of the stopping criterion makes convergence detection somewhat arbitrary.
   In particular, the smaller the matrix norm $\|B\|_2$, the more premature the convergence detection may be.
2. Even for standard problems, i.e., when $B = I_n$, if the magnitude of an approximate eigenvalue λ_i is too small compared to the matrix norm $\|A\|_2$, the eigenresidual norm $\|r_i\|_2$ cannot be computed in floating point arithmetic to satisfy the stopping criterion, in which case convergence may not be detected.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Improvement of convergence detection, cont'd

- As a means to circumvent issues 1. and 2., Duersch et al. (2018) use the backward stable criterion

$$\frac{\|r_i\|_2}{(\|A\|_2 + |\lambda_i|\|B\|_2)\|x_i\|_2} = \frac{\|Ax_i - \lambda_i Bx_i\|_2}{(\|A\|_2 + |\lambda_i|\|B\|_2)\|x_i\|_2} \leq \tau.$$

Since evaluating the matrix 2-norms $\|A\|_2$ and $\|B\|_2$ can be costly, Duersch et al. (2018) suggest to rely on random sketching $x \mapsto \Omega x$ s.t.

$$\|A\|_2^{(\Omega)} := \frac{\|\Omega A\|_F}{\|\Omega\|_F} \leq \|A\|_2.$$

Then, convergence is monitored with the following criterion instead:

$$\frac{\|r_i\|_2}{(\|A\|_2 + |\lambda_i|\|B\|_2)\|x_i\|_2} \leq \frac{\|r_i\|_2}{(\|A\|_2^{(\Omega)} + |\lambda_i|\|B\|_2^{(\Omega)})\|x_i\|_2} \leq \tau.$$

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Monitoring and handling convergence

Handling convergence

- Irrespective of the criterion used, see slide on convergence detection, different eigenvectors may converge at different stages of the iteration. Maintaining converged eigenvectors to perform subsequent iterations
  1. requires unnecessary computational work,
  2. can lead to instabilities.

  $\Longrightarrow$ A robust and efficient implementation of LOBPCG needs to detect, and properly handle converged eigenvectors.
- Two approaches possible, see Knyazev (2004) and Knyazev et al. (2007):
  - **Hard locking**: converged eigenvectors are set aside, kept unchanged, and B-orthogonalized against by the non-converged, still iterated eigenvectors.
    - As the number of hard locked vectors increases, the attainable accuracy of the iterated eigenvectors may decrease, possibly making convergence unachievable.
  - **Soft locking**: the residuals and search directions of converged eigenvectors are set aside, and kept unchanged, but the corresponding locked eigenvectors still participate to subsequent Rayleigh-Ritz procedures.
    - The locked eigenpairs keep getting more accurate over subsequent iterations, and the B-orthogonality is maintained implicitly through the Rayleigh-Ritz procedures.

Knyazev, A. V. (2004). Hard and soft locking in iterative methods for symmetric eigenvalue problems. In Presentation at the eighth copper mountain conference on iterative methods.

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Handling convergence, cont'd

- Soft locking is more computationally demanding than hard locking, but it enables more robust convergence behaviors when more accurate solutions are needed.
- In practice, convergence may be detected in unordered fashions, i.e., the inner eigenpairs converge before the smallest eigenpairs.
- We denote two distinct approaches to deal with this situation:
  - **out-of-order locking**: if locking is implemented out of order, one needs to re-order the stored iterates so as to seamlessly rely on standard BLAS libraries, which operate most efficiently on contiguous data.
  - **in-order locking**: more commonly in practice, locking is implemented in order, disregarding the fact that some inner eigenpairs may converge before the sought least dominant eigenpairs.
    - Maintaining such unlocked but converged eigenvectors in the iterations can lead to unstable behaviors of LOBPCG.
- The theoretical underpinnings of locking, particularly out-of-order locking, are not well studied.

Knyazev, A. V. (2004). Hard and soft locking in iterative methods for symmetric eigenvalue problems. In Presentation at the eighth copper mountain conference on iterative methods.

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Landscape of existing software

Existing implementations of LOBPCG

Different implementations of LOBPCG have been developed over the years. In particular, we know of implementations and bindings in the following libraries:

- BLOPEX: C implementation with MPI support after Knyazev et al. (2007). On GitHub at `lobpcg/blopex`.
  - BLOPEX can be called from Matlab, SLEPc and Hypre.
- MAGMA: C++ implementation based on BLOPEX (?) for $B := I_n$ and $T^{-1} := I_n$ with GPU support. On GitHub at

  `CEED/MAGMA/sparse/src/zlobpcg.cpp`
- SciPy: Python implementation based on BLOPEX. On GitHub at

  `scipy/sparse/linalg/eigen/lobpcg/lobpcg.py`
- IterativeSolvers.jl: Julia implementation based on BLOPEX with multithreaded BLAS support. On GitHub at

  `JuliaLinearAlgebra/IterativeSolvers.jl/src/lobpcg.jl`

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Existing implementations of LOBPCG, cont'd

- BLOPEX from Knyazev et al. (2007) has become the most common reference among widely used implementations of LOBPCG.
- At the moment, there seems to be no widely used implementations of
  - `Ortho_LOBPCG` (Hetmaniuk and Lehoucq, 2006)
  - `Skip_ortho_LOBPCG` (Duersch et al., 2018)
  - Mixed precision LOBPCG (Kressner et al., 2023)
  - Randomized LOBPCG (Xiang, 2024)

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Kressner, D., Ma, Y., & Shao, M. (2023). A mixed precision LOBPCG algorithm. Numerical Algorithms, 94(4), 1653-1671.

Xiang, Y. (2024). Randomized LOBPCG algorithm with dimension reduction maps. Technical Report of Inria, hal-04517617.

Our implementations

Functions implemented

- The following functions are implemented in Julia and Python:
  - `Basic_LOBPCG` as in Algo. 1, see Knyazev (2001)
  - `Ortho_LOBPCG` as in Algo. 5, see Hetmaniuk & Lehoucq (2006)
  - `BLOPEX_LOBPCG` as in Algo. 8, see Knyazev et al. (2007)
  - `Skip_ortho_LOBPCG` as in Algo. 12, see Duersch et al. (2018)
    - **Basis truncation** of Duersch et al. (2018) is **not implemented**.
    - **Improved basis selection** of Duersch et al. (2018) is **implemented**.
    - **Improved convergence detection** of Duersch et al. (2018) is **not implemented**.
    - **Orthogonalization** is **skipped** if $cond(L)^{-3} > 2 \times \epsilon_{mach}$.
- **In-order soft locking** is **implemented for** `Basic_LOBPCG` only.
- **Hard-locking** is **not implemented** in any of the routines.
- **Implicit product updates** are **enabled by default**, but **can be deactivated** for products with either A, B or both.

Knyazev, A. V. (2001). Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM journal on scientific computing, 23(2), 517-541.

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Numerical experiments

Previous experiments on stability

- The following matrices and matrix pencils were used for the experiments and comments on BLOPEX's behavior by Duersch et al. (2018):

Matrix	Type of problem	Size	nnz	BLOPEX
C60*	standard	17 556	407 204	✘
Si5H12†	standard	19 896	738 598	✓
c-65†	standard	48 066	360 428	✗
Andrews†	standard	60 000	760 154	✓
Ga3As3H12†	standard	61 349	5 970 947	✓
Ga10As10H30†	standard	113 081	6 115 633	✗
nanotube°	generalized	9984	5 076 862	✘
graphene°	generalized	21 060	2 357 415	✘

*: Matrix generated with the Matlab RSDFT package for a Buckyball molecule.

†: SuiteSparse matrices.

°: Matrix pencils generated with the electronic structure software package SIESTA.

✗: fails. ✘: converges to incorrect eigen-pairs. ✓: converges to wanted eigen-pairs.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

New generalized stability experiments (Julia)

- The following tests are run for **generalized** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau := 10^{-5}}$.
  - Results obtained with **implicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`IterativeSolvers.jl`	`Ortho`	`Skip_ortho`
(`bcsstk12`, `bcsstm12`)*	✓	✗	✓°	✓	✓
(`bcsstk25`, `bcsstm25`)†	✗	✗	✗°	✘	✘

  - Results obtained with **explicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`IterativeSolvers.jl`	`Ortho`	`Skip_ortho`
(`bcsstk12`, `bcsstm12`)*	✓	✓	N.A.	✓	✓
(`bcsstk25`, `bcsstm25`)†	✗	✗	N.A.	✓	✓

✗: fails in the Rayleigh-Ritz procedure when trying to factorize the Gram matrix $V^T BV$.

✘: does not converge with wanted accuracy.

✓: converges with wanted accuracy.

°: If nev is specified, `IterativeSolvers.jl` discards $X_0[:, nev+1:m]$, and then sets $m := nev$. To work around this implementation error, we need not specify nev, and monitor convergence ourselves.

*: SuiteSparse matrices from Boeing with $n = 1473$, $nnz(A) = 34\,241$ and $nnz(B) = 19659$.

†: SuiteSparse matrices from Boeing with $n = 15\,439$, $nnz(A) = 252\,241$ and $nnz(B) = n$.

New generalized stability experiments (Julia), cont'd

- The following tests are run for **generalized** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau := 10^{-6}}$.
  - Results obtained with **implicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`IterativeSolvers.jl`	`Ortho`	`Skip_ortho`
(`bcsstk12`, `bcsstm12`)*	✗	✗	✗°	✓	✓
(`bcsstk25`, `bcsstm25`)†	✗	✗	✗°	✘	✘

  - Results obtained with **explicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`IterativeSolvers.jl`	`Ortho`	`Skip_ortho`
(`bcsstk12`, `bcsstm12`)*	✗	✗	N.A.	✓	✓
(`bcsstk25`, `bcsstm25`)†	✗	✗	N.A.	✓	✔

✗: fails in the Rayleigh-Ritz procedure when trying to factorize the Gram matrix $V^T BV$.

✘: does not converge with wanted accuracy. ✓: converges with wanted accuracy.

✔: converges with wanted accuracy, but requires 2.42X more iterations.

°: If nev is specified, `IterativeSolvers.jl` discards $X_0[:, nev+1:m]$, and then sets $m := nev$. To work around this implementation error, we need not specify nev, and monitor convergence ourselves.

*: SuiteSparse matrices from Boeing with $n = 1473$, $nnz(A) = 34\,241$ and $nnz(B) = 19659$.

†: SuiteSparse matrices from Boeing with $n = 15\,439$, $nnz(A) = 252\,241$ and $nnz(B) = n$.

New generalized stability experiments (Python)

- The following tests are run for **generalized** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau := 10^{-5}}$.
  - Results obtained with **implicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`SciPy`	`Ortho`	`Skip_ortho`
(`bcsstk12`, `bcsstm12`)*	✓	✓	✓°	✓	✓
(`bcsstk25`, `bcsstm25`)†	✗	✗	✘°	✗	✘

  - Results obtained with **explicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`SciPy`	`Ortho`	`Skip_ortho`
(`bcsstk12`, `bcsstm12`)*	✓	✓	N.A.	✓	✓
(`bcsstk25`, `bcsstm25`)†	✗	✗	N.A.	✗	✘

✗: fails in the Rayleigh-Ritz procedure when trying to factorize the Gram matrix $V^T BV$.

✗: fails in the (implicit) B-orthogonalization of P against X.

✘: does not converge with wanted accuracy. ✓: converges to wanted eigen-pairs.

°: `SciPy` does not allow for the specification of some $nev < m$.
To work around this implementation choice, we need to monitor convergence ourselves.

*: SuiteSparse matrices from Boeing with $n = 1473$, $nnz(A) = 34\,241$ and $nnz(B) = 19659$.

†: SuiteSparse matrices from Boeing with $n = 15\,439$, $nnz(A) = 252\,241$ and $nnz(B) = n$.

New generalized stability experiments (Python), cont'd

- The following tests are run for **generalized** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau := 10^{-6}}$.
  - Results obtained with **implicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`SciPy`	`Ortho`	`Skip_ortho`
(`bcsstk12`, `bcsstm12`)*	✗	✗	✘°	✓	✔
(`bcsstk25`, `bcsstm25`)†	✗	✗	✘°	✗	✘

  - Results obtained with **explicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`SciPy`	`Ortho`	`Skip_ortho`
(`bcsstk12`, `bcsstm12`)*	✗	✓	N.A.	✓	✓
(`bcsstk25`, `bcsstm25`)†	✗	✗	N.A.	✗	✘

✗: fails in the Rayleigh-Ritz procedure when trying to factorize the Gram matrix $V^T BV$.
✘: does not converge with wanted accuracy. ✓: converges with wanted accuracy.
✔: converges with wanted accuracy, but requires significantly more iterations.

°: `SciPy` does not allow for the specification of some $nev < m$.
To work around this implementation choice, we need to monitor convergence ourselves.
*: SuiteSparse matrices from Boeing with $n = 1473$, $nnz(A) = 34\,241$ and $nnz(B) = 19659$.
†: SuiteSparse matrices from Boeing with $n = 15\,439$, $nnz(A) = 252\,241$ and $nnz(B) = n$.

New generalized performance experiments (Julia)

- The following tests are run for **generalized** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau} := \mathbf{10^{-5}}$ and `num_threads` $:= 6$.
  - Runtimes to reach convergence with **implicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`IterativeSolvers.jl`	`Ortho`	`Skip_ortho`
`Poisson8`†	16 s 27 it	16 s 27 it	26 s ° 28 it	29 s 27 it	23 s 27 it
`Poisson16`†	✗	36 s 42 it	57 s ° 42 it	73 s 41 it	48 s 41 it
`Poisson32`†	✗	82 s 55 it	132 s ° 55 it	188 s 54 it	108 s 54 it
`Poisson64`†	✗	249 s 89 it	423 s ° 89 it	640 s 88 it	378 s 88 it
`Poisson128`†	628 s 102 it	761 s 102 it	1253 s ° 103 it	2038 s 102 it	1199 s 102 it

Experiences done on an Intel CPU 12th Gen Core i7-1255U.

†: `PoissonX` refers to a matrix pencil from the unstructured FE discretization of a 2D Poisson PDE with a random variable coefficient s.t. $n(A) \approx \texttt{X} \times 1000$.

°: If nev is specified, `IterativeSolvers.jl` discards $X_0[:, nev+1:m]$, and then sets $m := nev$. To work around this implementation error, we need not specify nev, and monitor convergence ourselves.

New generalized performance experiments (Python)

- The following tests are run for **generalized** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau} := \mathbf{10^{-5}}$ and `num_threads` $:= 6$.
  - Runtimes to reach convergence with **implicit product updates**:

Matrix pencil	`Basic`	`BLOPEX`	`SciPy`	`Ortho`	`Skip_ortho`
`Poisson8`†	15 s 27 it	18 s 27 it	14 s ° 27 it	24 s 27 it	24 s 27 it
`Poisson16`†	✗	54 s 42 it	48 s ° 42 it	83 s 41 it	69 s 41 it
`Poisson32`†	✗	161 s 55 it	157 s ° 55 it	268 s 54 it	197 s 54 it
`Poisson64`†	✗	✗	508 s ° 88 it	807 s 88 it	628 s 88 it
`Poisson128`†	971 s 102 it	1105 s 103 it	1212 s ° 102 it	1782 s 102 it	1465 s 102 it

Experiences done on an Intel CPU 12th Gen Core i7-1255U.

†: `PoissonX` refers to a matrix pencil from the unstructured FE discretization of a 2D Poisson PDE with a random variable coefficient s.t. $n(A) \approx \mathtt{X} \times 1000$.

°: `SciPy` does not allow for the specification of some $nev < m$.
To work around this implementation choice, we need to monitor convergence ourselves.

New standard stability experiments (Julia)

- The following tests are run for **standard** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau := 10^{-5}}$.
  - Results obtained with **implicit product updates**:

Matrix	`Basic`	`BLOPEX`	`IterativeSolvers.jl`	`Ortho`	`Skip_ortho`
`bcsstk15`[1]	✗	✗	✗°	✓	✓
`bcsstk21`[2]	✗	✗	✗°	✓	✓
`bodyy6`[3]	✗	✗	✗°	✓	✓
`tmt_sym`[4]	✗	✓	✘°	✓	✓

✗: fails in the Rayleigh-Ritz procedure when trying to factorize the cross-product matrix $V^T V$.

✓: converges to wanted eigen-pairs.

°: If nev is specified, `IterativeSolvers.jl` discards $X_0[:, nev+1:m]$, and then sets $m := nev$. To work around this implementation error, we need not specify nev, and monitor convergence ourselves.

[1]: SuiteSparse matrices from Boeing with $n = 3948$ and $nnz(A) = 117\,816$.

[2]: SuiteSparse matrices from Boeing with $n = 3600$ and $nnz(A) = 26\,600$.

[3]: SuiteSparse matrices from NASA with $n = 19\,366$ and $nnz(A) = 134\,208$.

[4]: SuiteSparse matrices from CEMW with $n = 726\,713$ and $nnz(A) = 5\,080\,961$.

New standard stability experiments (Python)

- The following tests are run for **standard** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau := 10^{-5}}$.
  - Results obtained with **implicit product updates**:

Matrix	`Basic`	`BLOPEX`	`SciPy`	`Ortho`	`Skip_ortho`
`bcsstk15`[1]	✗	✗	✘ °	✓	✓
`bcsstk21`[2]	✗	✗	✓ °	✓	✓
`bodyy6`[3]	✗	✗	✘ °	✓	✓
`tmt_sym`[4]	✗	✓	✘ °	✓	✓

✗: fails in the Rayleigh-Ritz procedure when trying to factorize the cross-product matrix $V^T V$.

✘: does not converge with wanted accuracy.

✓: converges with wanted accuracy.

°: `SciPy` does not allow for the specification of some $nev < m$.
To work around this implementation choice, we need to monitor convergence ourselves.

[1]: SuiteSparse matrices from Boeing with $n = 3948$ and $nnz(A) = 117\,816$.

[2]: SuiteSparse matrices from Boeing with $n = 3600$ and $nnz(A) = 26\,600$.

[3]: SuiteSparse matrices from NASA with $n = 19\,366$ and $nnz(A) = 134\,208$.

[4]: SuiteSparse matrices from CEMW with $n = 726\,713$ and $nnz(A) = 5\,080\,961$.

New standard performance experiments (Julia)

- The following tests are run for **standard** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau := 10^{-5}}$ and `num_threads` $:= 6$.
  - Runtimes to reach convergence with **implicit product updates**:

Matrix	`Basic`	`BLOPEX`	`IterativeSolvers.jl`	`Ortho`	`Skip_ortho`
`Poisson8_k`†	✗	31 s 36 it	✗$^\circ$	50 s 36 it	39 s 36 it
`Poisson16_k`†	✗	59 s 54 it	✗$^\circ$	104 s 54 it	67 s 54 it
`Poisson32_k`†	✗	121 s 73 it	✗$^\circ$	240 s 73 it	134 s 73 it
`Poisson64_k`†	✗	✗	✗$^\circ$	674 s 116 it	593 s 116 it
`Poisson128_k`†	✗	693 s 140 it	✗$^\circ$	1484 s 140 it	1165 s 140 it

Experiences done on an Intel CPU 12th Gen Core i7-1255U.

†: `PoissonX_k` refers to a stiffness matrix from the unstructured FE discretization of a 2D Poisson PDE with a random variable coefficient s.t. $n(A) \approx$ `X` $\times 1000$.

$^\circ$: If nev is specified, `IterativeSolvers.jl` discards $X_0[:, nev+1:m]$, and then sets $m := nev$. To work around this implementation error, we need not specify nev, and monitor convergence ourselves.

New standard performance experiments (Python)

- The following tests are run for **standard** eigenvalue problems.
  - $T :=$ block Jacobi with Cholesky factors of 10 diagonal blocks of A.
  - $nev := 10$, $m := 200$, $\boldsymbol{\tau} := \mathbf{10^{-5}}$ and `num_threads` $:= 6$.
  - Runtimes to reach convergence with **implicit product updates**:

Matrix	`Basic`	`BLOPEX`	`SciPy`	`Ortho`	`Skip_ortho`
`Poisson8_k`†	✗	20 s 37 it	✗°	25 s 36 it	26 s 36 it
`Poisson16_k`†	✗	✗	✗°	86 s 54 it	76 s 54 it
`Poisson32_k`†	✗	174 s 76 it	✗°	251 s 73 it	208 s 73 it
`Poisson64_k`†	✗	✗	✗°	825 s 116 it	740 s 116 it
`Poisson128_k`†	✗	✗	✗°	1949 s 140 it	1940 s 140 it

Experiences done on an Intel CPU 12th Gen Core i7-1255U.

†: `PoissonX_k` refers to a stiffness matrix from the unstructured FE discretization of a 2D Poisson PDE with a random variable coefficient s.t. $n(A) \approx \mathtt{X} \times 1000$.

°: `SciPy` does not allow for the specification of some $nev < m$.
To work around this implementation choice, we need to monitor convergence ourselves.

Conclusion

- Stability issues are prevalent when deploying LOBPCG implementations. This is especially the case for the Basic iterations of Knyazev (2001):
  - The core of the issue lies in the poor conditioning of $V := [X, Z, P]$, the difficulty to factorize the Gram matrix $V^T BV$ in the Rayleigh-Ritz procedure, and the round-off error accumulated by subsequent triangular solves.
  - Instabilities occur irrespective of the conditioning of B, i.e., even for $B = I_n$.
- Knyazev et al. (2007) introduce BLOPEX iterations, in which blocks of V are B-orthogonalized, independently from each other:
  - More stable than Basic iterations, but still unstable.
- Hetmaniuk & Lehoucq (2006) introduce Ortho iterations, where V is consistently B-orthogonalized:
  - Most stable iterations, but also computationally most expensive to deploy.
- Duersch et al. (2018) suggest improvements to LOBPCG implementations, with a criterion to skip the most expensive parts of the B-orthogonalization of V, leading to robust and efficient Skip_ortho iterations.

Knyazev, A. V. (2001). Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM journal on scientific computing, 23(2), 517-541.

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Conclusion, cont'd$_1$

- We implement the following types of preconditioned LOBPCG iterations, in Julia and Python, for both standard and generalized problems:
  - `Basic_LOBPCG` as in Knyazev (2001), with in-order soft locking.
  - `BLOPEX_LOBPCG` as in Knyazev et al. (2007), but without locking.
  - `Ortho_LOBPCG` as in Hetmaniuk & Lehoucq (2006), without locking.
  - `Skip_ortho_LOBPCG` partly as in Duersch et al. (2018), skipping costly steps of B-orthogonalization when possible, without locking.
- Our implementations allow for implicit or explicit matrix-products updates:
  - As pointed out by Duersch et al. (2018), some generalized problems require explicit product updates for a better stability, and faster convergence.
- In Julia, our numerical experiments show that:
  - `IterativeSolvers.jl`'s LOBPCG, which is based on BLOPEX, is 40% slower on average than our `BLOPEX_LOBPCG` implementation.
  - Our implementation of `Skip_ortho_LOBPCG` is nearly as stable as `Ortho_LOBPCG`, while being as fast as `IterativeSolvers.jl`'s LOBPCG.

Knyazev, A. V. (2001). Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM journal on scientific computing, 23(2), 517-541.

Knyazev, A. V., Argentati, M. E., Lashuk, I., & Ovtchinnikov, E. E. (2007). Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in Hypre and PETSc. SIAM Journal on Scientific Computing, 29(5), 2224-2239.

Hetmaniuk, U., & Lehoucq, R. (2006). Basis selection in LOBPCG. Journal of Computational Physics, 218(1), 324-332.

Duersch, J. A., Shao, M., Yang, C., & Gu, M. (2018). A robust and efficient implementation of LOBPCG. SIAM Journal on Scientific Computing, 40(5), C655-C676.

Conclusion, cont'd$_2$

- In Python, our numerical experiments show that:
  - `SciPy`'s implementation of LOBPCG iterations never seem to fail, per se. Instead, checks are made on the definiteness of Gram matrices, and Rayleigh-Ritz performed accordingly, leading to stagnation of iterates.
  - Our implementation of `BLOPEX_LOBPCG` is not consistently faster than `SciPy`'s LOBPCG iterations.
  - Our implementation of `Skip_ortho_LOBPCG` is nearly as stable as `Ortho_LOBPCG` and significantly more stable than `SciPy`'s LOBPCG.¨
- While LOBPCG iterations are faster for larger problems in Julia, they are faster in Python for smaller problems.
- To consider:
  - Enabling computation of approximate largest eigenpairs in methods implemented (need better preconditioners).
  - Implementing better basis selection for the implicit B-orthonormalization of $[X, P]$ when $[X, Z, P]$'s B-orgonolization is skipped in Skip_ortho.
  - Implementing mixed-precision.
  - Implementing randomization, and investigate performance, accuracy and stability.