UNIVERSITY OF MINNESOTA TWIN CITIES

Divide and conquer algorithms and software for large Hermitian eigenvalue problems

Yousef Saad

Department of Computer Science and Engineering

University of Minnesota

Computational methods for quantum systems

C.R.M., Montreal, 12/15/2018

Solving large interior eigenvalue problems

Three broad approaches:

1. Shift-invert: $A \longrightarrow (A - \sigma I)^{-1}$
2. Polynomial filtering: $A \longrightarrow p(A)$
3. Rational filtering: $A \rightarrow \sum \alpha_i (A - \sigma_i I)^{-1}$

Main issue with shift-and invert (and related approaches)

➤ Direct methods may be too expensive – don't scale well

➤ Could use iterative methods .... But these do not always work – because :

- Systems are highly indefinite

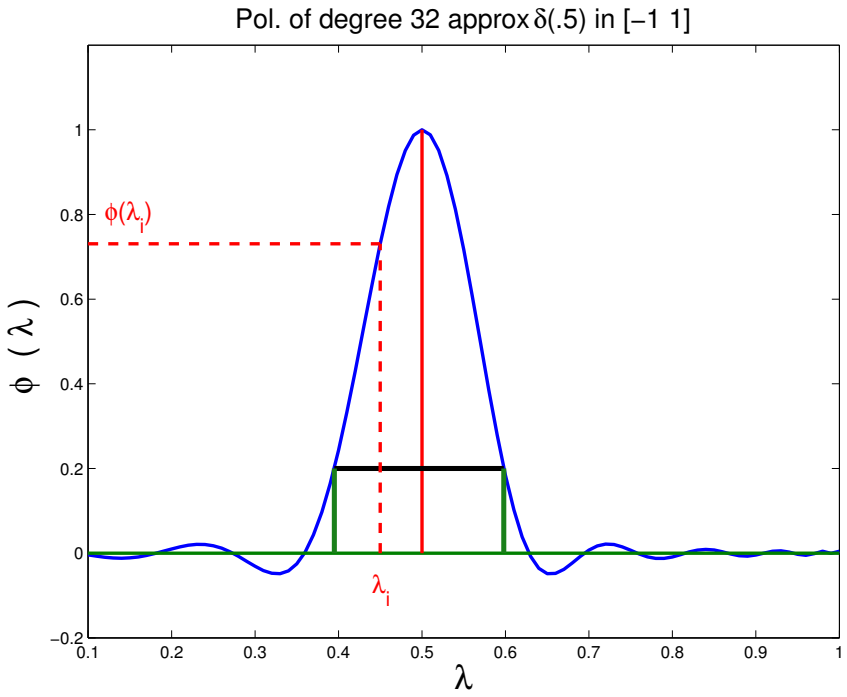
➤ Alternative: 'Spectrum slicing' with filtering

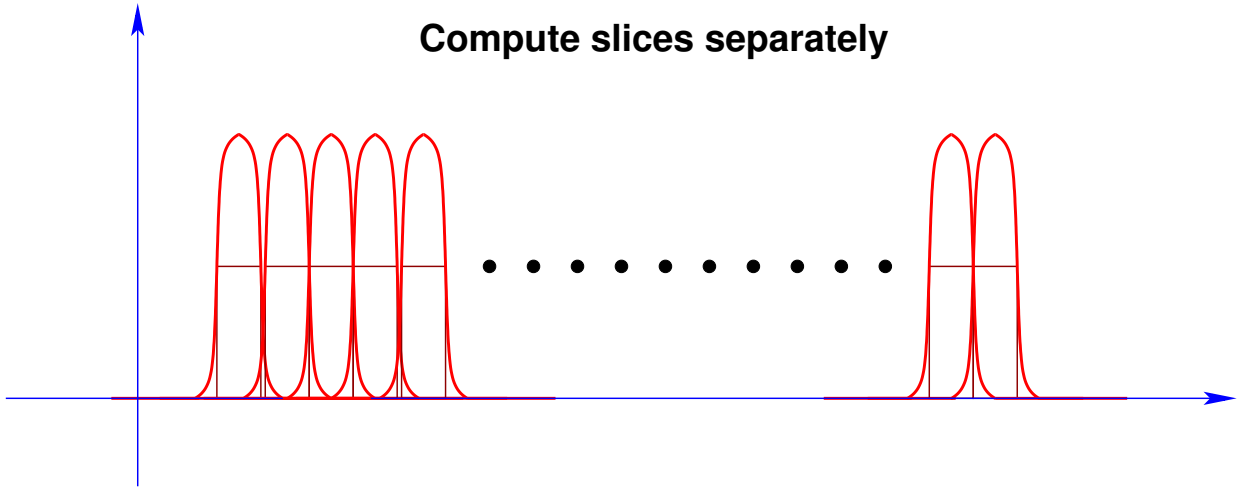
Filtering and "Spectrum Slicing"

- Context: very large number of eigenvalues to be computed
- Goal: compute spectrum by slices by applying filtering
- Apply Lanczos or Sub-space iteration to problem:

$$\phi(A)u = \mu u$$

$\phi(t)$ $\equiv$ a polynomial or rational function that enhances wanted eigenvalues





- For each slice Do:
  [get *all* eigenpairs in a slice]
  EndDo

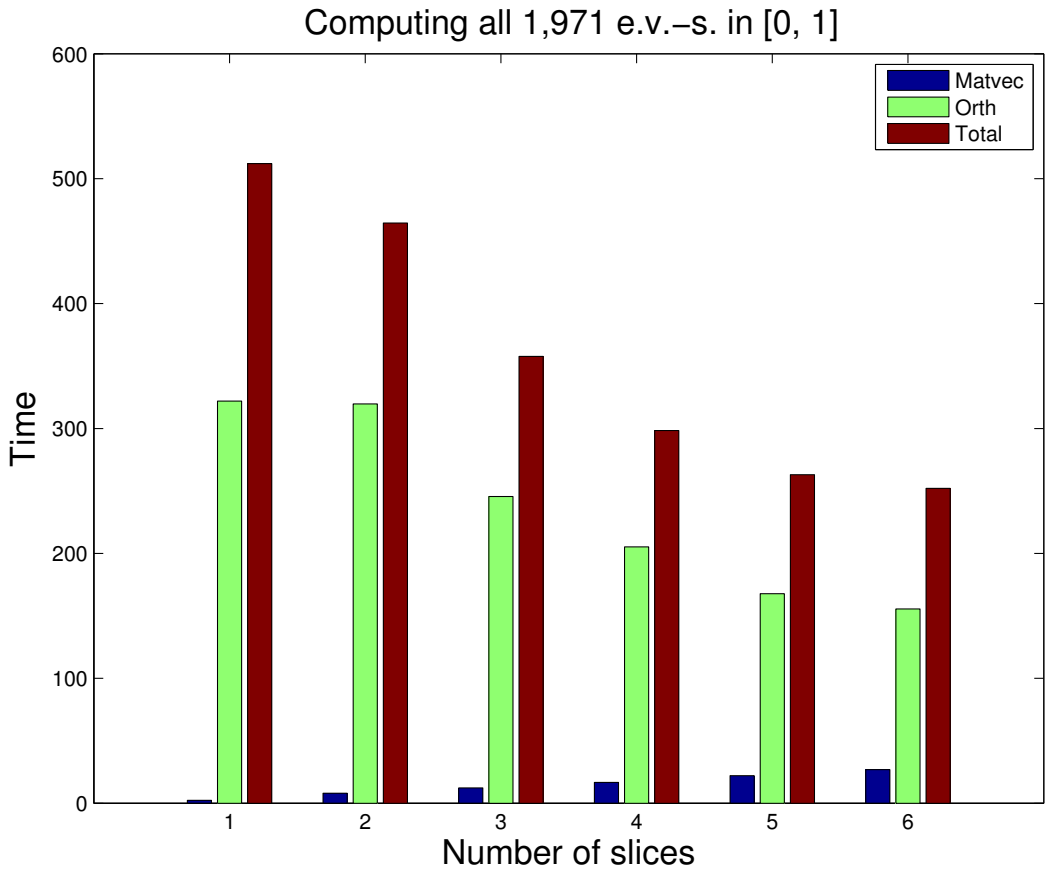
Goal: Compute each slice independently from the others.

Rationale. Eigenvectors associated with different slices need not be orthogonalized against each other :

➤ Can get the spectrum by ‘slices’ or ’windows’ [e.g., a few hundreds or thousands of pairs at a time]

➤ Note: Orthogonalization + RR cost can be very high if we do not slice the spectrum

Illustration: All eigenvalues in [0, 1] of a 49^3 Laplacean



Note: This is a small pb. in a scalar environment. Effect likely much more pronounced in a fully parallel case.

POLYNOMIAL FILTERS

Polynomial filtering

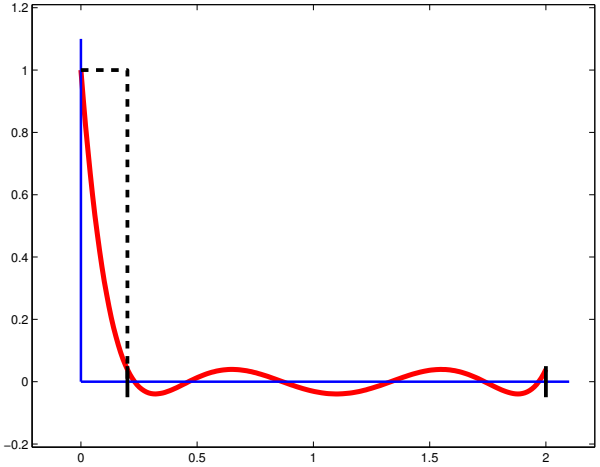
- Apply Lanczos or Subspace iteration to: $M = \phi(A)$ where $\phi(t)$ is a polynomial
- Each *matvec* $y = Av$ is replaced by $y = \phi(A)v$
- Eigenvalues in high part of filter will be computed first
- Old (forgotten) idea. But new context is *very* favorable

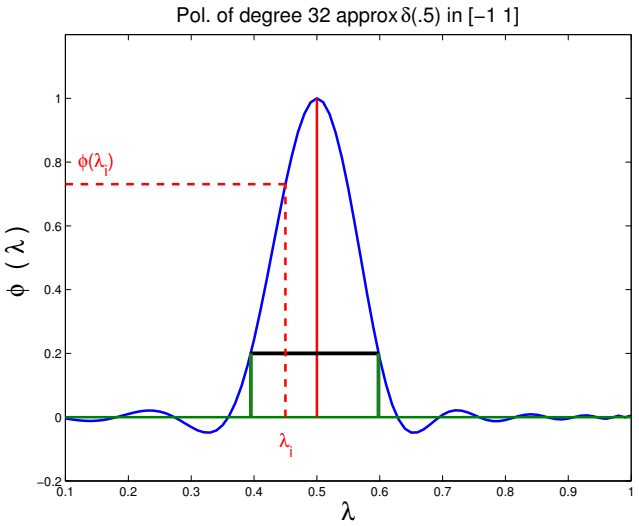
What polynomials?

- For end-intervals: use standard Chebyshev polynomials (1st kind)

- For 'interior case' we need *a polynomial that has large values for* $\lambda \in [a, b]$ *small values elsewhere*

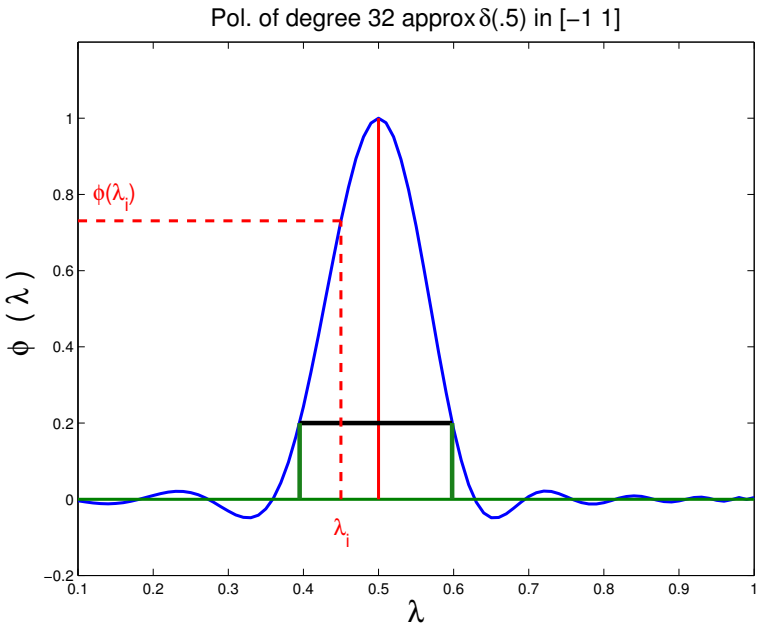
Deg. 6 Cheb. polynom., damped interv=[0.2, 2]

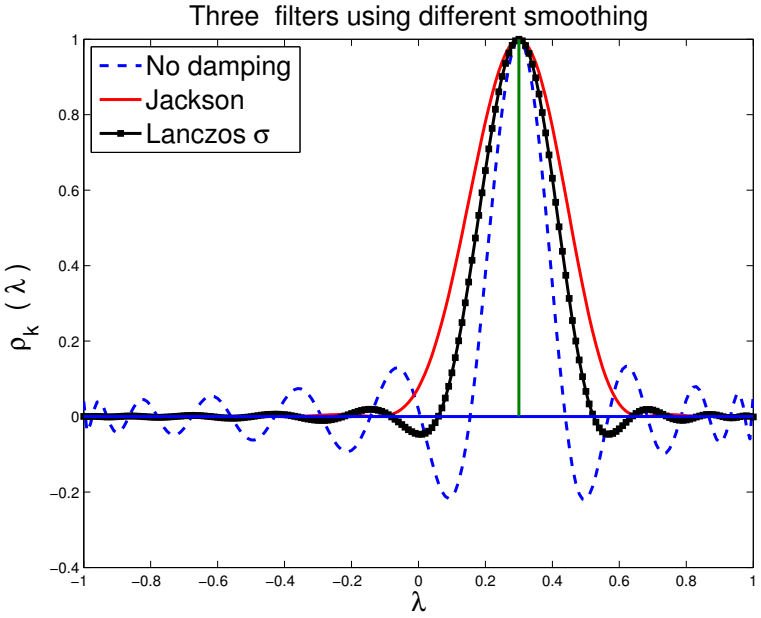




Simplest technique: δ-Dirac function

➤ Obtain the LS approximation to the δ— Dirac function – Centered at some point (TBD) inside the interval. ⟶





⟵ Can use same damping: Jackson, Lanczos σ damping, or none.

Theory

The Chebyshev expansion of δ_γ is

$$\rho_k(t) = \sum_{j=0}^{k} \mu_j T_j(t) \text{ with } \mu_j = \begin{cases} \frac{1}{2} & j = 0 \\ \cos(j \cos^{-1}(\gamma)) & j > 0 \end{cases}$$

➤ Recall: The delta Dirac function is not a function – we can't properly approximate it in least-squares sense. However:

Proposition Let $\hat{\rho}_k(t)$ be the polynomial that minimizes $\|r(t)\|_w$ over all polynomials r of degree $\leq k$, such that $r(\gamma) = 1$, where $\|.\|_w$ represents the Chebyshev L^2-norm. Then $\hat{\rho}_k(t) = \rho_k(t)/\rho_k(\gamma)$.

'The soul of a new filter' – A few technical details

$$p_m(t) = \sum_{j=0}^{m} \gamma_j^{(m)} \mu_j T_j(t)$$

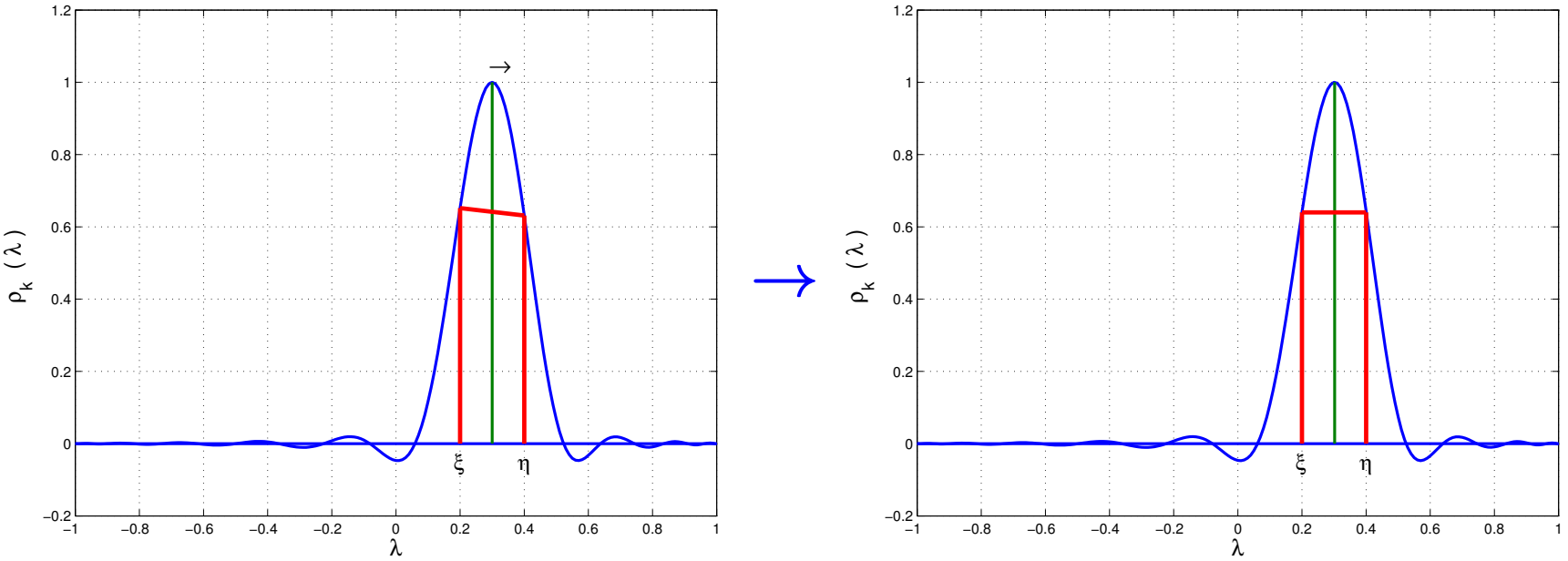
$$\mu_k = \begin{cases} 1/2 & \text{if } k == 0 \\ \cos(k \cos^{-1}(\gamma)) & \text{otherwise} \end{cases}$$

$\gamma_j^{(m)}$ = Damping coefficients.

- quite simple...
- .. provided we handle a few practical issues

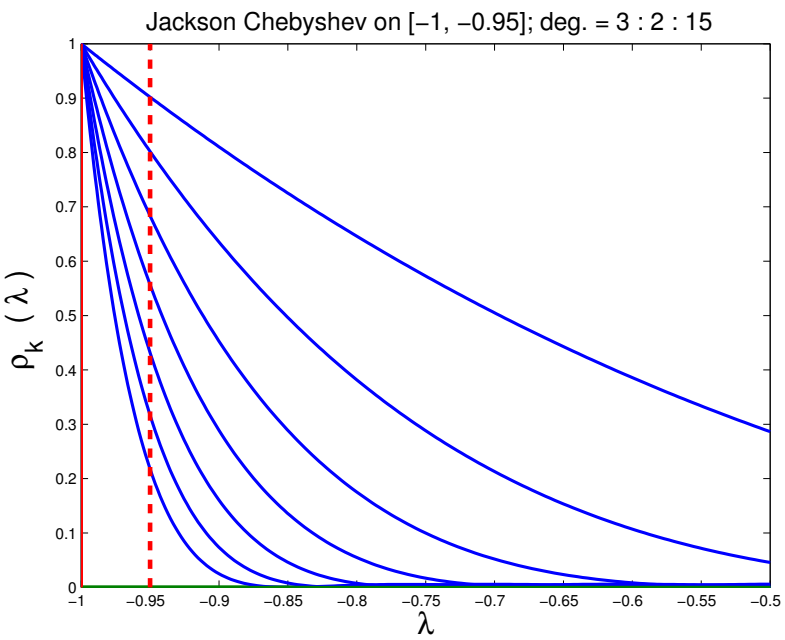
Issue # one: 'balance the filter'

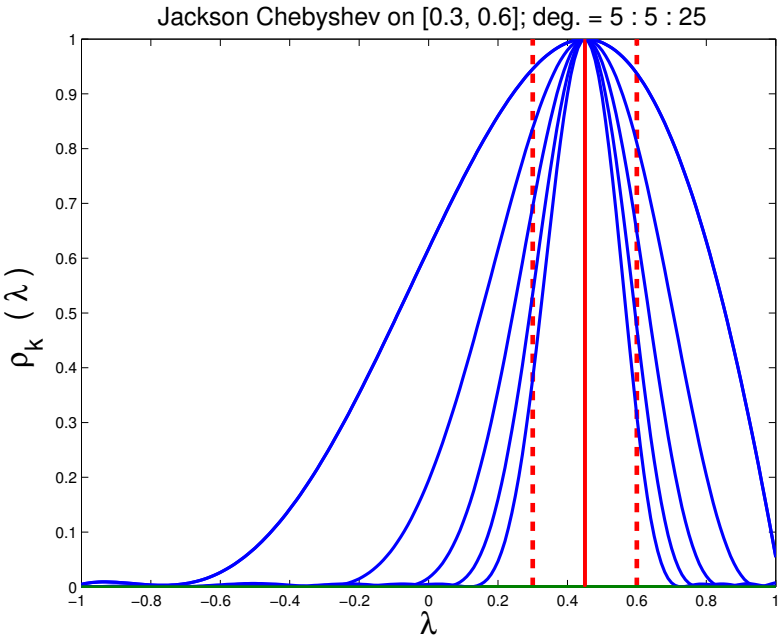
- To facilitate the selection of *'wanted'* eigenvalues [Select λ's such that $\phi(\lambda) >$ bar] we need to ...
- ... find γ so that $\phi(\xi) == \phi(\eta)$



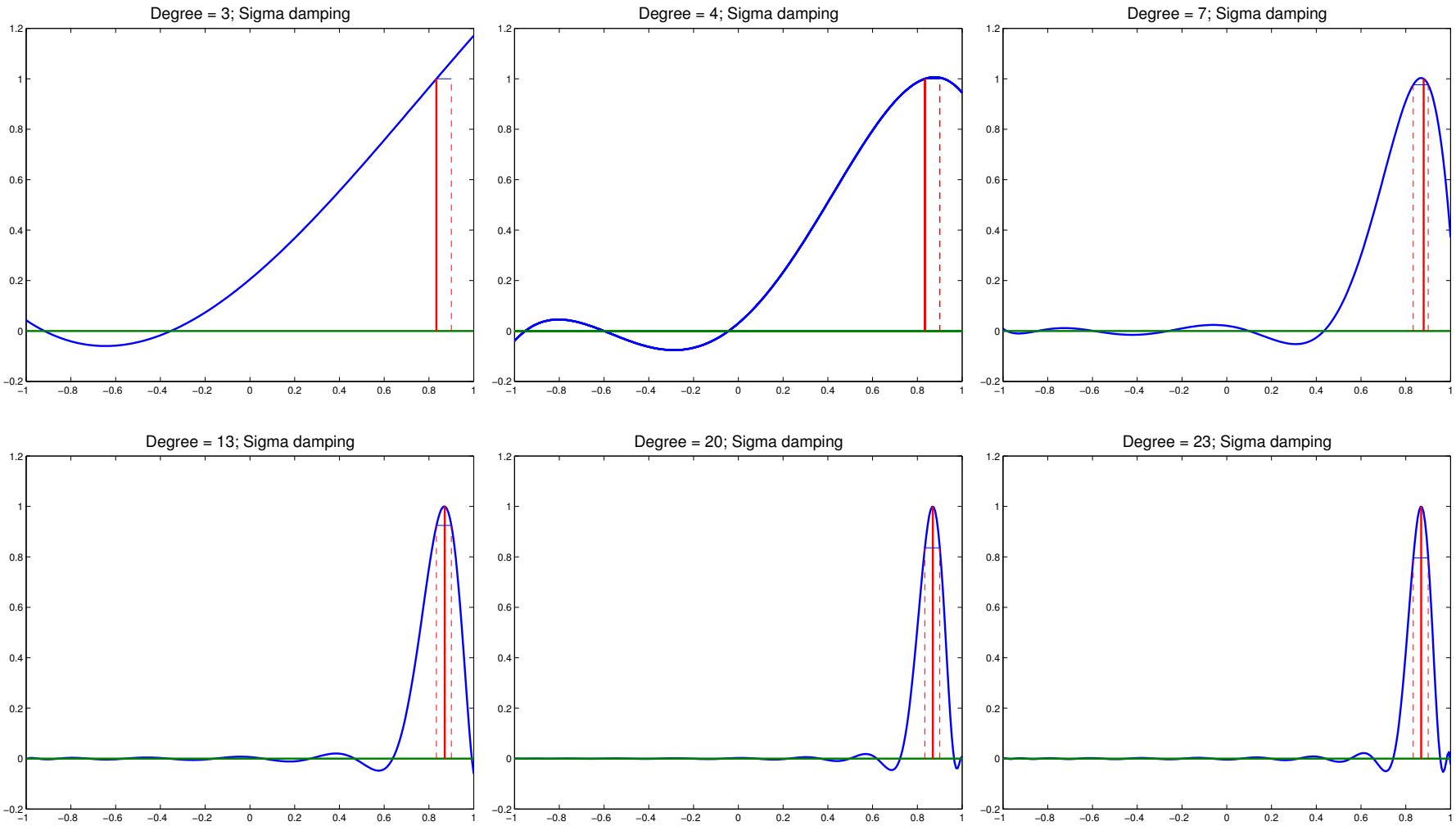
Procedure: Solve the equation $\phi_\gamma(\xi) - \phi_\gamma(\eta) = 0$ with respect to γ, accurately. Use Newton or eigenvalue formulation.

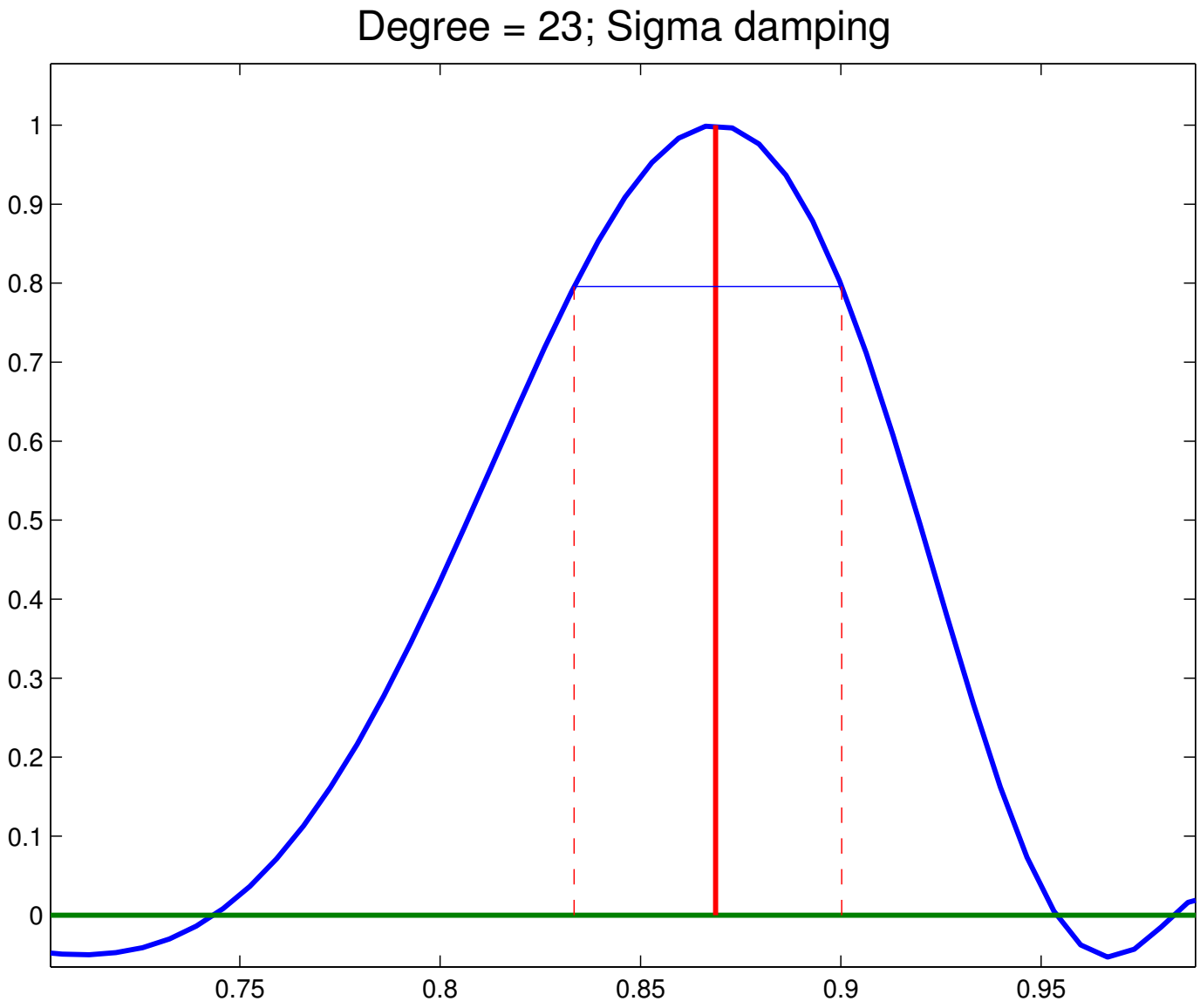
Issue # two: Determine degree & polynomial (automatically)





- 1) Start low (e.g. 2); 2) Increase degree until value (s) at the boundary (ies) become small enough –
- Can also use criterion based on derivatives at ξ & η


Degree = 3; Sigma damping
Degree = 4; Sigma damping
Degree = 7; Sigma damping
Degree = 13; Sigma damping
Degree = 20; Sigma damping
Degree = 23; Sigma damping

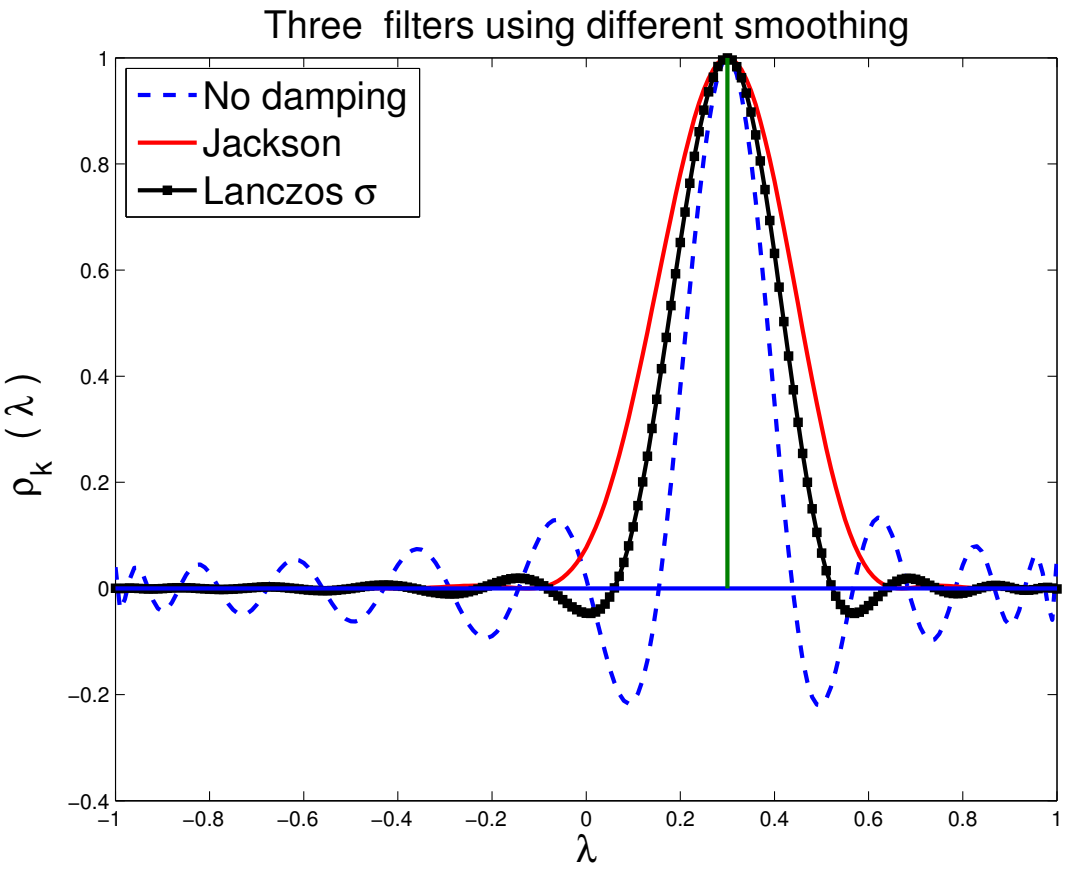


A zoom on the final polynomial found

Issue # Three : Gibbs oscillations

- Discontinuous 'function' approximated → Gibbs oscillations

- Three options:
  - No damping
  - Jackson damping
  - Lanczos σ damping



- Good compromise: Lanczos σ damping

USING A PROJECTION METHOD

Backround: The Lanczos Algorithm

➤ Algorithm builds orthonormal basis $V_m = [v_1, v_2, \cdots, v_m]$ for the Krylov subspace: $\mathbf{span}\{v_1, Av_1, \cdots, A^{m-1}v_1\}$

➤ ... such that: $V_m^H A V_m = T_m$ - with

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \beta_4 & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \cdot \\ & & & & \beta_m & \alpha_m \end{pmatrix}$$

➤ Note: three term recurrence:

$$\beta_{j+1} v_{j+1} = A v_j - \alpha_j v_j - \beta_j v_{j-1}$$

➤ Eigenvalues of A on both ends of spectrum are well approximated by eigenvalues of T_m (Ritz values).

Which Projection: Lanczos,w/o restarts, Subspace iteration,..

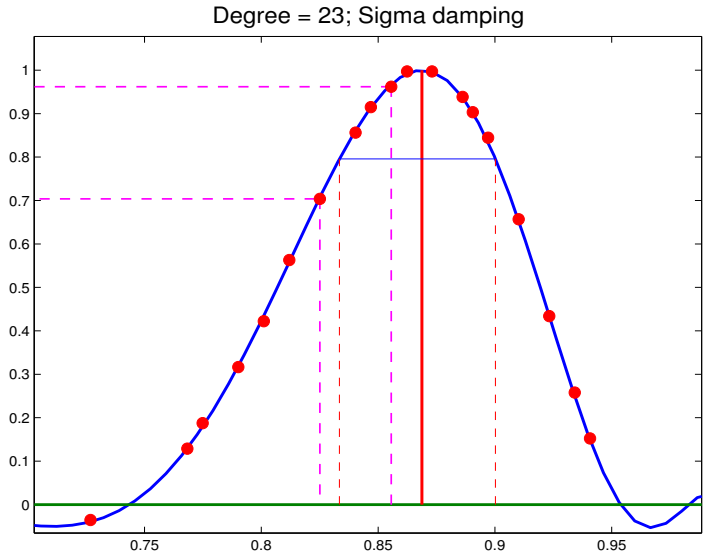
Options:

- Subspace iteration: quite appealing in some applications (e.g., electronic structure): Can re-use previous subspace.
- Simplest: (+ most efficient) Lanczos without restarts
- Lanczos with Thick-Restarting [TR Lanczos, Stathopoulos et al '98, Wu & Simon'00]
- Crucial tool in TR Lanczos: deflation ('Locking')

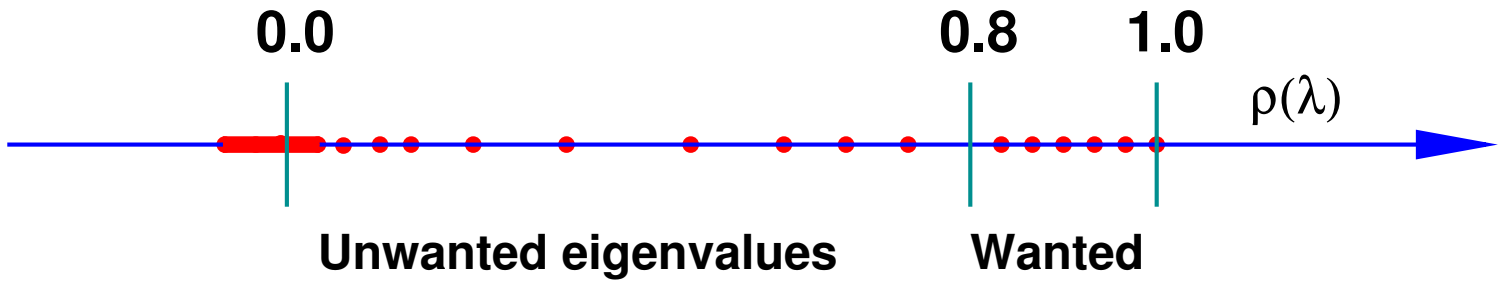
Main idea: *Keep extracting eigenvalues in interval* $[\xi, \eta]$ *until none are left.*

- If filter is good: Can catch all eigenvalues in interval

Polynomial filtered Lanczos: No-Restart version



- Use Lanczos with full reorthogonalization on $\rho(A)$. Eigenvalues of $\rho(A)$: $\rho(\lambda_i)$
- Accept if $\rho(\lambda_i) \geq$ `bar`
- Ignore if $\rho(\lambda_i) <$ `bar`



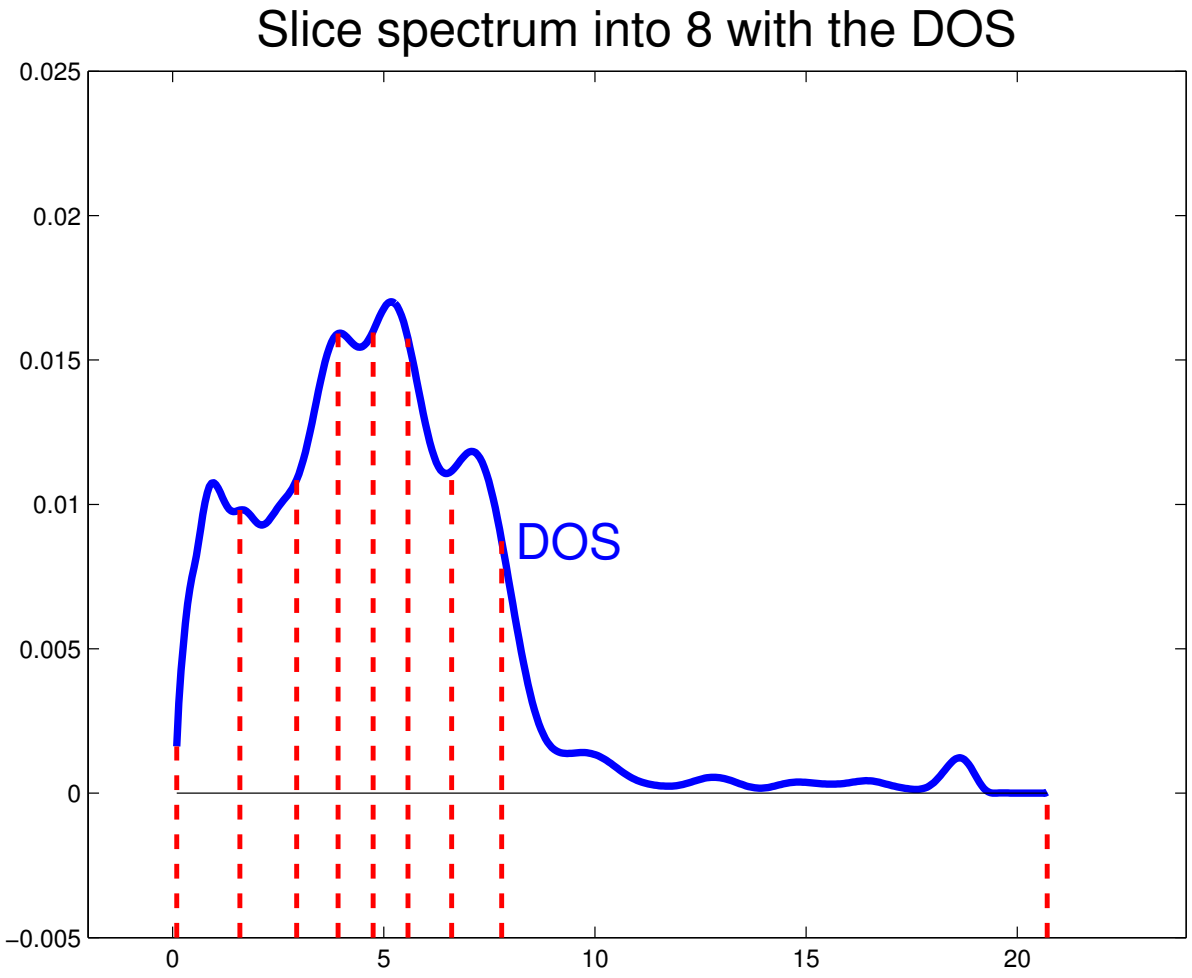
How do I slice a spectrum?

➤ Tools: Density of States (used in EVSL) or eigenvalue counts (used in FEAST)

- L. Lin, YS, Chao Yang [Siam review '16] – E. Di Napoli, E. Polizzi, YS ['16]
- KPM method – see, e.g., : *[Weisse, Wellein, Alvermann, Fehske, '06]*
- Interesting instance of a tool from physics used in linear algebra.

➤ Misconception: *'load balancing will be assured by just having slices with roughly equal numbers of eigenvalues'*

➤ In fact - will help mainly in balancing memory usage..



- We must have:

$$\int_{t_i}^{t_{i+1}} \phi(t)dt = \frac{1}{n_{slices}} \int_a^b \phi(t)dt$$

What about matrix pencils?

- DOS for generalized eigenvalue problems

$$Ax = \lambda Bx$$

- Assume: A is symmetric and B is SPD.
- In principle: can just apply methods to $B^{-1}Ax = \lambda x$, using B - inner products.
- Requires factoring B. Too expensive [Think 3D Pbs]

⋆ *Observe:* B is usually very *strongly* diagonally dominant.

- Especially true after Left+Right Diag. scaling :

$$\tilde{B} = S^{-1}BS^{-1} \qquad S = diag(B)^{1/2}$$

General observation for FEM mass matrices [See, e.g., Wathen'87, Wathen Rees '08]:
* Conforming tetrahedral (P1) elements in 3D $\rightarrow \kappa(\tilde{B}) \leq 5$
* Rectangular bilinear (Q1) elements in 2D $\rightarrow \kappa(\tilde{B}) \leq 9$.

Example: Matrix pair *Kuu, Muu* from Suite Sparse collection.

➤ Matrices A and B have dimension $n = 7,102$. $\texttt{nnz}(A) = 340,200$ $\texttt{nnz}(B) = 170,134$.

➤ After scaling by diagonals to have diag. entries equal to one, all eigenvalues of B are in interval

$$[0.6254,\ 1.5899]$$

Approximation theory to the rescue.

⋆ *Idea:* Compute the DOS for the standard problem

$$B^{-1/2}AB^{-1/2}u = \lambda u$$

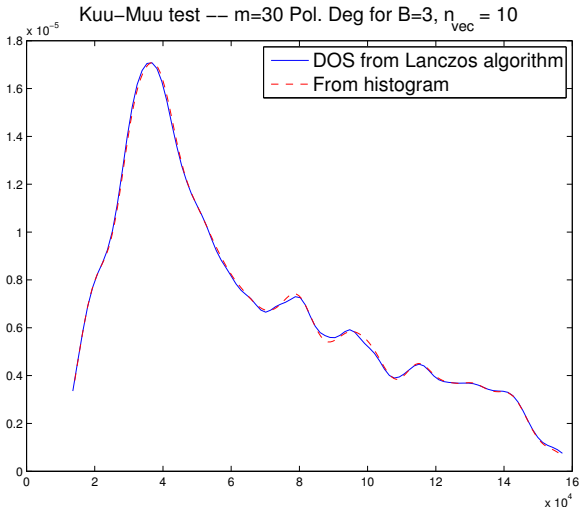
- Use a very low degree polynomial to approximate $B^{-1/2}$.
- We use Chebyshev expansions.
- Degree k determined automatically by enforcing

$$\|t^{-1/2} - p_k(t)\|_\infty < tol$$

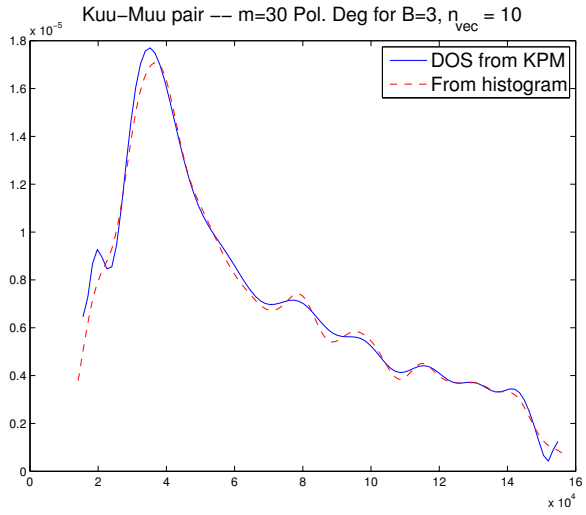
- Theoretical results establish convergence that is exponential with respect to degree.

Example: Results for Kuu-Muu example

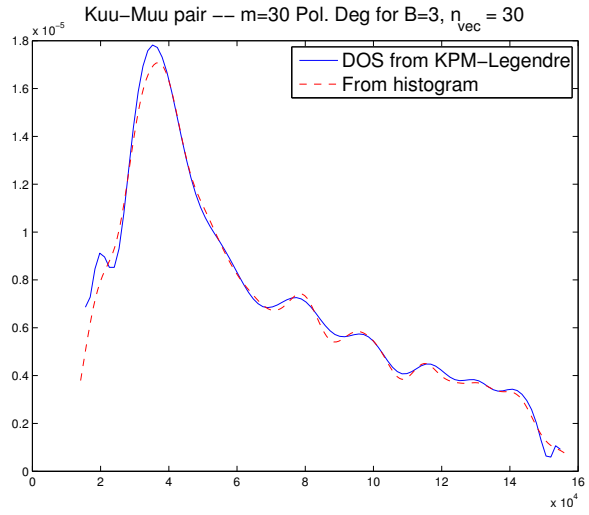
- Using polynomials of degree 3 (!) to approximate $B^{-1/2}$
- Krylov subspace of dim. 30 (== deg. of polynomial in KPM)
- 10 Sample vectors used



Lanczos



KPM-Chebyshev



KPM-Legendre

A digression: The KPM method

➤ Formally, the Density Of States (DOS) of a matrix A is

$$\phi(t) = \frac{1}{n} \sum_{j=1}^{n} \delta(t - \lambda_j),$$

where
- δ is the Dirac δ-function or Dirac distribution
- $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of A

➤ $\phi(t)$ == a probability distribution function == probability of finding eigenvalues of A in a given infinitesimal interval near t.

➤ Also known as the spectral density

➤ Very important uses in Solid-State physics

The Kernel Polynomial Method

- Used by Chemists to calculate the DOS – see Silver and Röder'94 , Wang '94, Drabold-Sankey'93, + others
- Basic idea: expand DOS into Chebyshev polynomials
- Exploits trace estimators [discovered independently]
- Next: A few details
- Assume change of variable done so eigenvalues lie in $[-1,\ 1]$.
- Include the weight function in the expansion so expand:

$$\hat{\phi}(t) = \sqrt{1-t^2}\phi(t) = \sqrt{1-t^2} \times \frac{1}{n}\sum_{j=1}^{n} \delta(t-\lambda_j).$$

- Then, (full) expansion is: $\hat{\phi}(t) = \sum_{k=0}^{\infty} \mu_k T_k(t)$.
- Expansion coefficients μ_k are formally defined by:

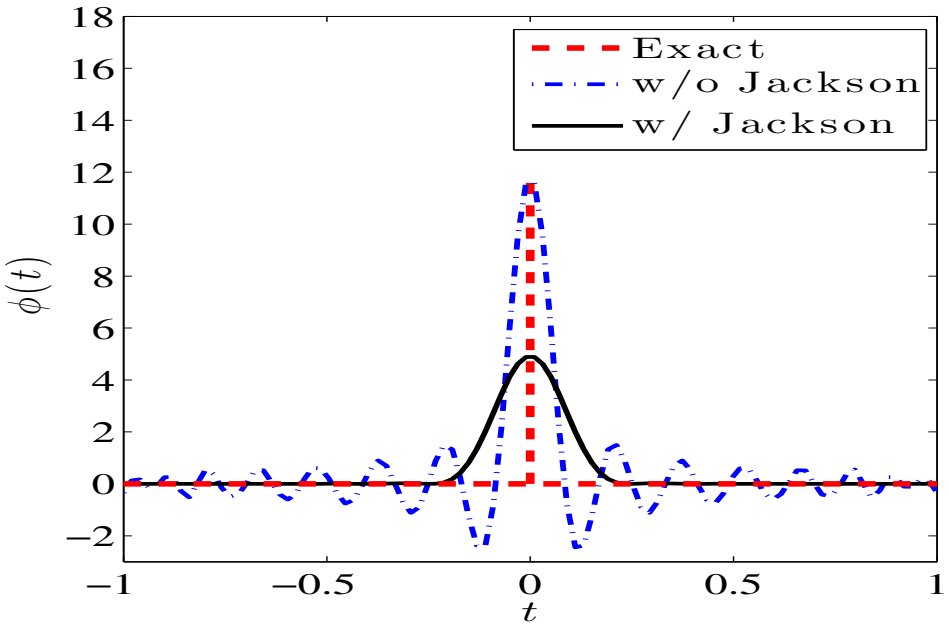
$$\begin{aligned}
\mu_k &= \frac{2-\delta_{k0}}{\pi} \int_{-1}^{1} \frac{1}{\sqrt{1-t^2}} T_k(t)\hat{\phi}(t)dt \\
&= \frac{2-\delta_{k0}}{\pi} \int_{-1}^{1} \frac{1}{\sqrt{1-t^2}} T_k(t)\sqrt{1-t^2}\phi(t)dt \\
&= \frac{2-\delta_{k0}}{n\pi} \sum_{j=1}^{n} T_k(\lambda_j), \qquad (\delta_{ij} = \text{Dirac symbol})
\end{aligned}$$

- Note: $\sum T_k(\lambda_i) = Trace[T_k(A)]$
- Estimate this, e.g., via stochastic estimator

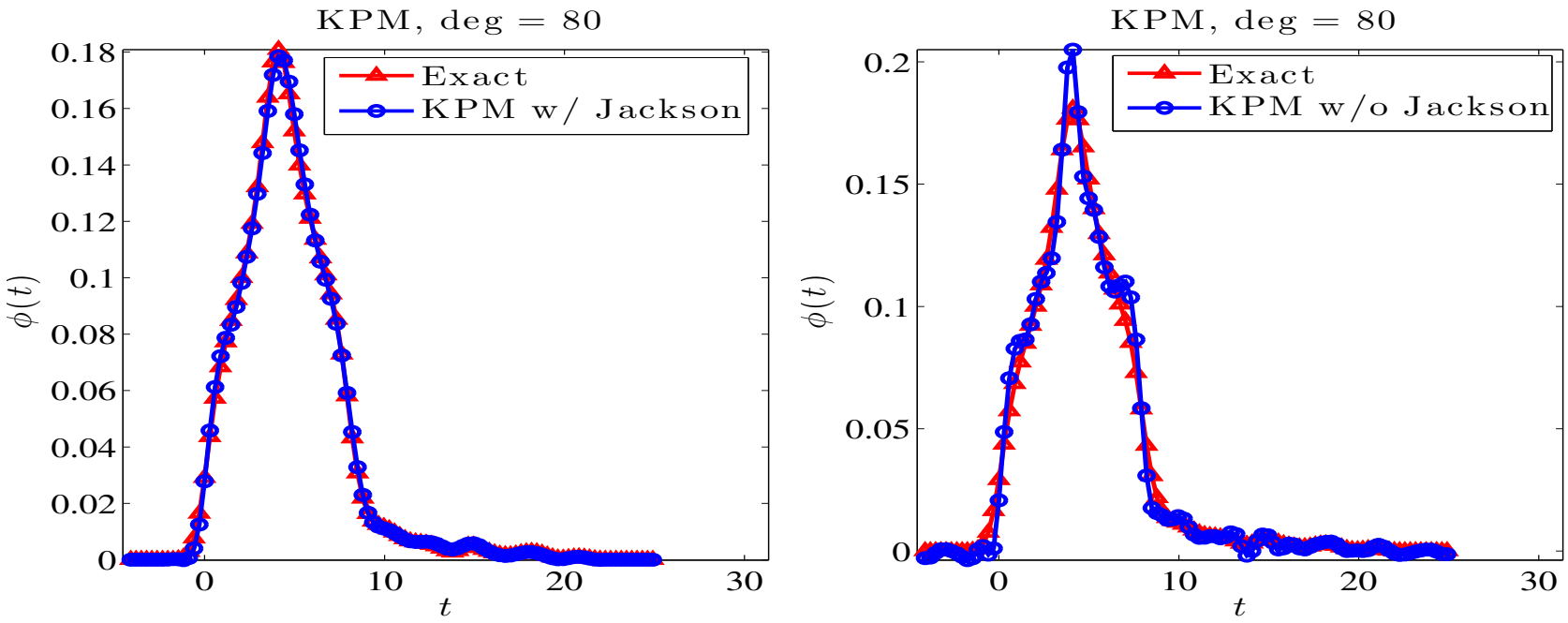
$$\text{Trace}(T_k(A)) \approx \frac{1}{n_{\text{vec}}} \sum_{l=1}^{n_{\text{vec}}} \left(v^{(l)}\right)^T T_k(A) v^{(l)}.$$

➤ To compute scalars of the form $v^T T_k(A)v$, exploit again 3-term recurrence of the Chebyshev polynomial ...

➤ Same Jackson smoothing as before can be used



An example with degree 80 polynomials



Left: Jackson damping; right: without Jackson damping.

Experiments

3D discrete Laplacian example ($60^3 \rightarrow n = 216,000$) *Used* $\phi = 0.8$. *Partitioning* $[0.6, 1.2]$ *into* 10 *sub-intervals.* ➤ *Goal: compute all 3,406 eigenvalues in interval [0.6, 1.2]*

i	$[\xi_i, \eta_i]$	$\eta_i - \xi_i$	$\nu_{[\xi_i,\eta_i]}$
1	$[0.60000, 0.67568]$	0.07568	337
2	$[0.67568, 0.74715]$	0.07147	351
3	$[0.74715, 0.81321]$	0.06606	355
4	$[0.81321, 0.87568]$	0.06247	321
5	$[0.87568, 0.93574]$	0.06006	333
6	$[0.93574, 0.99339]$	0.05765	340
7	$[0.99339, 1.04805]$	0.05466	348
8	$[1.04805, 1.10090]$	0.05285	339
9	$[1.10090, 1.15255]$	0.05165	334
10	$[1.15255, 1.20000]$	0.04745	348

Results

i	deg	iter	matvec	CPU time (sec)		residual	
				matvec	total	max	avg
1	116	1814	210892	430.11	759.24	6.90×10^{-09}	7.02×10^{-11}
2	129	2233	288681	587.14	986.67	5.30×10^{-09}	7.39×10^{-11}
3	145	2225	323293	658.44	1059.57	6.60×10^{-09}	5.25×10^{-11}
4	159	1785	284309	580.09	891.46	3.60×10^{-09}	4.72×10^{-11}
5	171	2239	383553	787.00	1180.67	6.80×10^{-09}	9.45×10^{-11}
6	183	2262	414668	848.71	1255.92	9.90×10^{-09}	1.13×10^{-11}
7	198	2277	451621	922.64	1338.47	2.30×10^{-09}	3.64×10^{-11}
8	209	1783	373211	762.39	1079.30	8.50×10^{-09}	1.34×10^{-10}
9	219	2283	500774	1023.24	1433.04	4.30×10^{-09}	4.41×10^{-11}
10	243	1753	426586	874.11	1184.76	5.70×10^{-09}	1.41×10^{-11}

Note: # of eigenvalues found inside each $[\xi_i, \eta_i]$ *is exact.*

Hamiltonian matrices from the PARSEC set

Matrix	n	$\sim$ nnz	$[a, b]$	$[\xi, \eta]$	$\nu_{[\xi,\eta]}$
$Ge_{87}H_{76}$	$112,985$	$7.9M$	$[-1.21, 32.76]$	$[-0.64, -0.0053]$	212
$Ge_{99}H_{100}$	$112,985$	$8.5M$	$[-1.22, 32.70]$	$[-0.65, -0.0096]$	250
$Si_{41}Ge_{41}H_{72}$	$185,639$	$15.0M$	$[-1.12, 49.82]$	$[-0.64, -0.0028]$	218
$Si_{87}H_{76}$	$240,369$	$10.6M$	$[-1.19, 43.07]$	$[-0.66, -0.0300]$	213
$Ga_{41}As_{41}H_{72}$	$268,096$	$18.5M$	$[-1.25, 1301]$	$[-0.64, -0.0000]$	201

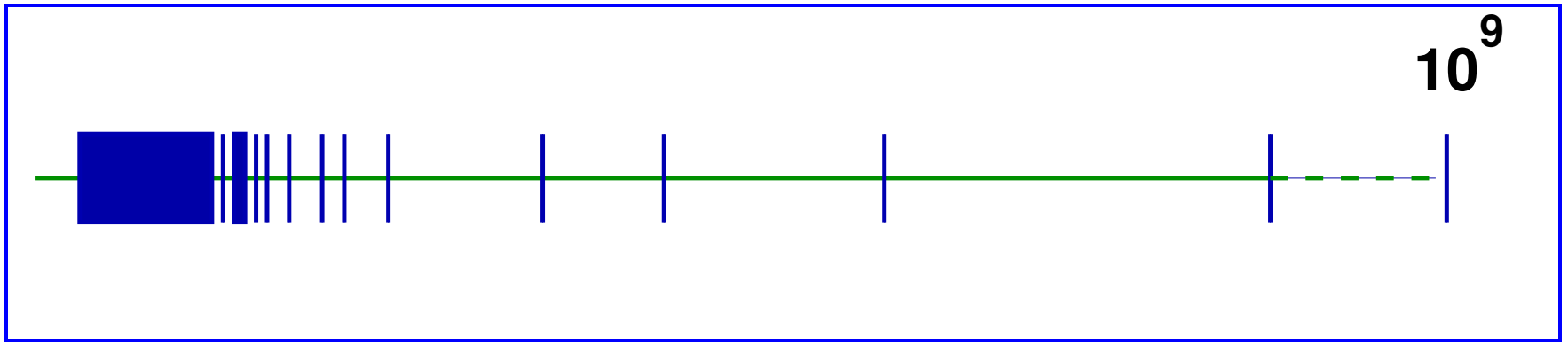
Numerical results for PARSEC matrices

Matrix	deg	iter	matvec	CPU time (sec)		residual	
				matvec	total	max	avg
$Ge_{87}H_{76}$	26	1431	37482	282.70	395.91	9.40×10^{-09}	2.55×10^{-10}
$Ge_{99}H_{100}$	26	1615	42330	338.76	488.91	9.10×10^{-09}	2.26×10^{-10}
$Si_{41}Ge_{41}H_{72}$	35	1420	50032	702.32	891.98	3.80×10^{-09}	8.38×10^{-11}
$Si_{87}H_{76}$	30	1427	43095	468.48	699.90	7.60×10^{-09}	3.29×10^{-10}
$Ga_{41}As_{41}H_{72}$	202	2334	471669	8179.51	9190.46	4.20×10^{-12}	4.33×10^{-13}

RATIONAL FILTERS

Why use rational filters?

- Consider a spectrum like this one:



- Polynomial filtering utterly ineffective for this case
- Second issue: situation when Matrix-vector products are expensive
- Generalized eigenvalue problems.

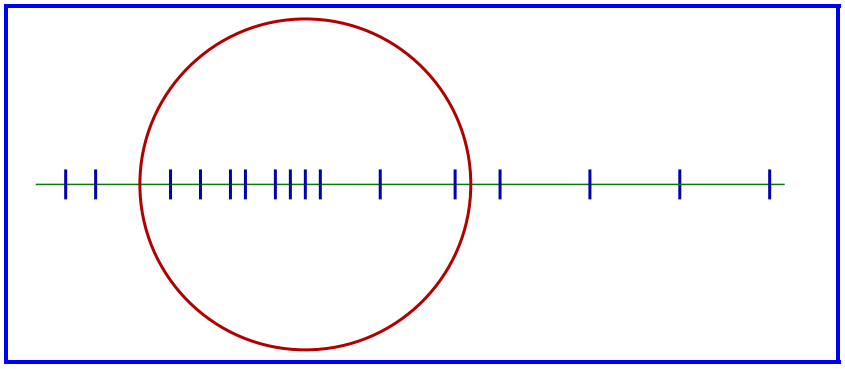
➤ Alternative is to use rational filters:

$$\phi(z) = \sum_j \frac{\alpha_j}{z - \sigma_j}$$

$$\phi(A) = \sum_j \alpha_j (A - \sigma_j I)^{-1} \quad \rightarrow$$

We now need to solve linear systems

➤ Tool: Cauchy integral representations of spectral projectors



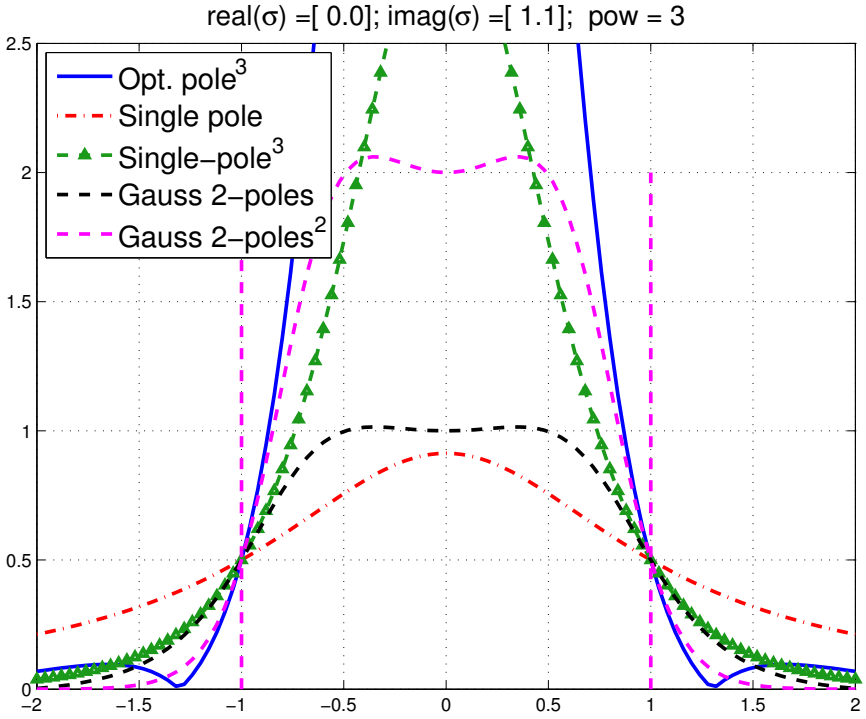
$$P = \frac{-1}{2i\pi} \int_\Gamma (A - sI)^{-1} ds$$

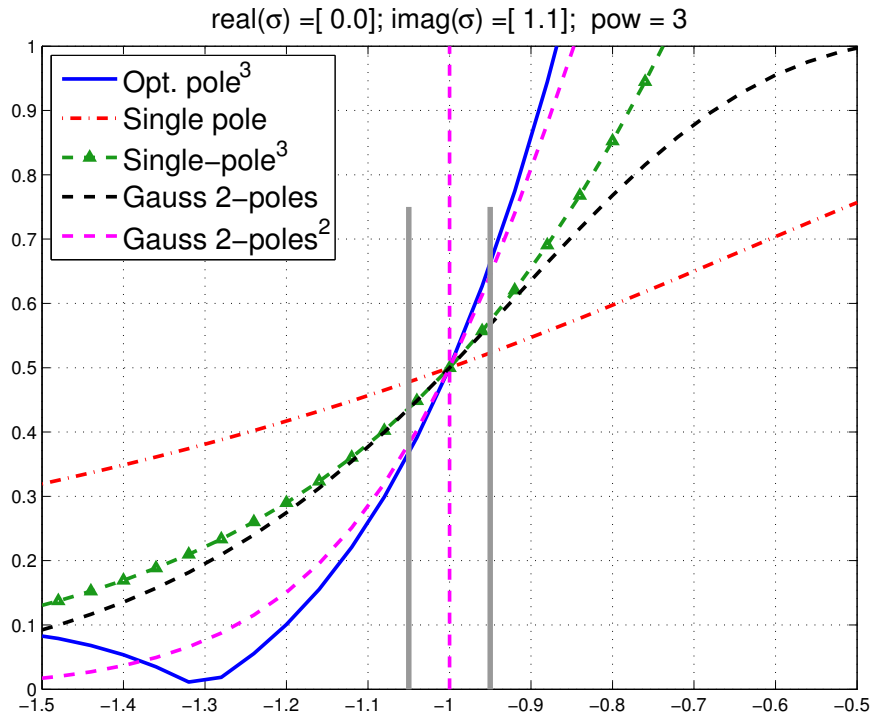
- Numer. integr. $P \rightarrow \tilde{P}$
- Use Krylov or S.I. on $\tilde{P}$

➤ Sakurai-Sugiura approach [Krylov]

➤ Polizzi [FEAST, Subsp. Iter.]

What makes a good filter





➤ Assume subspace iteration is used with above filters. Which filter will give better convergence?

➤ Simplest and best indicator of performance of a filter is the magnitude of its derivative at -1 (or 1)

The Gauss viewpoint: Least-squares rational filters

- Given: poles $\sigma_1, \sigma_2, \cdots, \sigma_p$
- Related basis functions $\phi_j(z) = \frac{1}{z-\sigma_j}$

Find $\phi(z) = \sum_{j=1}^{p} \alpha_j \phi_j(z)$ that minimizes

$$\int_{-\infty}^{\infty} w(t)|h(t) - \phi(t)|^2 dt$$

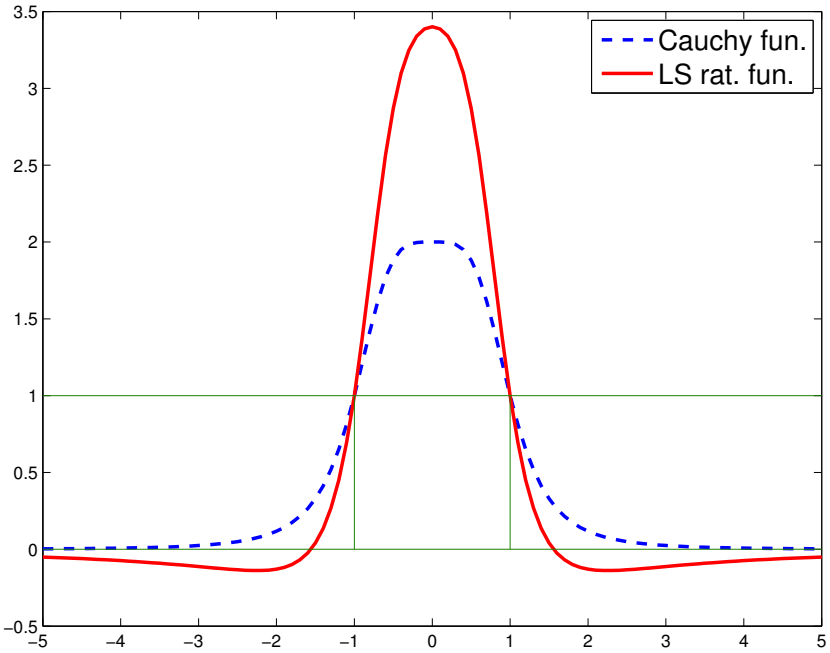
- $h(t)$ = step function $\chi_{[-1,1]}$.
- $w(t)$= weight function.

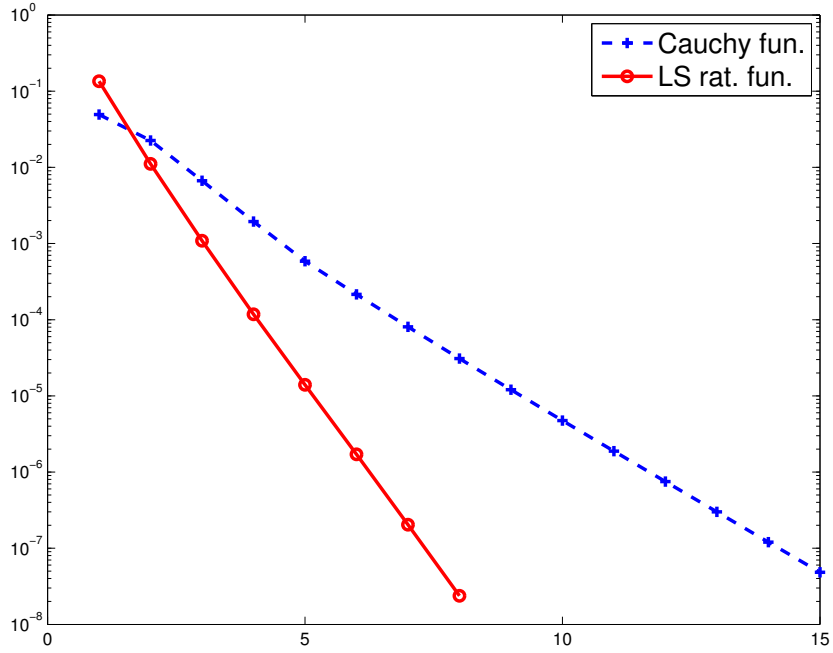
For example $a = 10$, $\beta = 0.2$

$$w(t) = \begin{cases} 0 & \text{if} \quad |t| > a \\ \beta & \text{if} \quad |t| \leq 1 \\ 1 & \text{else} \end{cases}$$

How does this work?

- Small example : Laplacean on a 43×53 grid. ($n = 2279$)
- 4 poles obtained from mid-point rule
- Want: all ($nev = 31$) eigenvalues in $[0,\ 0.2]$
- Use 1) standard subspace iteration + Cauchy (FEAST) then 2) subspace iteration + LS Rat. Appox.





- LS Uses the same poles + same factorizations as Cauchy but
- ... much faster as expected from a look at the curves of the functions

- Other advantages:

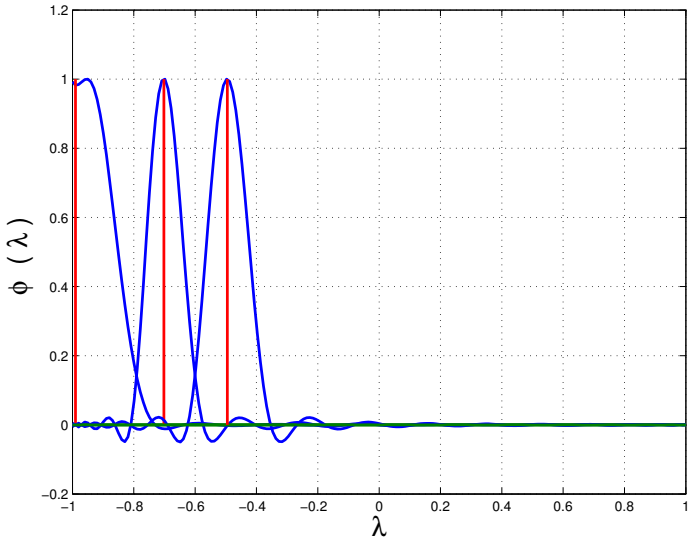
- Can select poles far away from real axis $\rightarrow$ faster iterative solvers
- Very flexible – can be adapted to many situations
- Can repeat poles (!)

- Implemented in EVSL.. [Interfaced to UMFPACK as a solver]

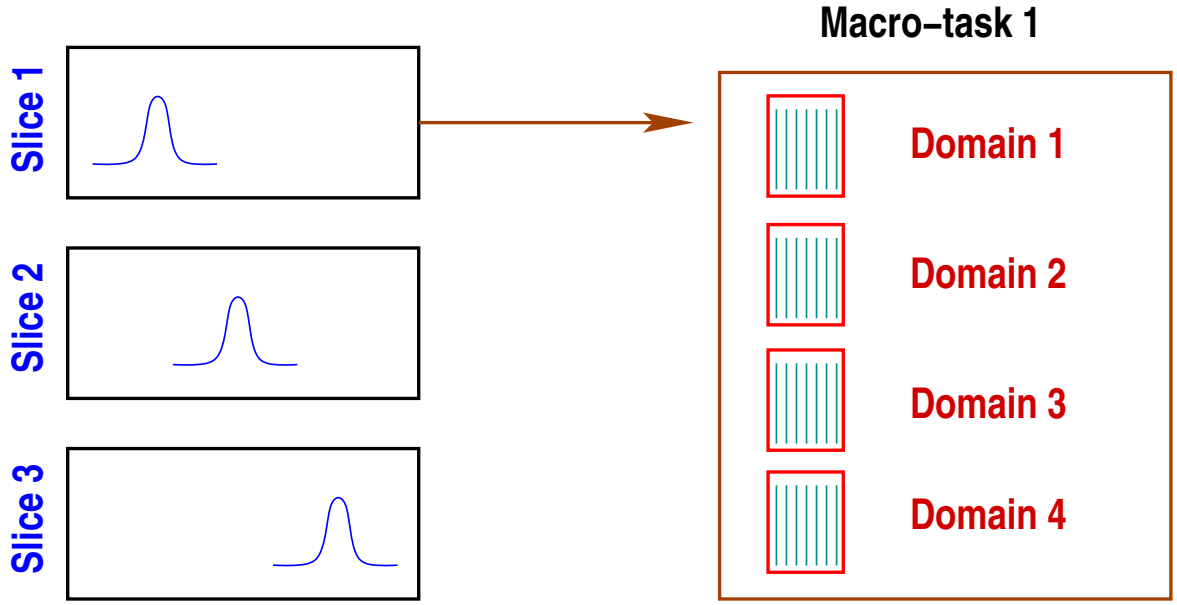
Spectrum Slicing and the EVSL project

- EVSL package now at version 1.1.x
- Uses polynomial and rational filtering: Each can be appealing in different situations.

Spectrum slicing: Invokes Kernel Polynomial Method or Lanczos quadrature to cut the overall interval containing the spectrum into small sub-intervals.



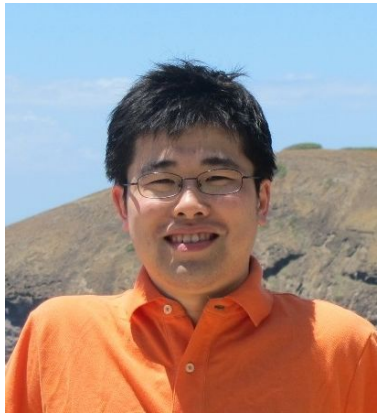
Levels of parallelism



The two main levels of parallelism in EVSL

EVSL Main Contributors (version 1.1.0+) & Support

- Ruipeng Li
LLNL



- Yuanzhe Xi
Asst. Prof. Emory

- Luke Erlandson
PhD Student, GTech.

➤ Work supported by NSF (past work: DOE)

➤ See web-site for details:

`http://www-users.cs.umn.edu/~saad/software/EVSL/`

EVSL: current status & plans

Version _1.0 Released in Sept. 2016

- Matrices in CSR format (only)
- Standard Hermitian problems (no generalized)
- Spectrum slicing with KPM (Kernel Polynomial Meth.)
- Trivial parallelism across slices with OpenMP
- Methods:
  - Non-restart Lanczos – polynomial & rational filters
  - Thick-Restart Lanczos – polynomial & rational filters
  - Subspace iteration – polynomial & rational filters

Version _1.1.x V_1.1.0 Released back in August 2017.

- general `matvec` [passed as function pointer]
- $Ax = \lambda Bx$
- Fortran (03) interface.
- Spectrum slicing by Lanczos and KPM
- Efficient Spectrum slicing for $Ax = \lambda Bx$ (no solves with B).

Version _1.2.x pEVSL – In progress

- Fully parallel version [MPI + openMP]

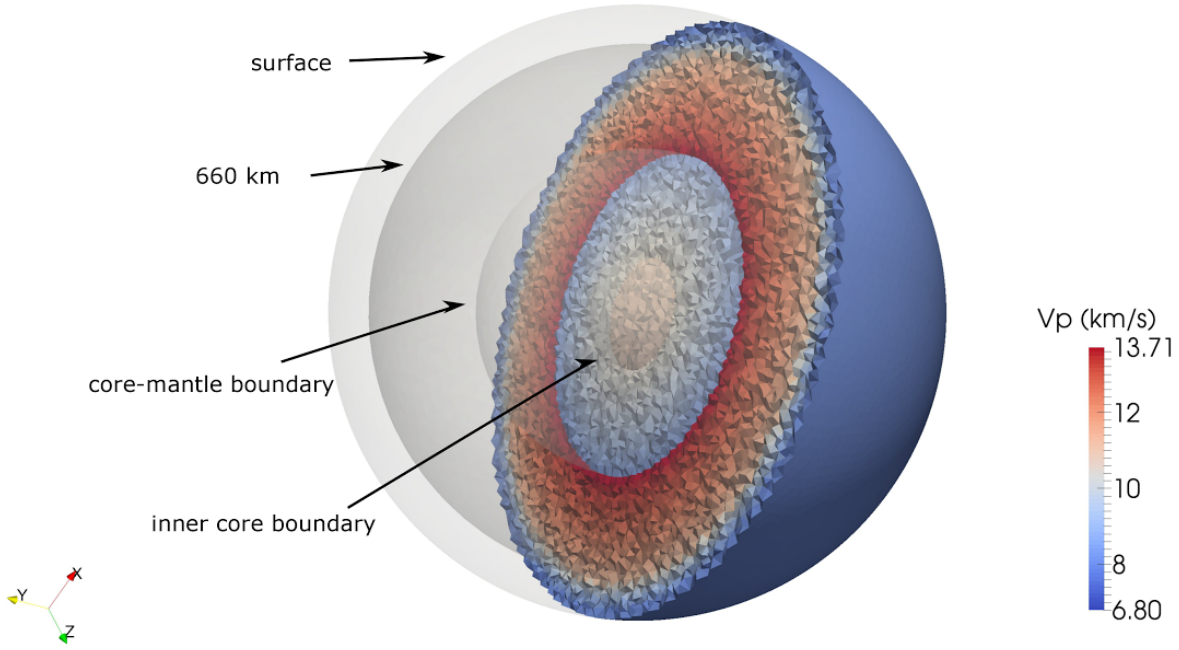
Spectrum slicing and the EVSL package

- All eigenvalues in $[0,\ 1]$ of of a 49^3 discretized Laplacian
- eigs(A,1971,'sa'): 14830.66 sec
- Solution: Use DOS to partition $[0,\ 1]$ into 5 slices
- Polynomial filtering from EVSL on Mesabi MSI, 23 threads/slice

$[a_i, a_{i+1}]$	# eigs	CPU time (sec)			max residual
		matvec	orth.	total	
$[0.00000, 0.37688]$	386	1.31	18.26	28.66	2.5×10^{-14}
$[0.37688, 0.57428]$	401	3.28	38.25	56.75	8.7×10^{-13}
$[0.57428, 0.73422]$	399	4.69	36.47	56.73	1.7×10^{-12}
$[0.73422, 0.87389]$	400	5.97	38.60	61.40	6.6×10^{-12}
$[0.87389, 1.00000]$	385	6.84	36.16	59.45	4.3×10^{-12}

➤ Grand tot. = 263 s. Time for slicing the spectrum: 1.22 sec.

Computing the Earth normal modes



- Collaborative effort: Rice-UMN:
  J. Shi, R. Li, Y. Xi, YS, and M. V. De Hoop
- FEM model leads to a generalized eigenvalue problem:

$$\begin{bmatrix} A_s & & E_{fs} \\ & 0 & A_d \\ E_{fs}^T & A_d^T & A_p \end{bmatrix} \begin{bmatrix} u^s \\ u^f \\ p^e \end{bmatrix} = \omega^2 \begin{bmatrix} M_s & & \\ & M_f & \\ & & 0 \end{bmatrix} \begin{bmatrix} u^s \\ u^f \\ p^e \end{bmatrix}$$

- Want all eigen-values/vectors inside a given interval
- Issue 1: 'mass' matrix has a large null space..
- Issue 2: interior eigenvalue problem
- Solution for 1: change formulation of matrix problem [eliminate p^e ...]

➤ New formulation :

$$\underbrace{\left\{\begin{pmatrix} A_s & 0 \\ 0 & 0 \end{pmatrix} - \begin{pmatrix} E_{fs} \\ A_d \end{pmatrix} A_p^{-1} \begin{pmatrix} E_{fs}^T & A_d^T \end{pmatrix}\right\}}_{\widehat{A}} \begin{pmatrix} u^s \\ u^f \end{pmatrix} =$$

$$\omega^2 \underbrace{\begin{pmatrix} M_s & 0 \\ 0 & M_f \end{pmatrix}}_{\widehat{M}} \begin{pmatrix} u^s \\ u^f \end{pmatrix}$$

➤ Use polynomial filtering – need to solve with $\widehat{M}$ but ...

- ... severe scaling problems if direct solvers are used

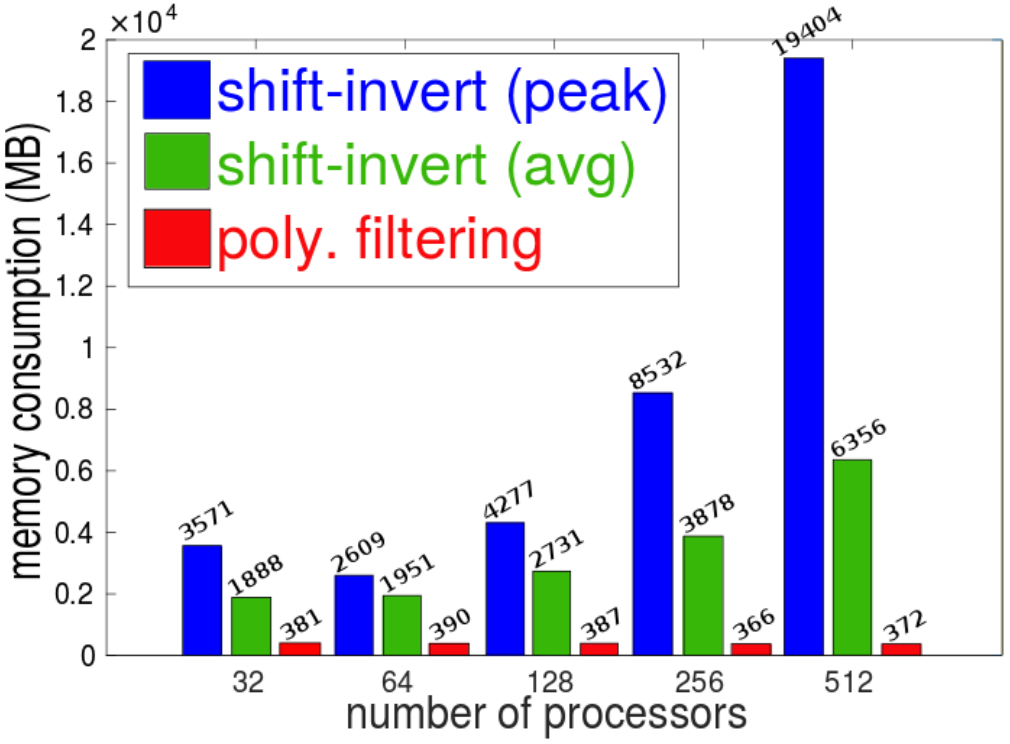
Hence:

➤ Replace action of M^{-1} by a low-deg. polynomial in M [to avoid direct solvers]

➤ Memory : parallel shift-invert and polynomial filtering

Machine: Comet, SDSC

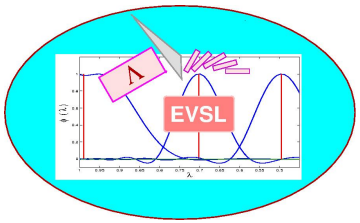
Matrix size	# Proc.s
591,303	32
1,157,131	64
2,425,349	128
4,778,004	256
9,037,671	512



Recent: weak calability test for different solid (Mars-like) models on TACC Stampede2

nn/np	Mat-size	Av (ms)	$\leftarrow$ Eff.	Mv (ms)	$\leftarrow$ Eff.	$M^{-1}v$ (μs)	$\leftarrow$ Eff.
2/96	1,038,084	1760	1.0	495	1.0	0.01044	1.0
4/192	2,060,190	1819	0.960	568	0.865	0.0119	0.870
8/384	3,894,783	1741	0.948	571	0.813	0.0119	0.825
16/768	7,954,392	1758	0.959	621	0.763	0.0129	0.774
32/1536	15,809,076	1660	1.009	572	0.824	0.0119	0.834
64/3072	31,138,518	1582	1.043	566	0.820	0.0117	0.837
128/6144	61,381,362	1435	1.133	546	0.838	0.0113	0.851
256/12288	120,336,519	1359	1.173	592	0.757	0.01221	0.774

S O F T W A R E



EVSL NEW! a library of (sequential) eigensolvers based on spectrum slicing. **Version 1.0** released on [09/11/2016]
EVSL provides routines for computing eigenvalues located in a given interval, and their associated eigenvectors, of real symmetric matrices. It also provides tools for spectrum slicing, i.e., the technique of subdividing a given interval into p smaller subintervals and computing the eigenvalues in each subinterval independently. EVSL implements a polynomial filtered Lanczos algorithm (thick restart, no restart) a rational filtered Lanczos algorithm (thick restart, no restart), and a polynomial filtered subspace iteration.

ITSOL a library of (sequential) iterative solvers. **Version 2** released. [11/16/2010]
ITSOL can be viewed as an extension of the ITSOL module in the SPARSKIT package. It is written in C and aims at providing additional preconditioners for solving general sparse linear systems of equations. Preconditioners so far in this package include (1) ILUK (ILU preconditioner with level of fill) (2) ILUT (ILU preconditioner with threshold) (3) ILUC (Crout version of ILUT) (4) VBILUK (variable block preconditioner with level of fill - with automatic block detection) (5) VBILUT (variable block preconditioner with threshold - with automatic block detection) (6) ARMS (Algebraic Recursive Multilevel Solvers -- includes actually several methods - In particular the standard ARMS and the ddPQ version which uses nonsymmetric permutations).
ZITSOL a complex version of some of the methods in ITSOL is also available.

Conclusion

- `EVSL` code available here: [Current version: version 1.1.1]

  `www.cs.umn.edu/~saad/software/EVSL`

- `EVSL` Also on github (development)

Plans: (1) Release fully parallel code; (2) Block versions; (3) Iterative solvers for rational filt.; (4) Nonhermitian case;

- Earth modes calculations done with fully parallel code
- Scalability issues with parallel direct solvers ...
- ... Needed: iterative solvers for the highly indefinite case
- Frontier in eigenvalue problem: *Nonlinear* case