Efficient subspace iteration with Chebyshev-type filtering

Bruno Lang

Bergische Universität Wuppertal, Mathematik / Angewandte Informatik

PMAA2016, July 8, 2016
The ESSEX project: Equipping Sparse Solvers for Exascale

This work is supported by Deutsche Forschungsgemeinschaft through the Priority Programme 1648 “Software for Exascale Computing”

G. Hager, M. Kreutzer, F. Shahzad, G. Wellein
A. Alvermann, H. Fehske, A. Pieper
A. Basermann, M. Röhrig-Zöllner, J. Thies
M. Galgon, S. Huber, B. Lang
K. Nakajima, A. Ida, M. Kawai
T. Sakurai, Y. Futamura, A. Imakura

Bruno Lang, Efficient subspace iteration with Chebyshev filtering
Subspace iteration with Rayleigh–Ritz extraction

Improving the coefficients of the polynomials

Adaptive control of the degree

A priori information about the spectrum

High performance computational kernels
Subspace iteration with Rayleigh–Ritz extraction

Given: \( A \in \mathbb{C}^{n \times n} \), \( I_\lambda = [\alpha, \beta] \subset \mathbb{R} \)

Sought: Those eigenpairs \((\lambda, v)\) of \(A\) such that \(\lambda \in I_\lambda\)

Start with a subspace \(Y \in \mathbb{C}^{n \times m}\) of suitable dimension \(m\)

While not yet converged

Compute \(U = f(A) \cdot Y\) for a suitable function \(f\)

Compute \(A_U = U^*A_U\) and \(B_U = U^*U\)

Solve the size-\(m\) generalized EVP \(A_UW = B_UW\Lambda\)

Replace \(Y\) with \(U \cdot W\)
Filter functions

Given an orthonormal set of eigenpairs \((x_i, \lambda_i)\) of \(A\) and an arbitrary vector \(y = \sum \xi_i x_i\), \(f(A) \cdot y\) should

- retain the “wanted” components \(\xi_i x_i, \lambda_i \in I_\lambda\), and
- cancel the “unwanted” components \(\xi_i x_i, \lambda_i \not\in I_\lambda\).

In practice, this function \(f = \chi_{I_\lambda}\) must be approximated:

- Rational approximation, e.g., FEAST
- Polynomial approximation
Subspace iteration with Rayleigh–Ritz extraction

Improving the coefficients of the polynomials

Adaptive control of the degree

A priori information about the spectrum

High performance computational kernels
Expansion w.r.t. Chebyshev polynomials of the first kind yields

\[ \chi_{[\alpha, \beta]}(x) \approx \sum_{k=0}^{d} c_k T_k(x), \]

where

\[ T_0(x) \equiv 1, \]
\[ T_1(x) = x, \]
\[ T_k(x) = 2x \cdot T_{k-1}(x) - T_{k-2}(x), \quad k \geq 2, \]

and

\[ c_0 = \frac{1}{\pi} \cdot (\arccos \alpha - \arccos \beta), \]
\[ c_k = \frac{2}{k\pi} \cdot (\sin(k \cdot \arccos \alpha) - \sin(k \cdot \arccos \beta)), \quad k \geq 1. \]
Polynomial approximation and kernel smoothing II

- **Left**: Degree-1600 Chebyshev approximation $p(x)$ to $\chi_{[\alpha,\beta]}$ for $[\alpha, \beta] = [0.238, 0.262]$ ($\sim$ Gibbs oscillations)

- **Right**: With (Lanczos, $\mu = 2$) kernel smoothing: replace $c_k$ with $g_k \cdot c_k$, where (Lanczos)

\[
g_k = \left( \text{sinc} \frac{k}{d + 1} \right)^\mu, \quad k \geq 0, \quad \text{with} \quad \text{sinc} \xi = \frac{\sin(\pi \xi)}{\pi \xi}.
\]
The target for improvement

- Light grey areas: The “damping condition” $|p(x)| \leq \tau_{\text{outside}} = 0.01$ may be violated
- Try to reduce the margin (i.e., the width of the grey areas)
- For any filter, let

$$\text{gain} = \frac{\text{margin(\text{Chebyshev approx with Lanczos kernel, } \mu = 2)}}{\text{margin(f\text{ilter under consideration})}}$$
Lanczos smoothing with optimized $\mu$

- No need to have $\mu \in \mathbb{N}$:

![Graph showing gain for $[\alpha, \beta] = [0.238, 0.262]$](image)

- The optimum $\mu$ can be determined from $\alpha$, $\beta$, and $d$ by considering $p(x)$, without evaluating $p(A)$.
Shrinking the interval I

- Determine a filter \( \hat{p}(x) \) for a smaller interval 
  \([\alpha, \beta] \mapsto [\tilde{\alpha}, \tilde{\beta}] \subseteq [\alpha, \beta]\)

- In general, \( \hat{p}(\alpha) \) and \( \hat{p}(\beta) \) will be smaller than 0.5
  ⇒ scale the polynomial,

\[
\hat{p} = \varphi \cdot \hat{p}, \quad \text{where} \quad \varphi = \frac{0.5}{\min\{\hat{p}(\alpha), \hat{p}(\beta)\}}.
\]

\([\alpha, \beta] = [0.238, 0.262], \quad [\tilde{\alpha}, \tilde{\beta}] = [0.24032, 0.25969], \quad d = 1600\]
Shrinking the interval II

How to choose \( \tilde{\alpha} \) and \( \tilde{\beta} \)?

- Let \( \sigma \geq 0 \) such that

\[
\tilde{\alpha} := \alpha + \sigma \frac{p(\alpha)}{p'(\alpha)} \leq \frac{\alpha + \beta}{2} \leq \beta + \sigma \frac{p(\beta)}{p'(\beta)} =: \tilde{\beta}
\]
Shrinking the interval III

This pattern is rather generic:

\[ [\alpha, \beta] = [-0.984, -0.960], \quad d = 400 \]
\[ [\alpha, \beta] = [0.560, 0.584], \quad d = 1131 \]

\[ [\alpha, \beta] = [-0.012, 0.012], \quad d = 1600 \]
\[ [\alpha, \beta] = [0.150, 0.350], \quad d = 200 \]
Shrinking the interval IV

There are three qualitatively different patterns:

\[ [α, β] = [0.238, 0.262] \]
\[ d = 141 \]
“low degree”

\[ [α, β] = [0.238, 0.262] \]
\[ d = 565 \]
“critical degree”

\[ [α, β] = [0.238, 0.262] \]
\[ d = 1600 \]
“high degree”
Shrinking the interval $V$

In all cases, the best $\text{gain}(\mu, \delta)$ is found close to the diagonal $\log_2(\mu) = \sigma$:

- Search on a grid along the BAND
- Optionally followed by refined search (PATH)
Start with a suitable target function $f_1$ (e.g., remove the upper corners of the window)

Determine approximation (dash-dotted) and scale to achieve $\min\{p_1(\alpha), p_1(\beta)\} = 0.5$ (solid)
“Compensate” for the oscillations by prescribing $f_2(x) = -\rho \cdot p_1(x)$ outside $[\alpha, \beta]$ (thick line) and determine new approximation (thin line)

We used $\rho = 0.75$

Iterate until no more improvement

Resulting filter function $p = p_{34}$

- No closed formula for the $c_k$ in the expansion ($\sim$ quadrature)
Numerical results I

Matlab: Number of overall MVMs vs. Lanczos ($\mu = 2$)

- Dotted: Shrunken Lanczos (BAND)
- Solid thin: Shrunken Lanczos (BAND and PATH)
- Dash-dotted: Iteratively compensating
- Solid thick: Combined
Runs on Emmy (two 2.2GHz 10-core Xeon 2260v2 per node) at Erlangen Regional Computing Center

<table>
<thead>
<tr>
<th>Filter</th>
<th>Final degree</th>
<th>Overall MVMs</th>
<th>Overall time</th>
<th>Time for coeffs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topological insulator, (n = 268,435,456), 148 evals, 128 nodes à 20 cores</td>
<td>4,525</td>
<td>5,598,502</td>
<td>7.11 h</td>
<td>0.00 h</td>
</tr>
<tr>
<td>Lanczos ((\mu = 2))</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improved (combined)</td>
<td>2,255</td>
<td>2,602,360</td>
<td>3.44 h</td>
<td>0.02 h</td>
</tr>
<tr>
<td>Topological insulator, (n = 67,108,864), 148 evals, 64 nodes à 20 cores</td>
<td>2,262</td>
<td>2,726,112</td>
<td>1.97 h</td>
<td>0.00 h</td>
</tr>
<tr>
<td>Lanczos ((\mu = 2))</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improved (combined)</td>
<td>1,127</td>
<td>1,482,035</td>
<td>1.10 h</td>
<td>0.01 h</td>
</tr>
</tbody>
</table>
Outline

Subspace iteration with Rayleigh–Ritz extraction

Improving the coefficients of the polynomials

Adaptive control of the degree

A priori information about the spectrum

High performance computational kernels
Overall MVM count for \textit{linverse} \((n = 11,999)\)

(Same matrix, both intervals contain roughly 300 eigenvalues)
How to choose the degree without prior knowledge? II

- Run the algorithm for different fixed degrees
- Count overall MVMs for each run
- Determine “drop” (of smallest residual) just before convergence sets in
- Determine the drops that lead to “close-to-best” MVM counts

⇒ most close-to-best runs achieved $\text{drop} \in [10^{-2.5}, 10^{-1.5}]$
Can going for a drop $\in [10^{-2.5}, 10^{-1.5}]$ be dangerous?

Determine “MVM overhead” for all runs that reached such drops

$\Rightarrow$ at most $20\%$ more MVMs than the best fixed-degree run
How to choose the degree without prior knowledge? IV

- Increase degree for the next iteration if \( \text{drop} > 10^{-2} \)

![Graph showing overall MVMs for adaptive and best fixed runs.]

- On average 14% more MVMs than the best fixed-degree run
- In two cases \( \geq 30\% \) more MVMs
Subspace iteration with Rayleigh–Ritz extraction

Improving the coefficients of the polynomials

Adaptive control of the degree

A priori information about the spectrum

High performance computational kernels
Many applications require (approximate) density of states (DOS) of $A$, 

$$
\rho(\lambda) = \frac{1}{n} \sum_{k=1}^{n} \delta(\lambda - \lambda_k)
$$

The Kernel Polynomial Method (KPM):

- Moment expansion of $\rho$:

$$
\rho(x) = \mu_0 \phi_0(x) + 2 \sum_{m=1}^{\infty} \mu_m \phi_m(x),
$$

where

$$
\phi_m(\xi) = \frac{T_m(\xi)}{\pi \sqrt{1 - \xi^2}}
$$

and
Estimating the number of eigenvalues in $I_\lambda$: The KPM II

$$
\mu_m = \langle \rho, \phi_m \rangle = \int_{-1}^{+1} \rho(\xi) T_m(\xi) d\xi = \frac{1}{n} \text{trace}(T_m(A)),
$$

with

$$
\text{trace}(T_m(A)) \approx \frac{1}{R} \sum_{r=1}^{R} r_r^* T_m(A) r_r
$$

($r_r$: suitable random vectors)

- Once you have the $\mu_m$, evaluating $\rho$ is easy (and cheap: FFT-type)

- The KPM uses the same Chebyshev kernel, with a few inner products after each MVM
Selecting a suitable subspace dimension and degree

- For certain distributions of eigenvalues (KPM), e.g.,
  - “flat”
  - “linearly increasing” from a (pseudo-)gap in $I_\lambda$

“good” values for

- $m$ (size of the search space)
- $d$ (degree)

can be derived

- “Over-populating” (selecting $m \gg \#\text{evals}$) may reduce the overall MVM count

$\rightsquigarrow$ A. Pieper et al., arXiv:1510.04895
Outline

Subspace iteration with Rayleigh–Ritz extraction

Improving the coefficients of the polynomials

Adaptive control of the degree

A priori information about the spectrum

High performance computational kernels
High performance computational kernels

- Several kernels occur in different eigensolvers:
  - Sparse matrix times (block) vector
  - Apply \( p(A) \) to a (block) vector
  - Orthogonalize columns of a block vector
  - ...

- Provide optimized versions for these
GHOST (General, Hybrid and Optimized Sparse Toolkit) provides

- shifted $sp(M)MVM$, augmented with dot products
- operations with dense block vectors (dense and scattered “views” to avoid copying)
- real and complex, single and double precision
- support for CPU, Phi, Nvidia (also in combination)
- possibility to specify “common” dimensions at compile time
  $\Rightarrow$ highly optimized kernels
- task management (e.g., for asynchronous checkpointing)

PHIST (Pipelined Hybrid-parallel Iterative Solver Toolkit) provides an abstraction layer and higher-level functionality (orthogonalization, etc.)
The SELL-$C$-$\sigma$ format

Combine slicing and sorting:

SELL-6-1
aka SELL
$\beta = 0.51$

SELL-6-24
$\beta = 0.84$

SELL-6-12
$\beta = 0.66$

SELL-1-1
aka CRS
$\beta = 1.00$
The SELL-$C$-$\sigma$ format

Combine slicing and sorting:

- SELL-6-1 aka SELL
  \[ \beta = 0.51 \]
- SELL-6-24
  \[ \beta = 0.84 \]
- SELL-6-12
  \[ \beta = 0.66 \]
- SELL-1-1 aka CRS
  \[ \beta = 1.00 \]