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

Purdue University
Oct. 7th, 2016

Collaborators:

Past work:

- Haw-ren Fang [former post-doc]
- Grady Schoefield and Jim Chelikowsky [UT Austin] - Windowing into PARSEC

New group effort:

- Ruipeng Li [now at LLNL]
- Eugene Vecharynski [Lawrence Berkeley Lab]
- Chao Yang [Lawrence Berkeley Lab]
- Yuanzhe Xi [Post-doc, Univ. of Minnesota]

➤ Work supported by DOE : *Scalable Computational Tools for Discovery and Design: Excited State Phenomena in Energy Materials* [Institutions: UT Austin, UC Berkeley, U Minn]

➤ And by NSF: *Advanced algorithms and high-performance software for large scale eigenvalue problems* [with E. Polizzi, U. Mass Amherst]

Introduction & Motivation

➤ Some applications require the computation of a large number of eigenvalues and vectors of very large matrices. These are found mostly in quantum physics/ chemistry.

➤ Density Functional Theory in electronic structure calculations: *'ground states'*

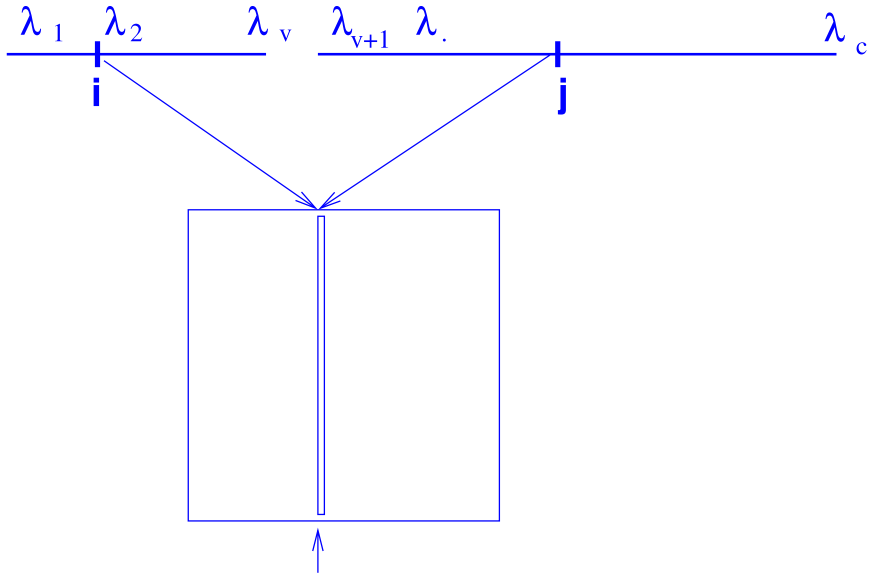
➤ *Excited states* involve transitions and invariably lead to much more complex computations

Illustration:

In Time-Dependent Density Functional Theory (TDDFT), the so-called Cassida approach computes eigenvalues of a matrix K built using both occupied and unocuppied states.

- ➤ Each pair → one column of K
- ➤ To compute each column need to solve a Poisson eqn. on domain
- ➤ Intense calculation, lots of parallelism

Occupied States Empty States



One column of K

One Difficulty: (among others): need to calculate unoccupied states as well as occupied states.

- ➤ Similar types of calculations in the GW approach [see, e.g., BerkeleyGW] – But more complex

- ➤ Challenge: ‘Hamiltonian of size $n \sim 10^6$ get 10% of bands’

Soving large eigenvalue problems: Current state of the art

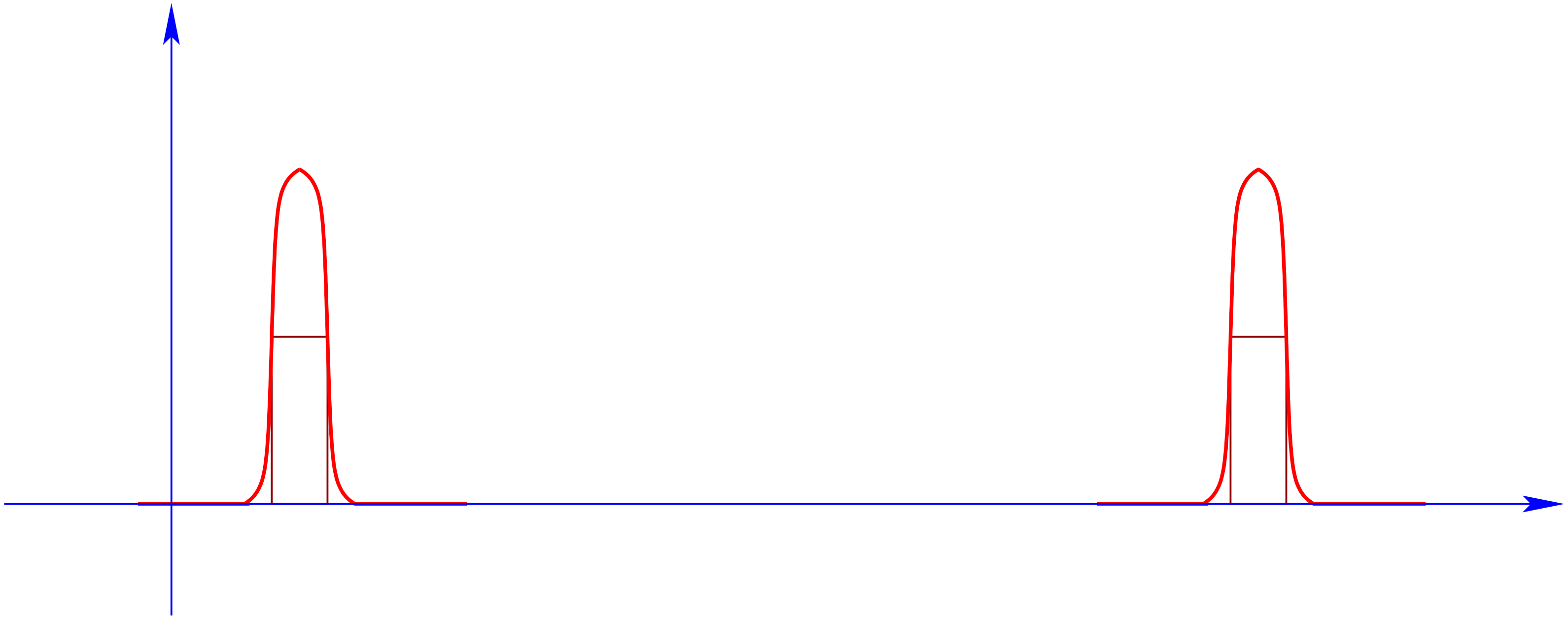
- Eigenvalues at one end of the spectrum:
  - Subspace iteration + filtering [e.g. FEAST, Cheb,...]
  - Lanczos+variants (no restart, thick restart, implicit restart, Davidson,..)
  - Block Algorithms [Block Lanczos, TraceMin, LOBPCG, ...]
  - + Many others - more or less related to above
- 'Interior' eigenvalue problems (middle of spectrum):
  - Combine shift-and-invert + Lanczos/block Lanczos. Used in, e.g., NASTRAN

Issues with shift-and invert (and related approaches)

- Issue 1: factorization not always feasible
  - Can use iterative methods?
- Issue 2: Iterative techniques often fail –
  - Reason: Highly indefinite problems.

Spectrum slicing for computing many eigenvalues

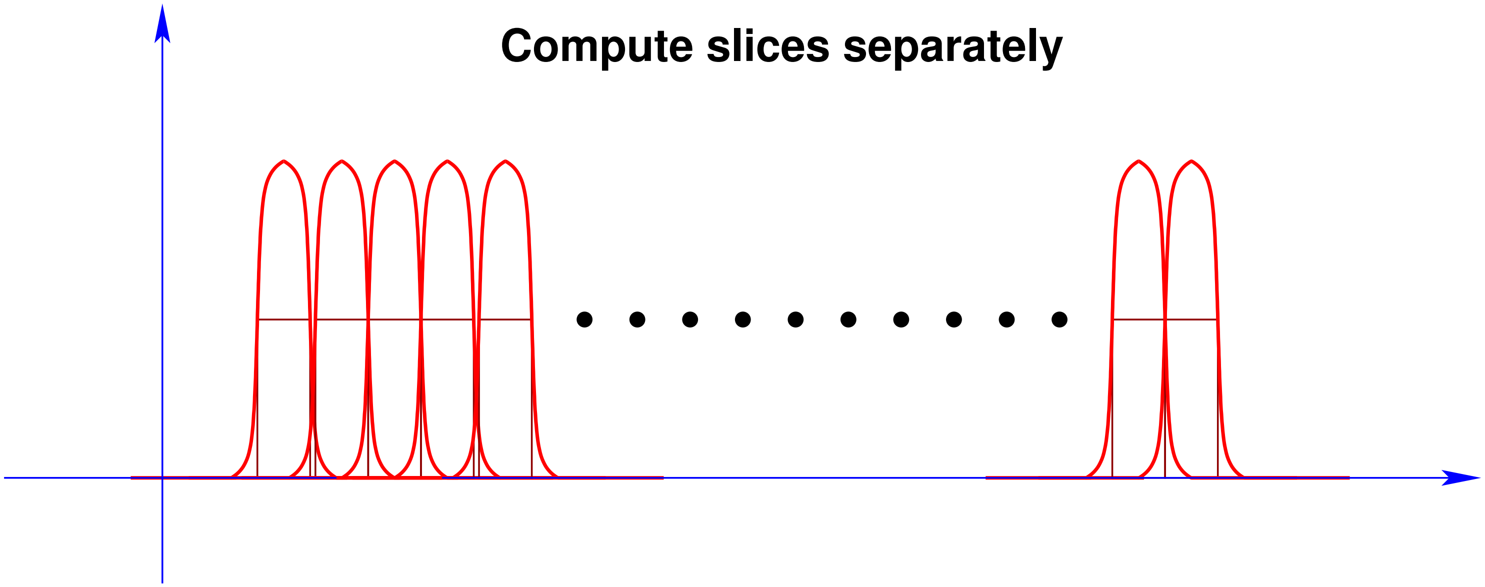
Rationale: Eigenvectors on both ends of wanted spectrum need not be orthogonalized against each other :



- ➤ Idea: Get the spectrum by ‘slices’ or ’windows’ [e.g., a few hundreds or thousands of pairs at a time]
- ➤ Can use polynomial or rational filters

- In an approach of this type the filter is the key ingredient.

Goal: Compute each slice independently from the others.



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

- Only need a good estimate of window size

Computing a slice of the spectrum

Q: How to compute eigenvalues in the middle of the spectrum of a large Hermitian matrix?

A: Common practice: Shift and invert + some projection process (Lanczos, subspace iteration..)

- ➤ Requires factorizations of $A - \sigma I$ for a sequence of σ's
- ➤ Out of the question for some (e.g. large 3D) problems.
- ➤ First Alternative: Polynomial filtering

Polynomial filtering

- Apply Lanczos or Subspace iteration to: $\boxed{M = \phi(A)}$ 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

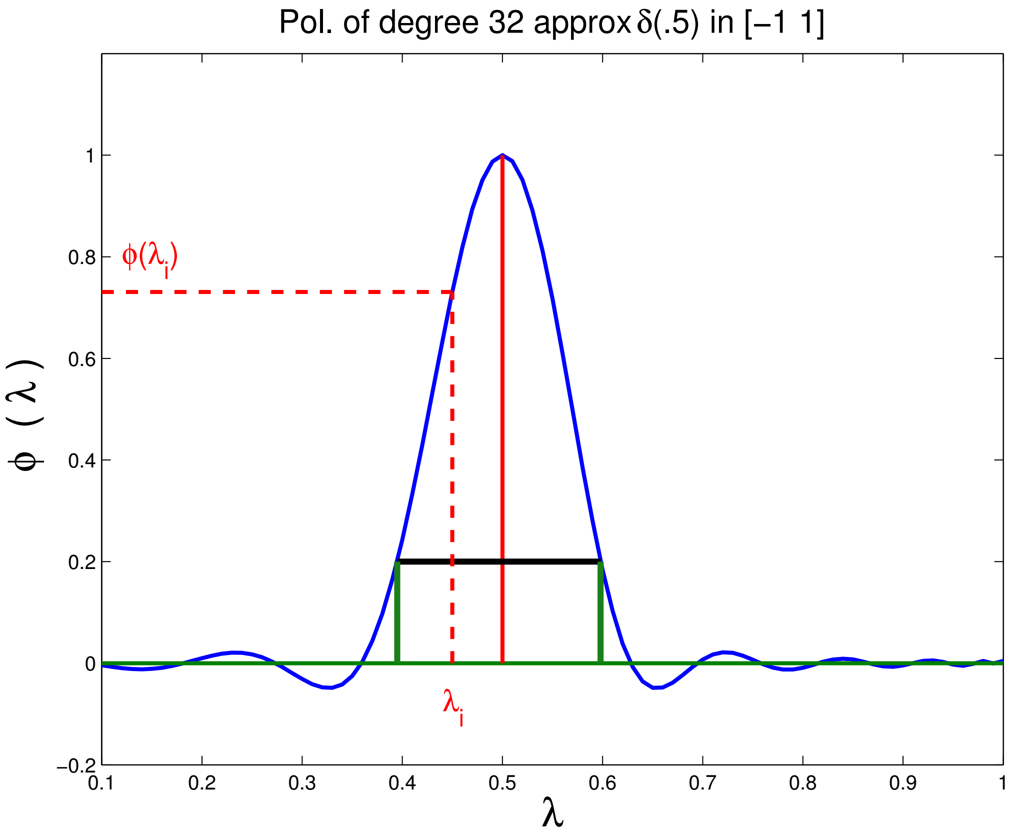
Key ingredients:

- Selecting Polynomials

- Locking/deflation or other strategies in order not to miss eigenvalues.

What polynomials?

- For end-intervals can just use Chebyshev
- For inside intervals: several choices



- Recall the main goal:

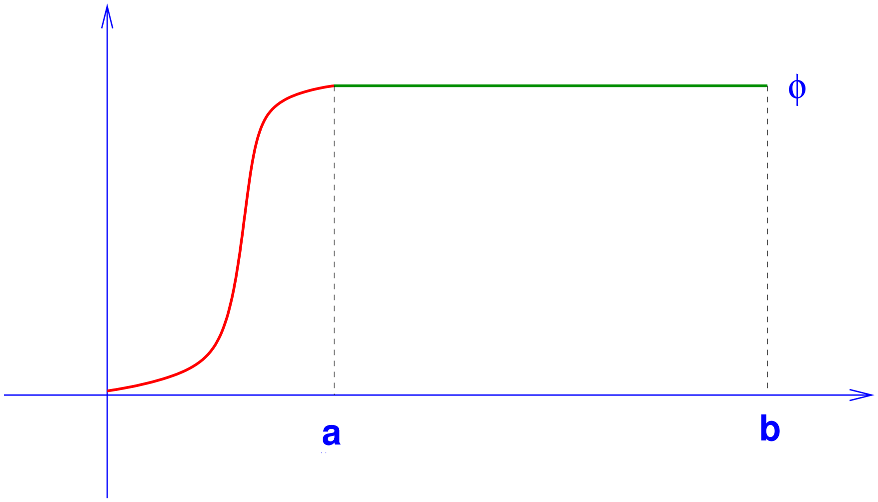
A polynomial that has large values for $\boldsymbol{\lambda} \in [\boldsymbol{a}, \boldsymbol{b}]$ *small values elsewhere*

Past work: Two-stage approach

➤ Two stage approach used in filtlan [H-r Fang, YS 2011] -

➤ First select a "base filter"

➤ e.g., a piecewise polynomial function [a spline]

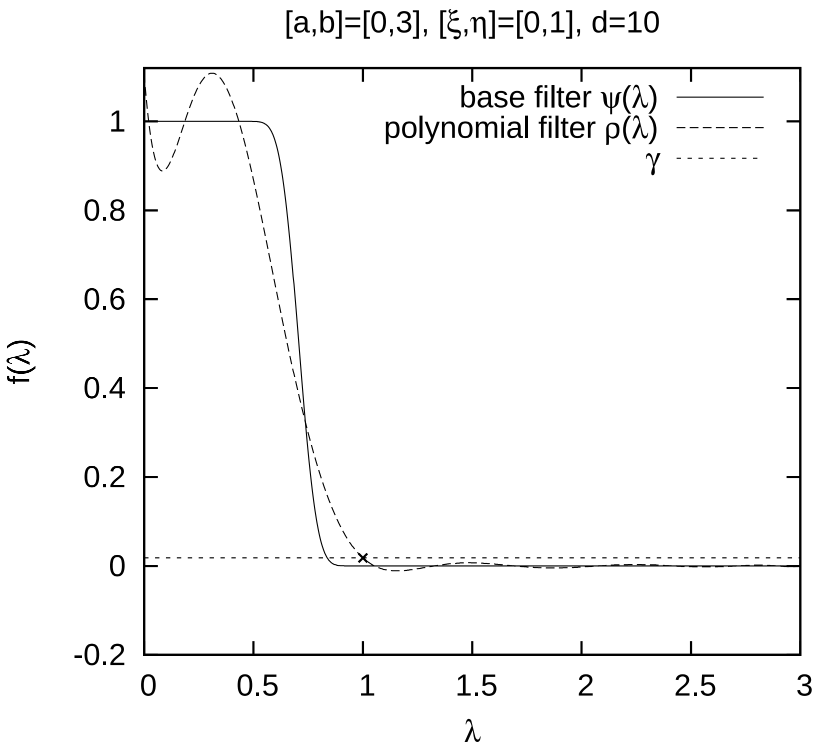


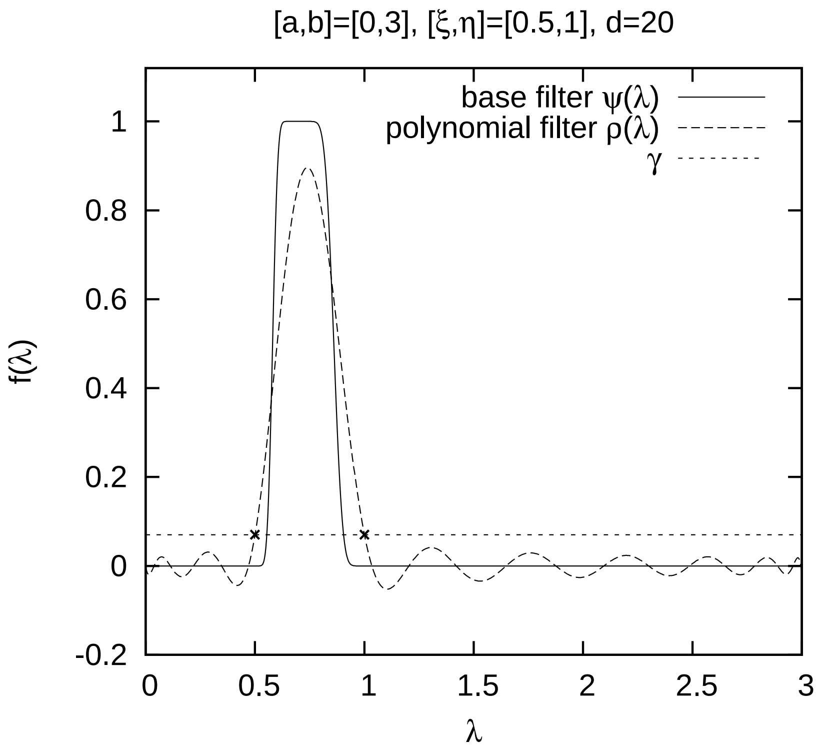
- Then approximate base filter by degree k polynomial in a least-squares sense.

- No numerical integration needed

Main advantage: Extremely flexible.

Low-pass, high-pass, & barrier (mid-pass) filters





- See Reference on Lanczos + pol. filtering: Bekas, Kokiopoulou, YS (2008) for motivation, etc.

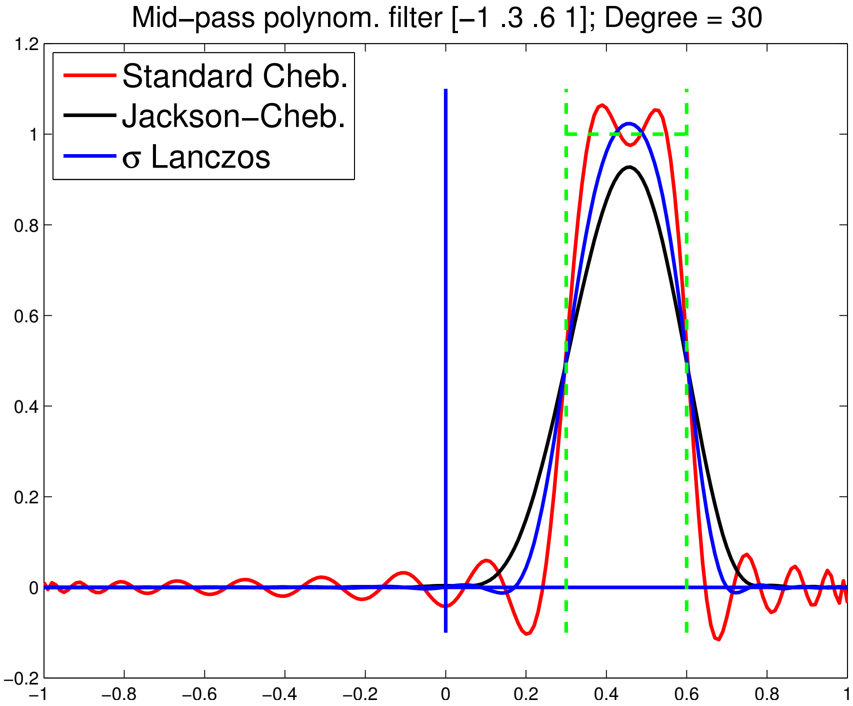
- H.-r Fang and YS “Filtlan” paper [SISC,2012] and code

Simpler: Step-function Chebyshev + Jackson damping

- Seek the best LS approximation to step function.

$$f(x) \approx \sum_{i=0}^{k} g_i^k \gamma_i T_i(x)$$

- Add 'Damping coefficients' to reduce/eliminate Gibbs oscillations

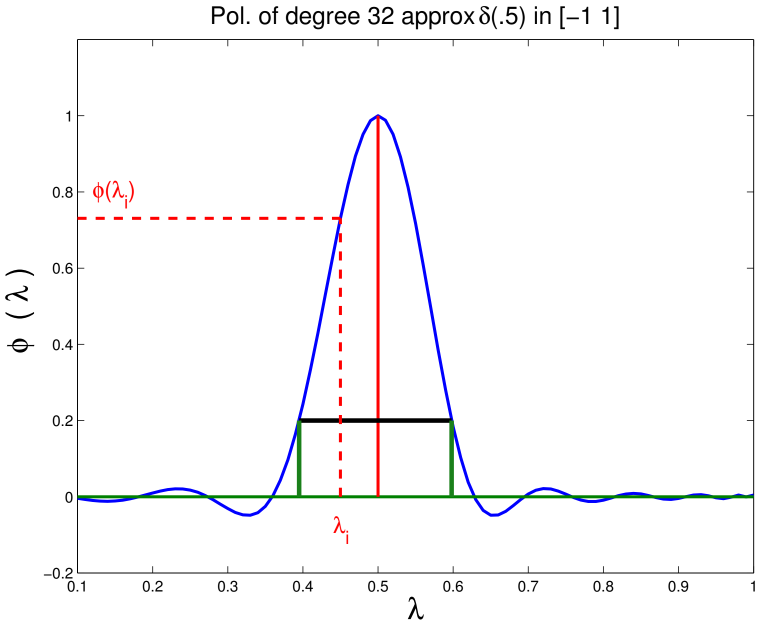

Mid−pass polynom. filter [−1 .3 .6 1]; Degree = 30

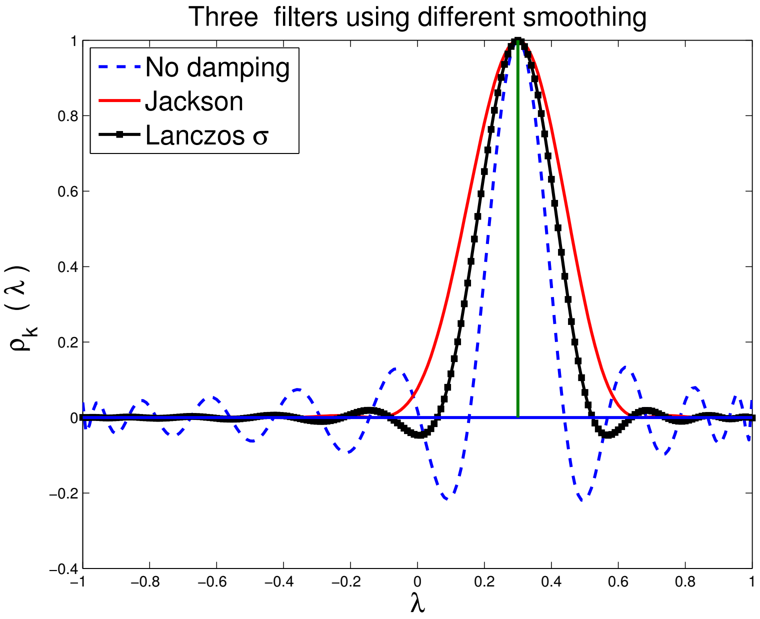
- G. Schofield, J. R. Chelikowsky and YS, CPC, 183, ('11)

Question: Why approximate the 'step function'?

Even Simpler: δ-Dirac function

➤ Obtain the LS approximation to the $\delta-$ 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 $\hat{\rho}_k(t) = \rho_k(t)/\rho_k(\gamma)$.

Theorem Assuming $k \geq 1$, the following equalities hold:

$$\int_{-1}^{1} \frac{[\hat{\rho}_k(s)]^2}{\sqrt{1-s^2}} ds = \frac{1}{\sum_{j=0}^{k}[\hat{T}_j(\gamma)]^2}$$
$$= \frac{2\pi}{(2k+1)} \times \frac{1}{1+\frac{\sin(2k+1)\theta_\gamma}{(2k+1)\sin\theta_\gamma}},$$

where $\theta_\gamma = \cos^{-1}\gamma$.

‘The soul of a new filter’ – A few details

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

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

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

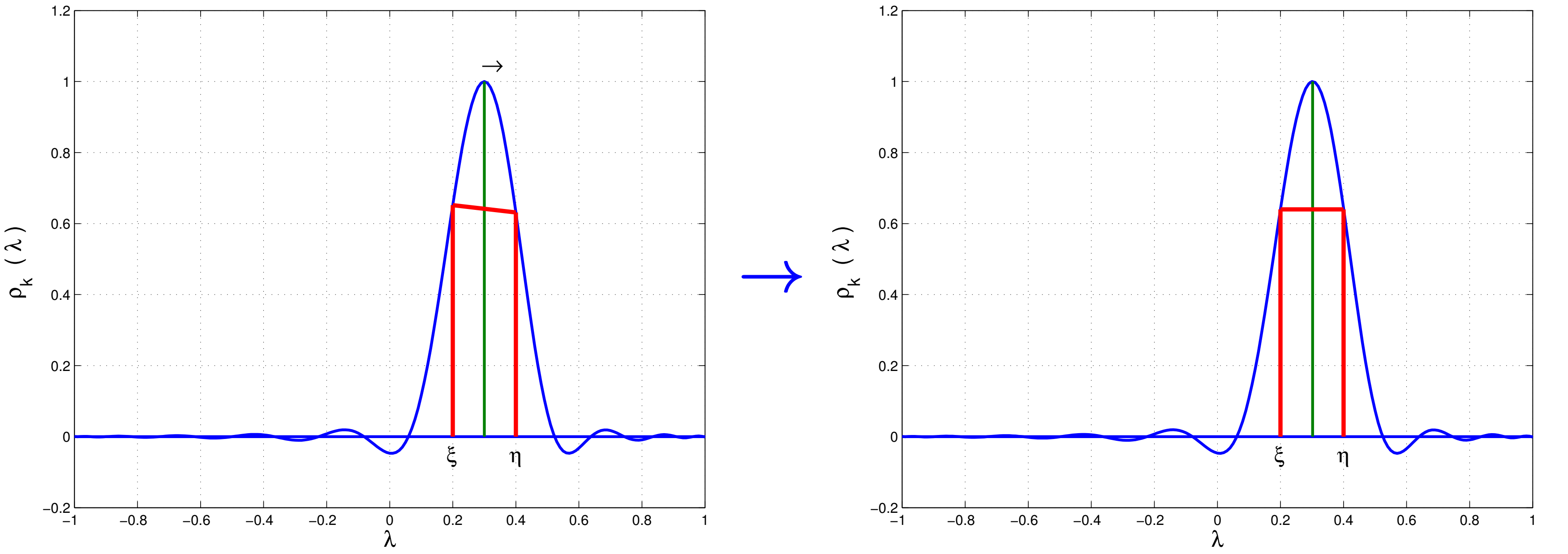
Problem:

Given interval $[\xi, \eta]$ find $p_m(t)$ of degree m, such that (1) convergence with e.g. subspace iteration or Lanczos is fast enough, (2) wanted eigenvalues are easy to identify and extract, and (3) unwanted ones are never an issue.

Issue # one: ‘balance the filter’

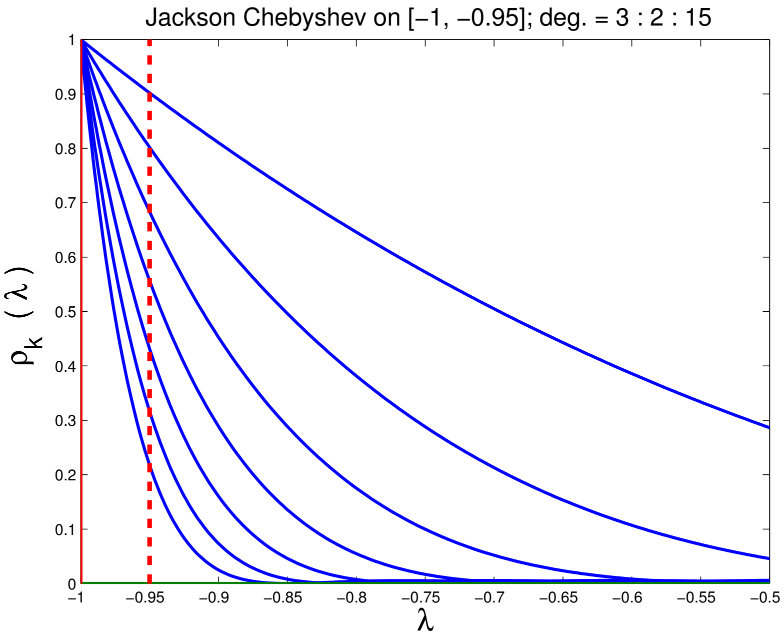
- To facilitate the selection of *‘wanted’* eigenvalues [Select $\boldsymbol{\lambda}$’s such that $\phi(\boldsymbol{\lambda}) >$ bar] we need to ...

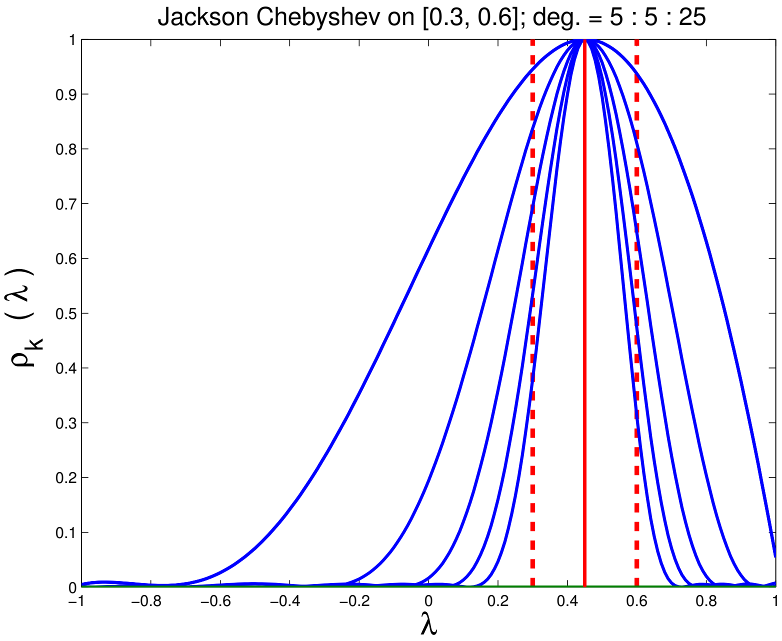
- ... find γ so that $\phi(\boldsymbol{\xi}) == \phi(\boldsymbol{\eta})$



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

Issue # two: Determine degree (automatically)



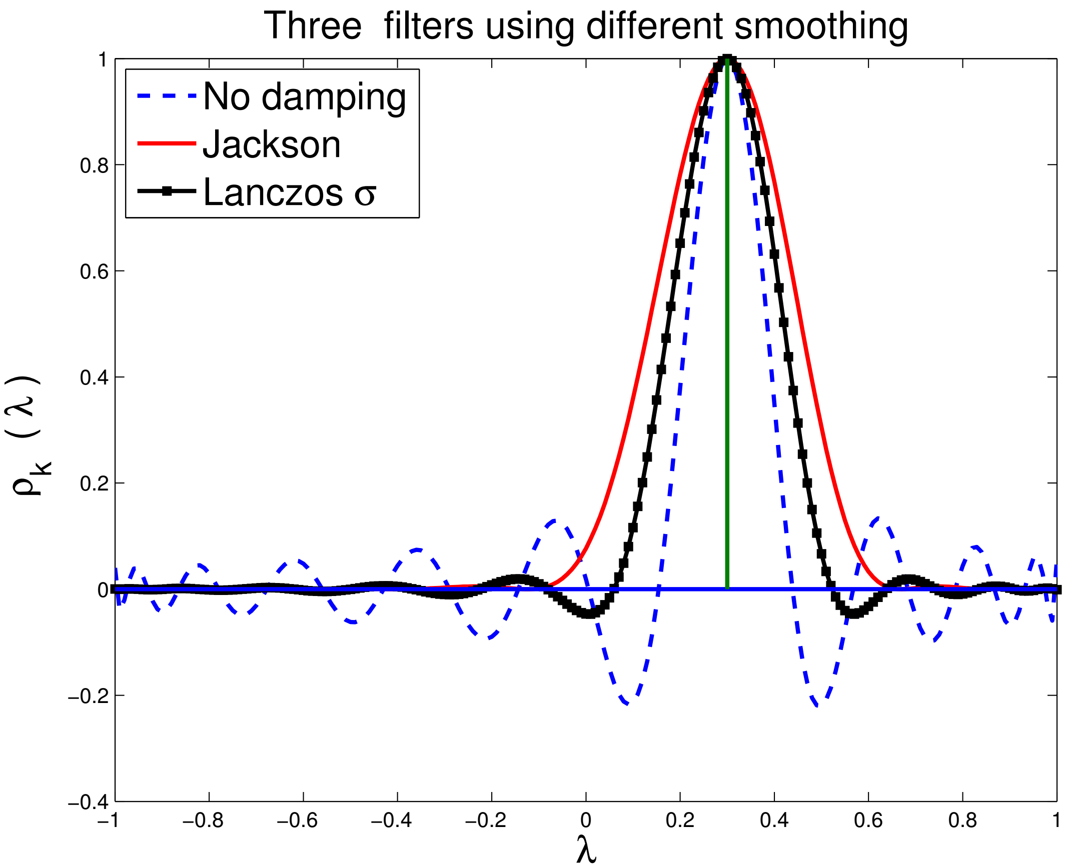


- 1) Start low (e.g. 2); 2) Increase degree until value (s) at the boundary (ies) become small enough –

- Eventually w'll use criterion based on derivatives at ξ & η

Issue # Three : Gibbs oscillations

- Discontinuous 'function' approximated → Gibbs oscillations



- Three options:

  - No damping
  - Jackson damping
  - Lanczos σ damping

- Good compromise: Lanczos σ damping

Extraction: Lanczos vs. Subspace iteration

- Subspace iteration is quite appealing in a electronic structure calculations – Can re-use previous subspace.

- Lanczos without restarts

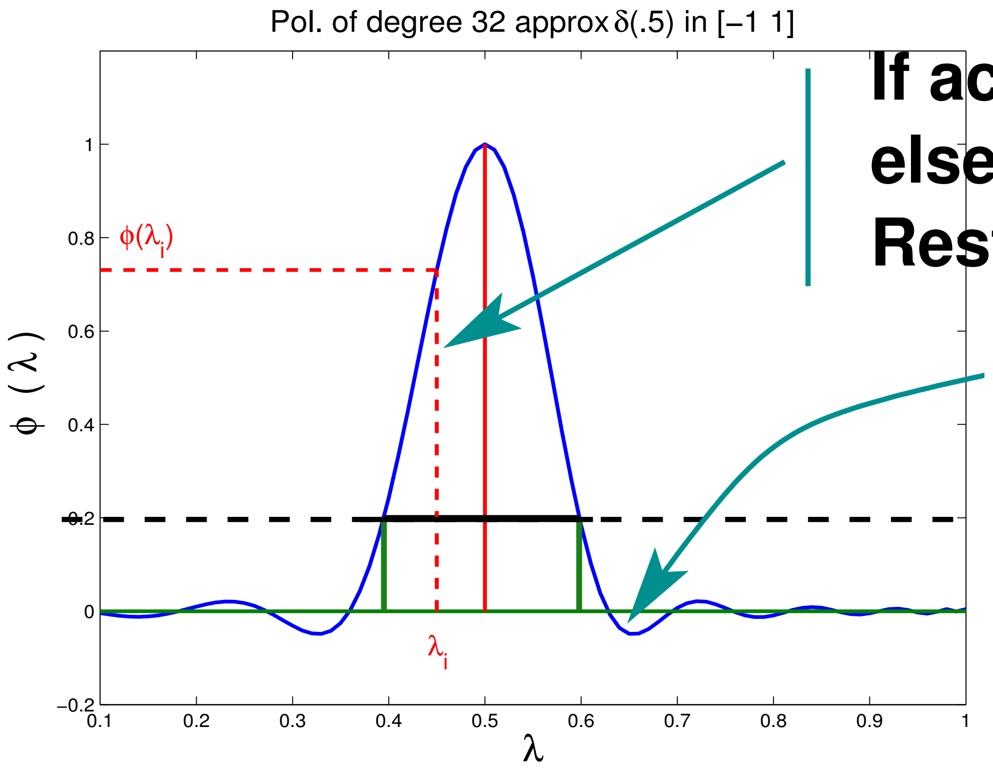
- Lanczos with Thick-Restarting [TR Lanczos, Stathopoulos et al '98, Wu & Simon'00]

- Crucial tool in TR Lanczos: deflation ('Locking')

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

- If filter is good: Can catch all eigenvalues in interval thanks to deflation + Lanczos.

- PolFilt Thick-Restart Lanczos in a picture:



If accurate then lock else add to Thick Restart set.

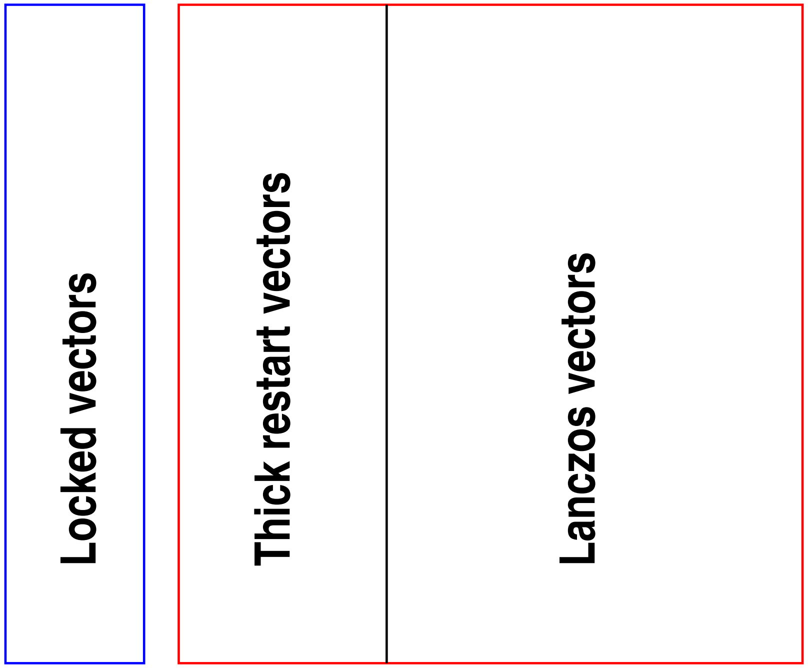
Reject

- Due to locking, no more candidates will show up in wanted area after some point $\longrightarrow$ Stop.

- Similar procedure possible with subspace iteration.

TR Lanczos: The 3 types of basis vectors

Basis vectors

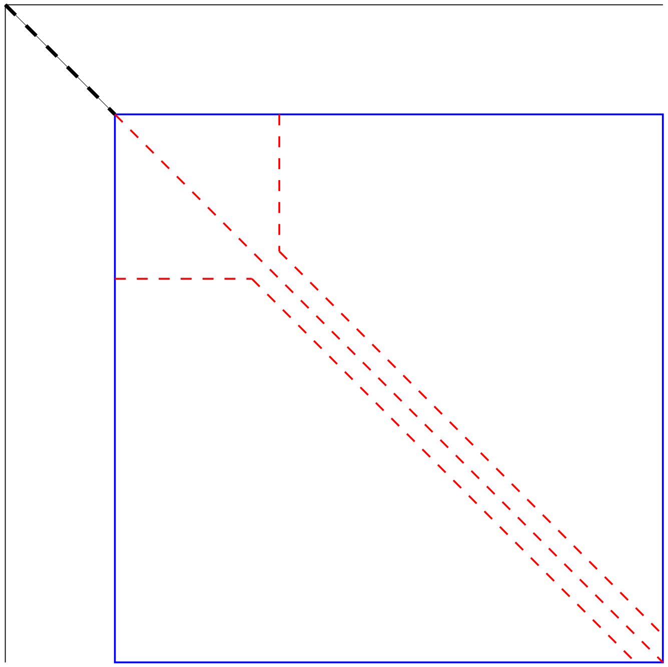


Locked vectors

Thick restart vectors

Lanczos vectors

Matrix representation



How do I slice a spectrum?

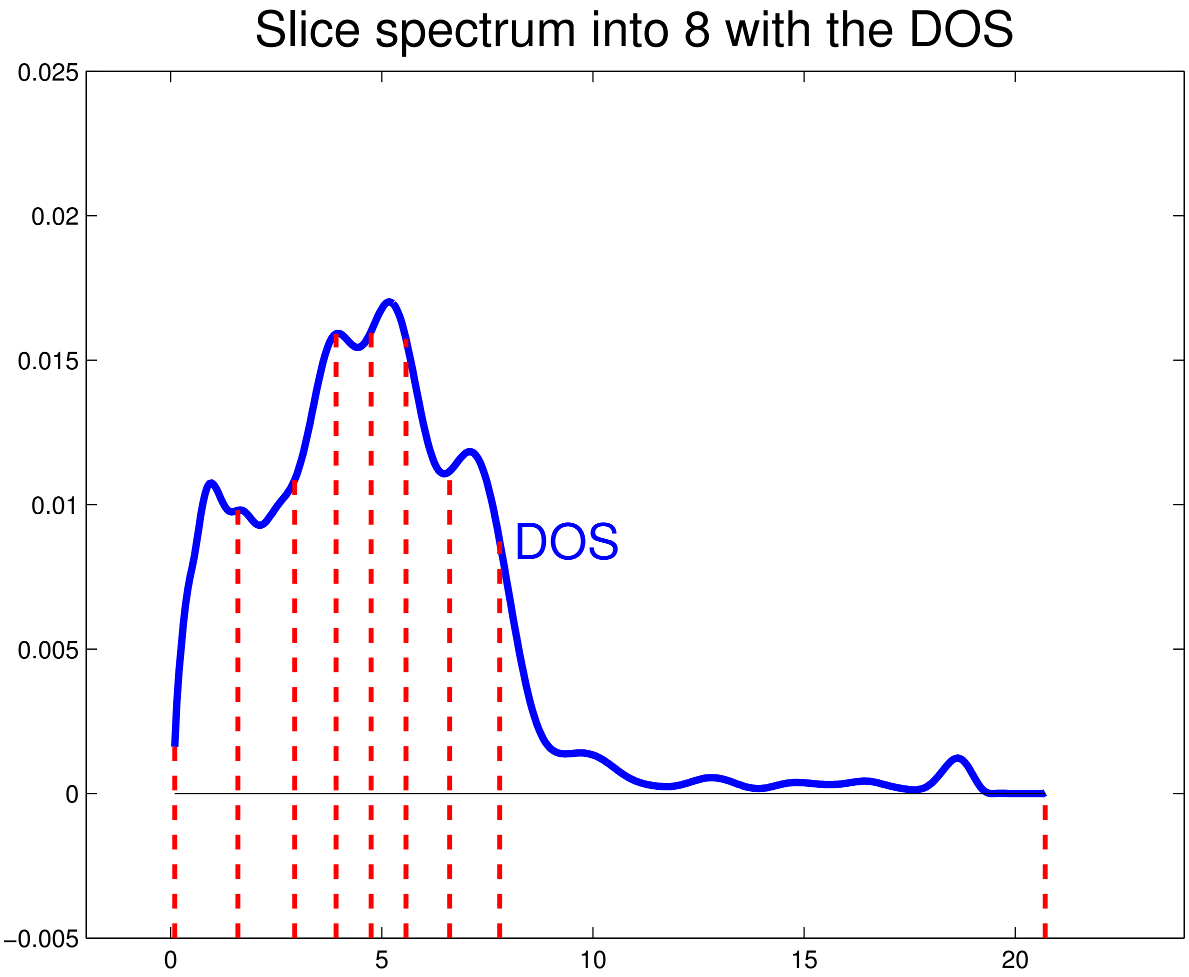
Analogue question:

How would I slice an onion if I want each slice to have about the same mass?

- A good tool: Density of States – see:
  - L. Lin, YS, Chao Yang recent paper.
  - 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:

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

An example

- Implemented in C: First version of (sequential) EVSL.

- Polynom. Filt. Lanczos Thick-Restart with deflation + spectrum slicing,

- Also: Subspace iteration, non-restarted Lanczos.

- A test example from the PARSEC collection.

- Matrix Ge99H100 $[\boldsymbol{n = 112,985, nnz = 7,892,195}]$

Matrix	$[\boldsymbol{a, b}]$	$[\boldsymbol{\eta, \xi}]$	#eig
$\mathbf{Ge_{99}H_{100}}$	$[\mathbf{-1.2264, 32.7031}]$	$[\mathbf{-0.65, -0.0096}]$	250

- Asked to compute eigenvalues/vectors in 6 slices.

- DOS + integration $\longrightarrow$ (estimated) $\mathbf{242}$ eigenvalues in $[\xi,\ \eta]$ – roughly 40/interval [actual $\mathbf{250/6 \approx 41.66}$]

Slice #	Width	actual # e.v.	Pol. Deg.	# Matvecs
1	0.0869	38	169	50738
2	0.0708	46	220	33046
3	0.0997	42	165	49542
4	0.2542	42	71	21342
5	0.0740	38	264	79238
6	0.0547	44	301	60244

- Computed all 250 eigenvalues -

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
- Coefficients γ_k lead to evaluating Tr $(T_k(A))$
- Use trace estimators [discovered independently] to get traces
- Next: A few details
- Assume change of variable done so eigenvalues lie in $[-1,\ 1]$.
- Include the weight function in the expansion so expand:

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

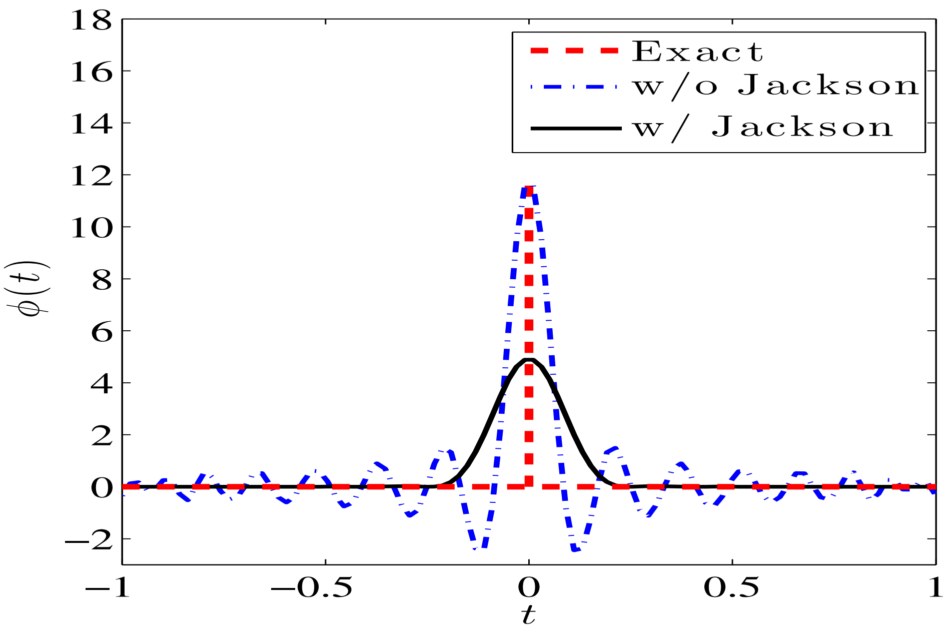
- Then, (full) expansion is: $\hat{\phi}(t) = \sum_{k=0}^{\infty} \mu_k T_k(t)$.
- Expansion coefficients μ_k are formally defined by:

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

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

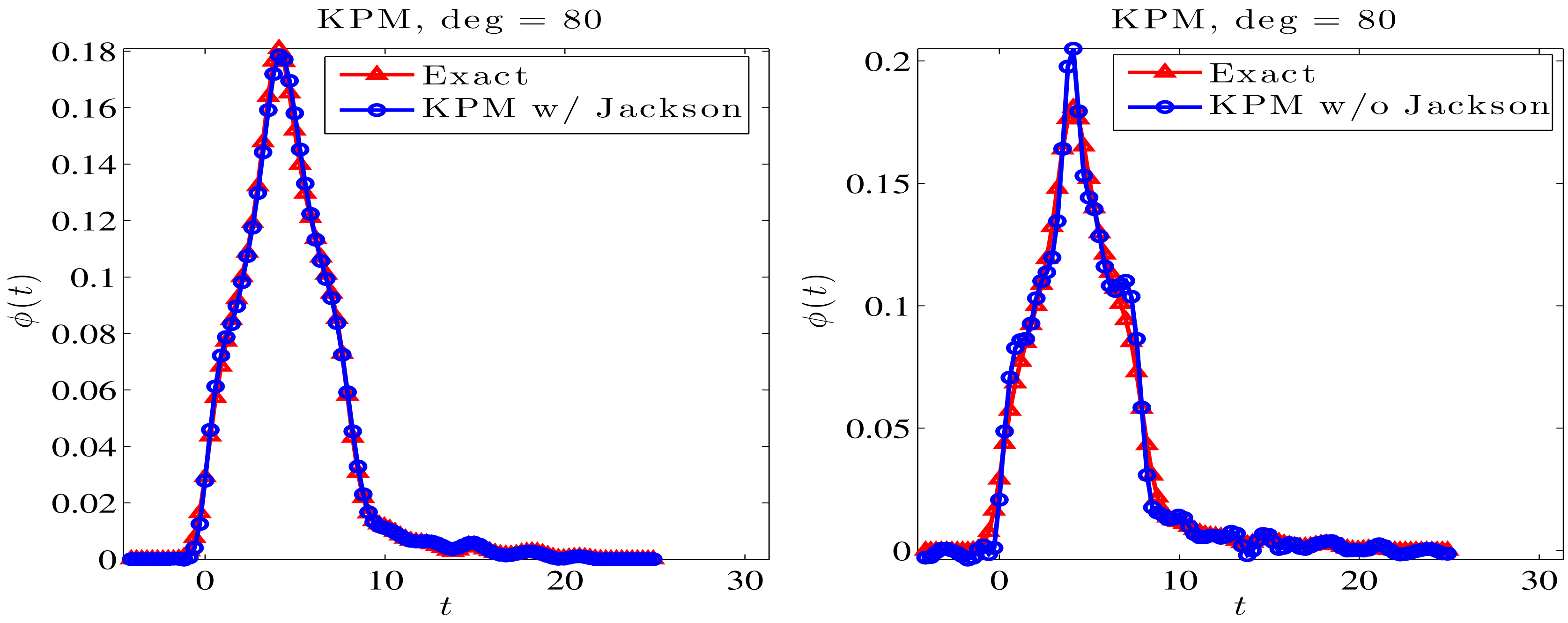
$$\text{Trace}(T_k(A)) \approx \frac{1}{n_{\text{vec}}}\sum_{l=1}^{n_{\text{vec}}}\left(v^{(l)}\right)^T T_k(A)v^{(l)}.$$

- 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



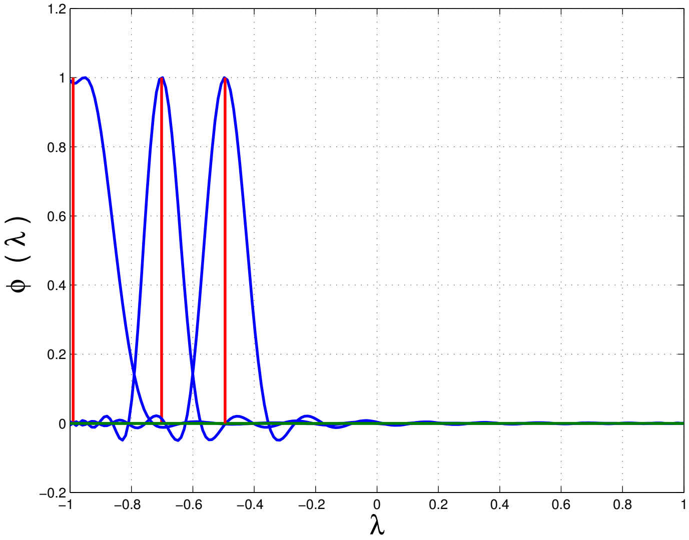
Left: Jackson damping; right: without Jackson damping.

Spectrum Slicing and the EVSL project

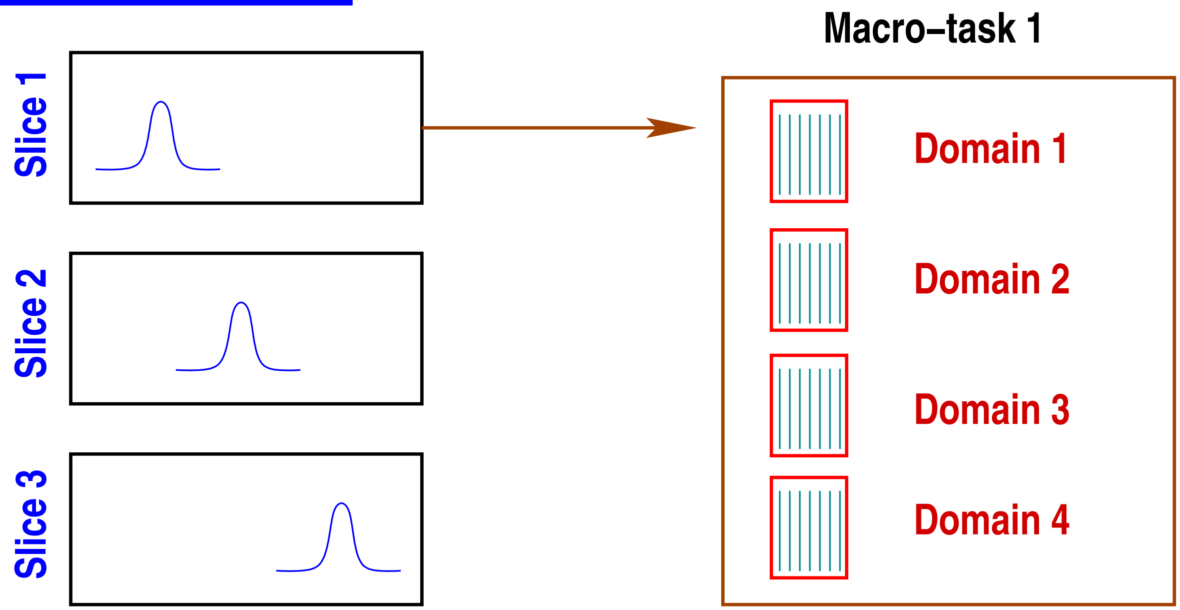
- EVSL uses polynomial and rational filters
- Each can be appealing in different situations.

> *Conceptually simple idea: cut the overall interval containing the spectrum into small sub-intervals and compute eigenpairs in each sub-interval independently.*

For each subinterval: select a filter polynomial of a certain degree so its high part captures the wanted eigenvalues. In illustration, the polynomials are of degree 20 (left), 30 (middle), and 32 (right).



Levels of parallelism



The two main levels of parallelism in EVSL

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.*

Average statistics per slice for different numbers of slice ($\boldsymbol{n_s}$), for the 3D discrete Laplacian example with $\phi = \mathbf{0.8}$.

n_s	deg	iter	matvec		CPU time	
			number	time	per slice	total
2	34.5	9284.5	328832.0	681.74	11817.35	23634.69
5	88.0	3891.8	347704.6	715.98	2126.97	10634.85
10	177.2	2065.4	365758.8	747.69	1116.91	11169.13
15	266.1	1351.9	361809.0	746.04	911.54	13673.12
20	356.8	1081.7	392083.3	807.46	909.62	18192.45

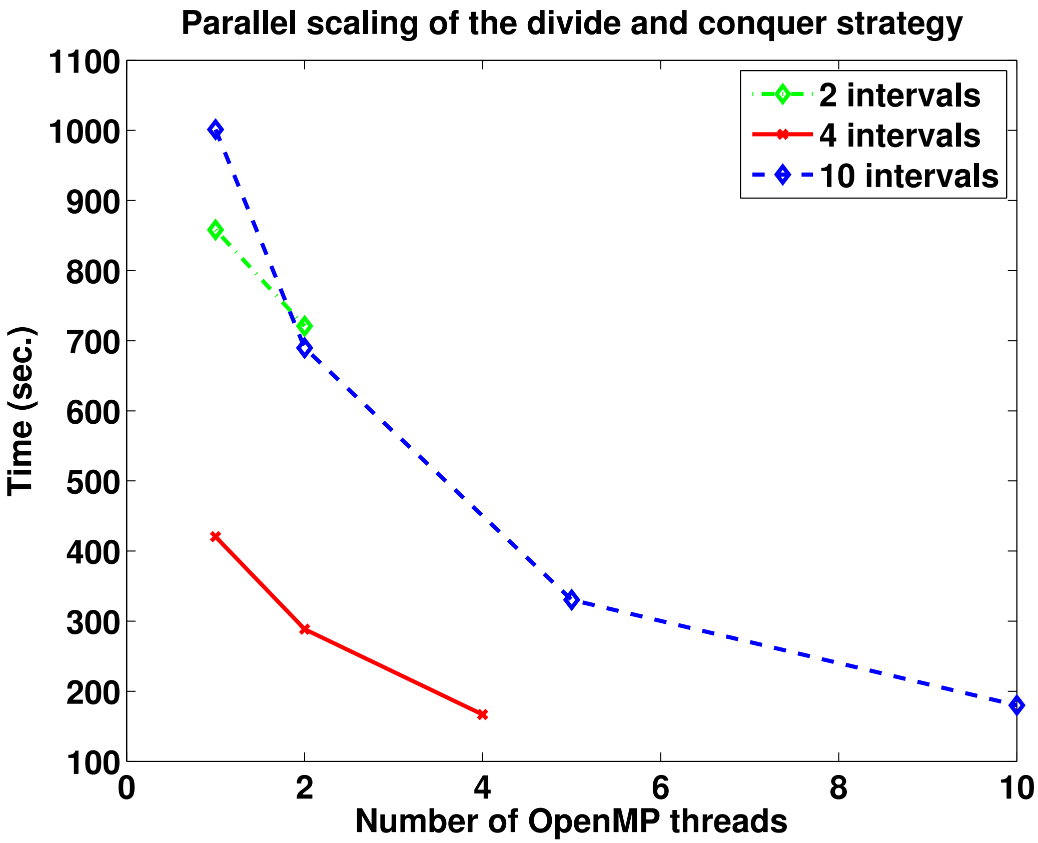
Hamiltonian matrices from the PARSEC set

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}

An OpenMP parallelization across 2, 4 and 10 spectral intervals of a divide and conquer approach for SiO matrix; $n = 33,401$ and $nnz = 1,317,655$.



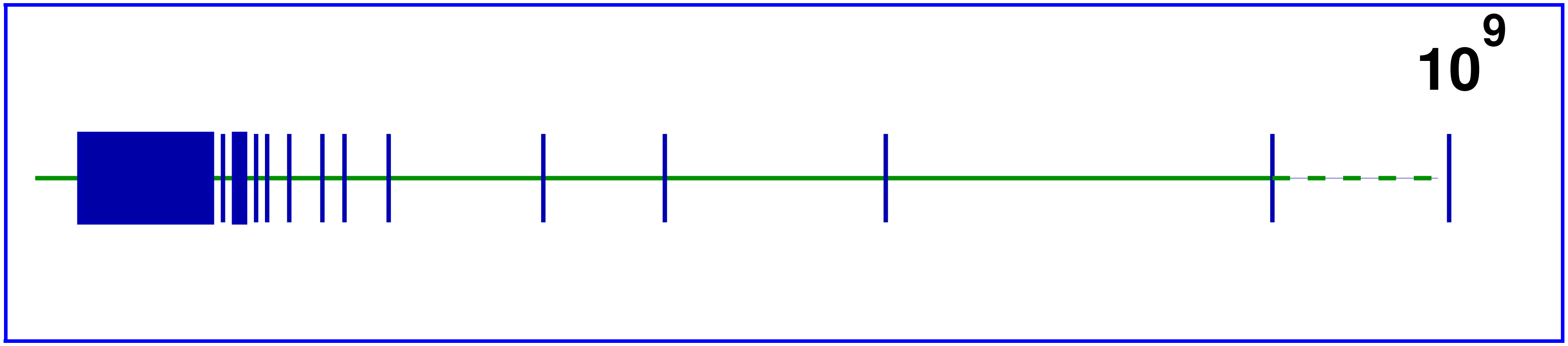
- Eigenvalues computed: 1002 lowest eigenpairs.

RATIONAL FILTERS

Why use rational filters?

** Joint work with Yuanzhe Xi

➤ Consider a spectrum like this one:



➤ Polynomial filtering utterly ineffective for this case

➤ Second issue: situation when Matrix-vector products are expensive

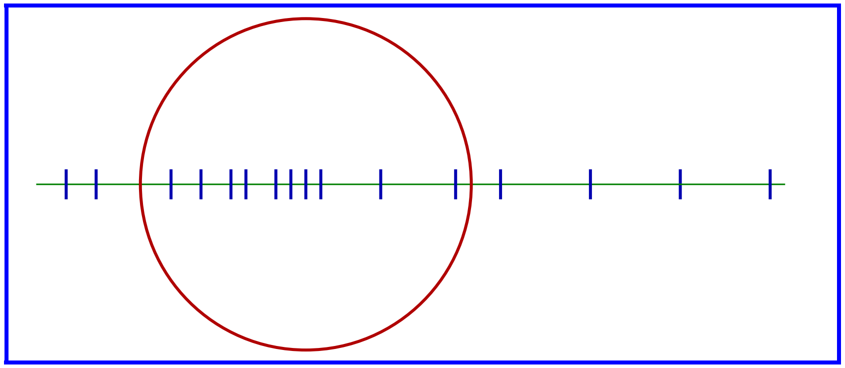
➤ Generalized eigenvalue problems.

- Alternative is to use rational filters:

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

- We now need to solve linear systems

- Tool: Cauchy integral representations of spectral projectors



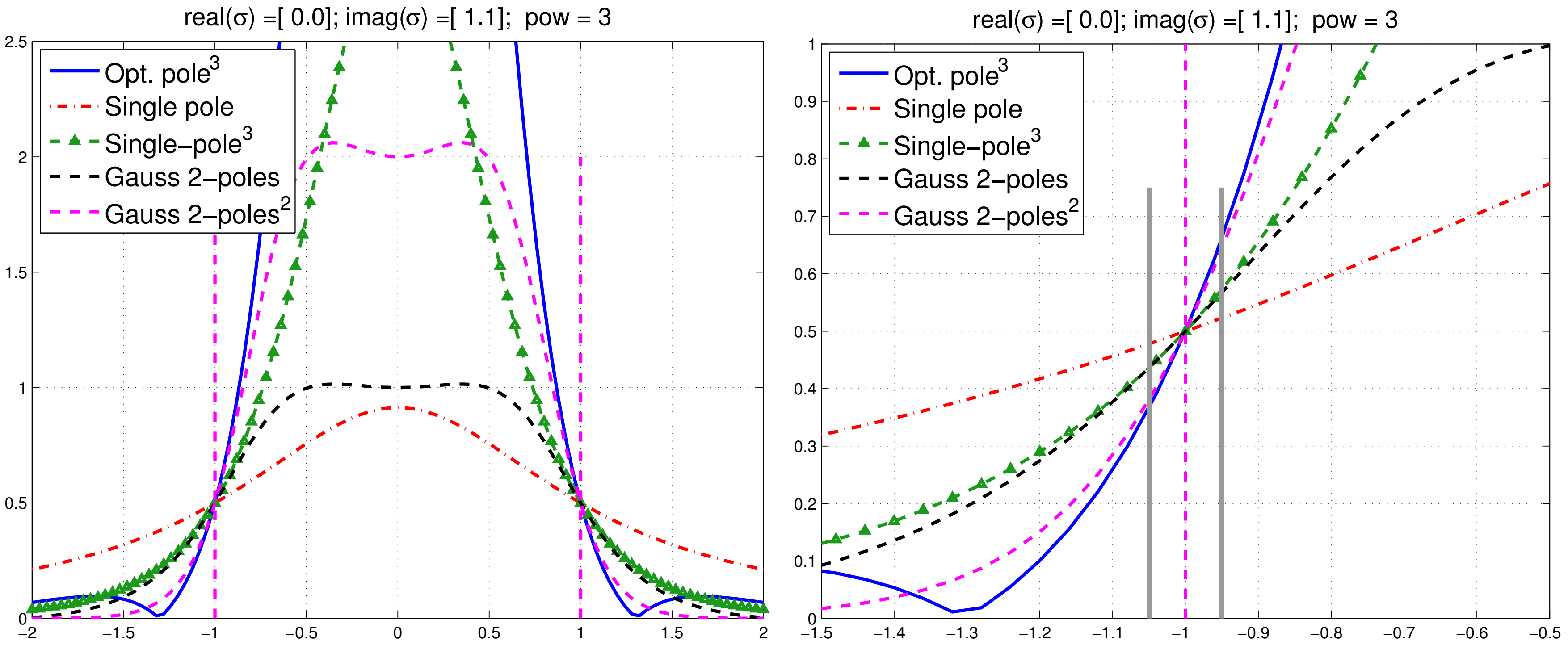
$$P = \frac{-1}{2i\pi}\int_\Gamma (A - sI)^{-1} ds$$

- 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

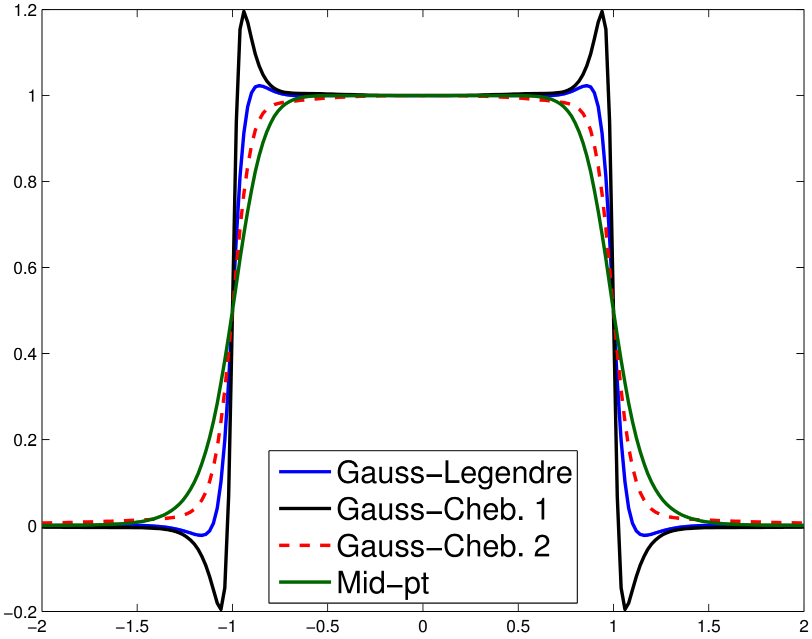


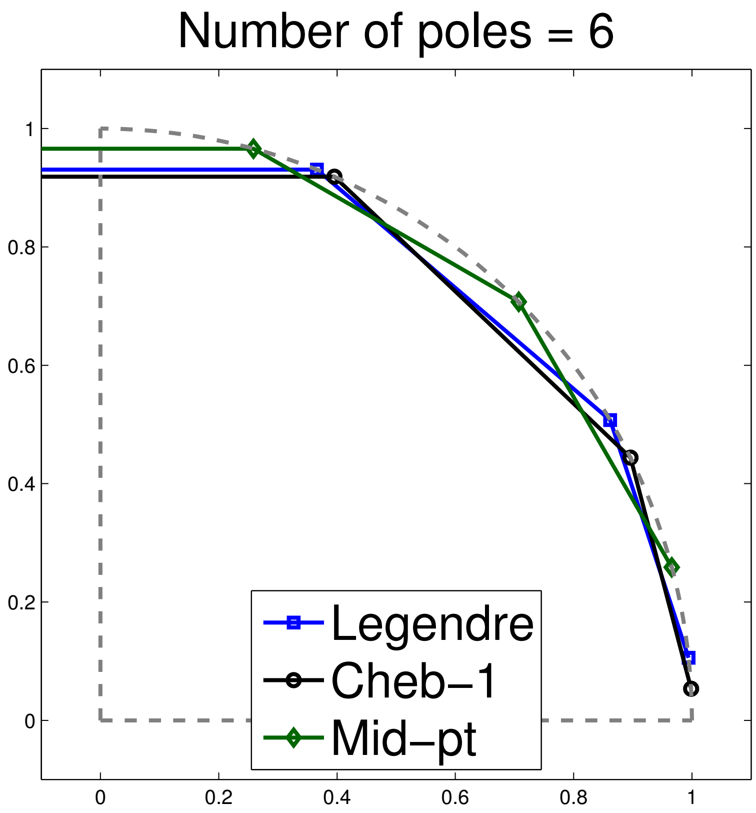
- Assume subspace iteration is used with above filters. Which filter will give better convergence?

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

The Cauchy integral viewpoint

- Standard Mid-point, Gauss-Chebyshev (1st, 2nd) and Gauss-Legendre quadratures. Left: filters, right: poles





- Notice how the sharper curves have poles close to real axis

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 $a = 10$,
$\beta = 0.1$

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

How does this work?

- A small example : Laplacean on a 43×53 grid. ($n = 2279$)

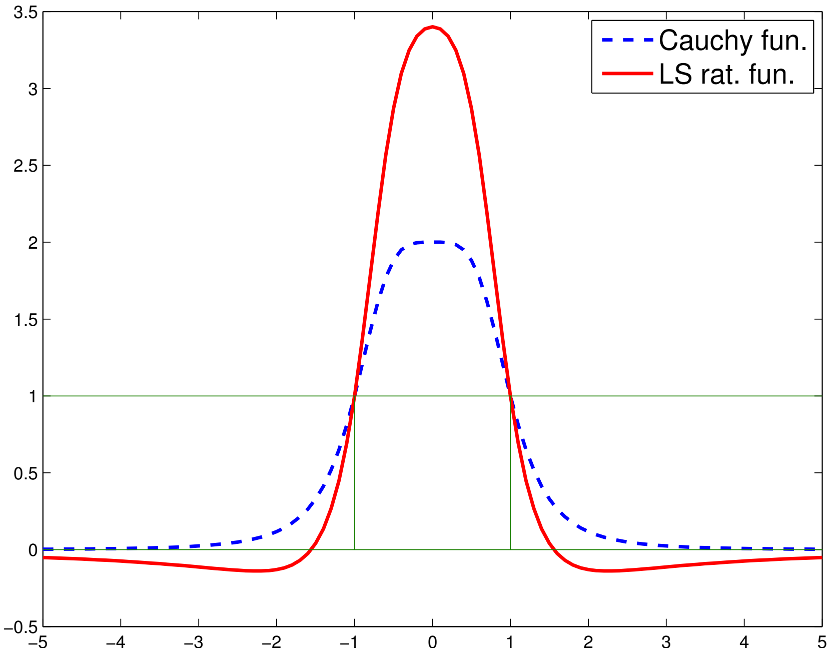
- Take 4 poles obtained from mid-point rule($N_c = 2$ on each 1/2 plane)

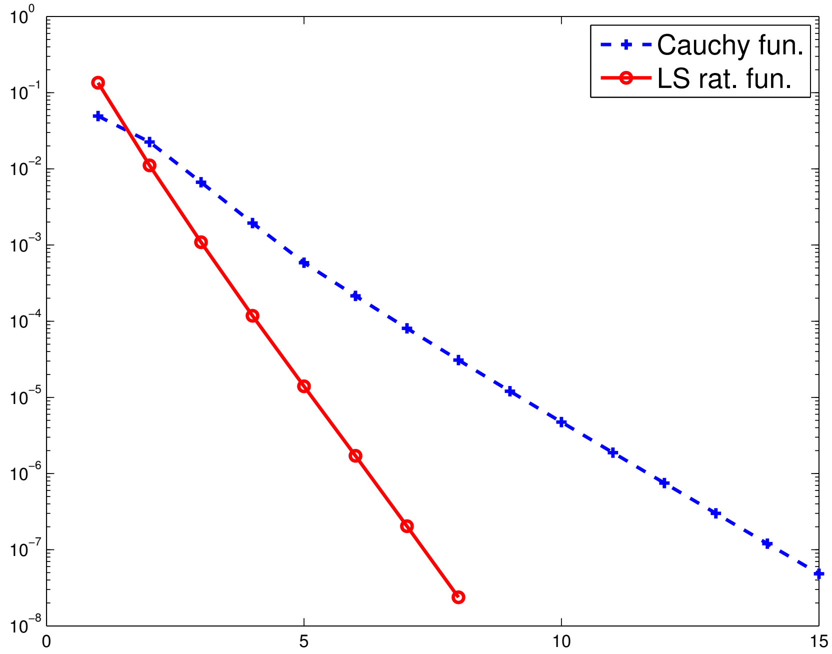
- Want: eigenvalues inside $[0, \ 0.2]$. There are $nev = 31$ of them.

- Use 1) standard subspace iteration + Cauchy (FEAST) then 2) subspace iteration + LS Rat. Appox.

- Use subspace of dim $nev + 6$

- $\beta = 0.2$



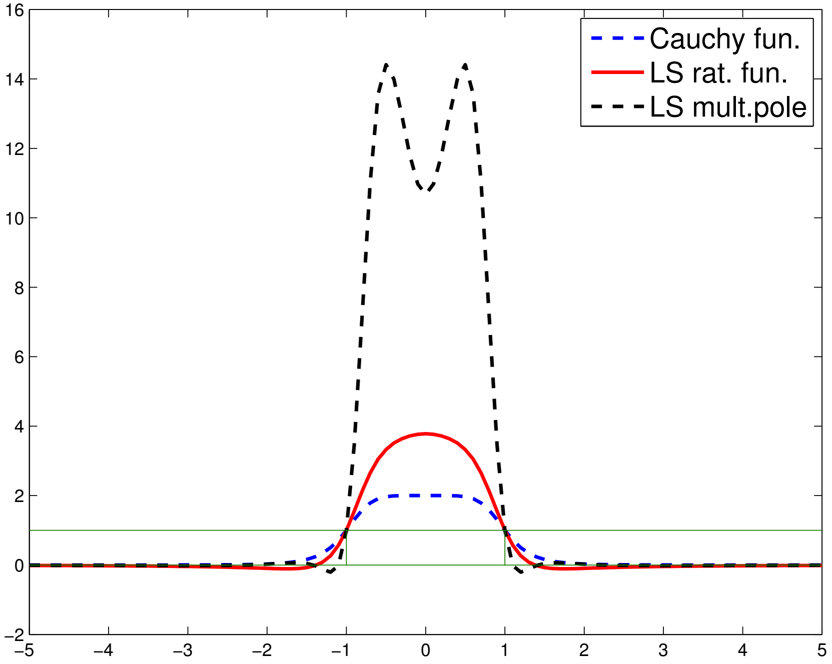


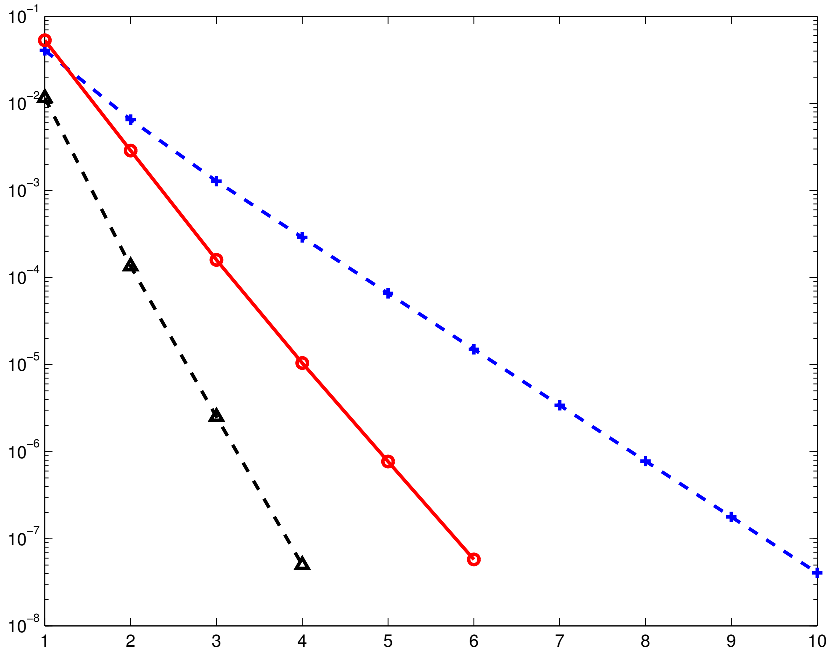
- LS Uses the same poles + same factorizations as Cauchy but
- ... much faster as expected from a look at the curves of the functions

- Other advantages:
  - Can select poles far away from real axis → faster iterative solvers [E. Di Napoli et al.]
  - Very flexible – can be adapted to many situations
  - Can use multiple poles (!)
- Implemented in EVSL.. [Interfaced to UMFPACK as a solver]

Better rational filters: Example

- Take same example as before $\mathbf{43} \times \mathbf{53}$ Laplacean
- Now take 6 poles [$\mathbf{3} \times \mathbf{2}$ midpoint rule]
- Repeat each pole [double poles.]



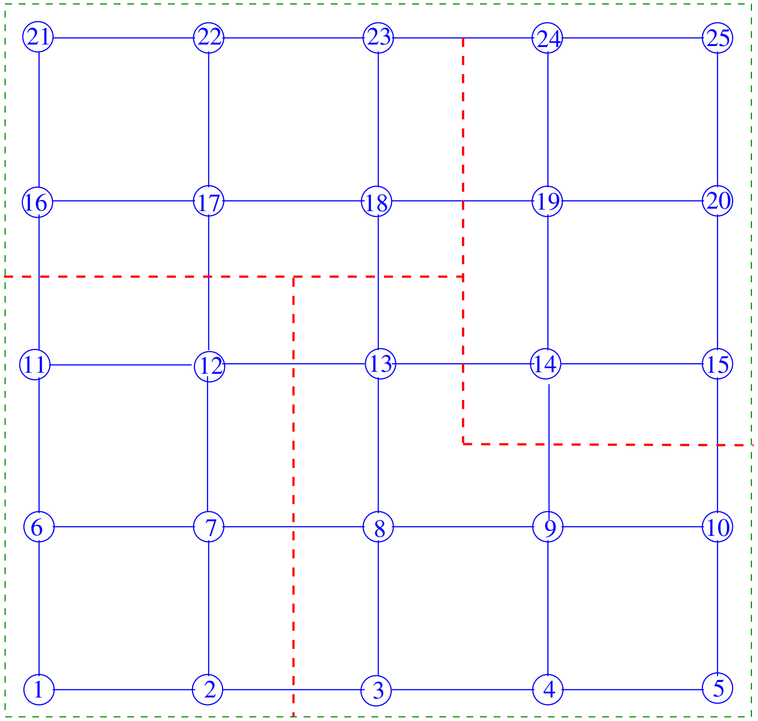


DOMAIN DECOMPOSITION

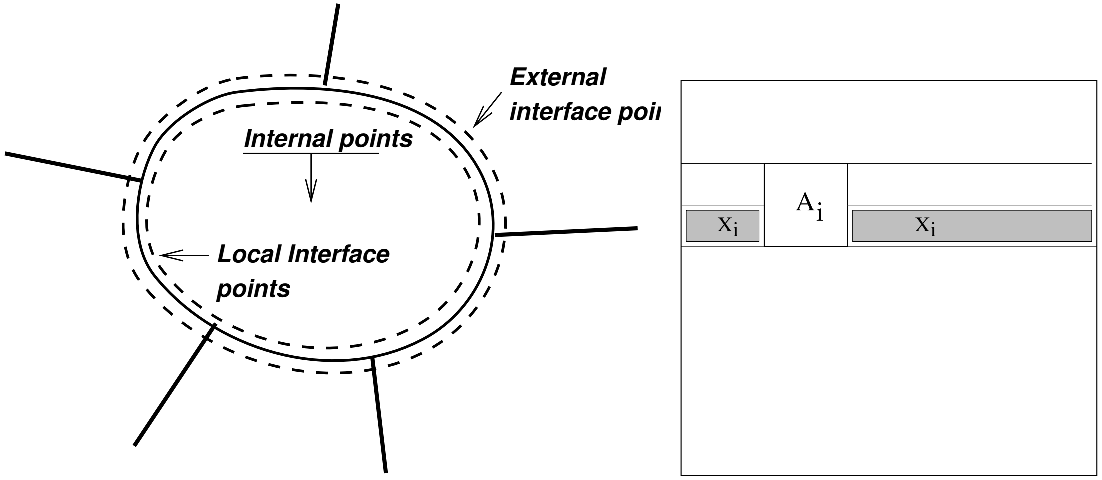
Introduction

* Joint work with Vasilis Kalantzis and Ruipeng Li

- ➤ Partition graph using edge-separators ('vertex-based partitioning')
- ➤ Common strategy: Exploit DD for matvec's



Distributed graph and its matrix representation



- Stack all interior variables $u_1, u_2, \cdots, u_p$ into a vector u, then interface variables y

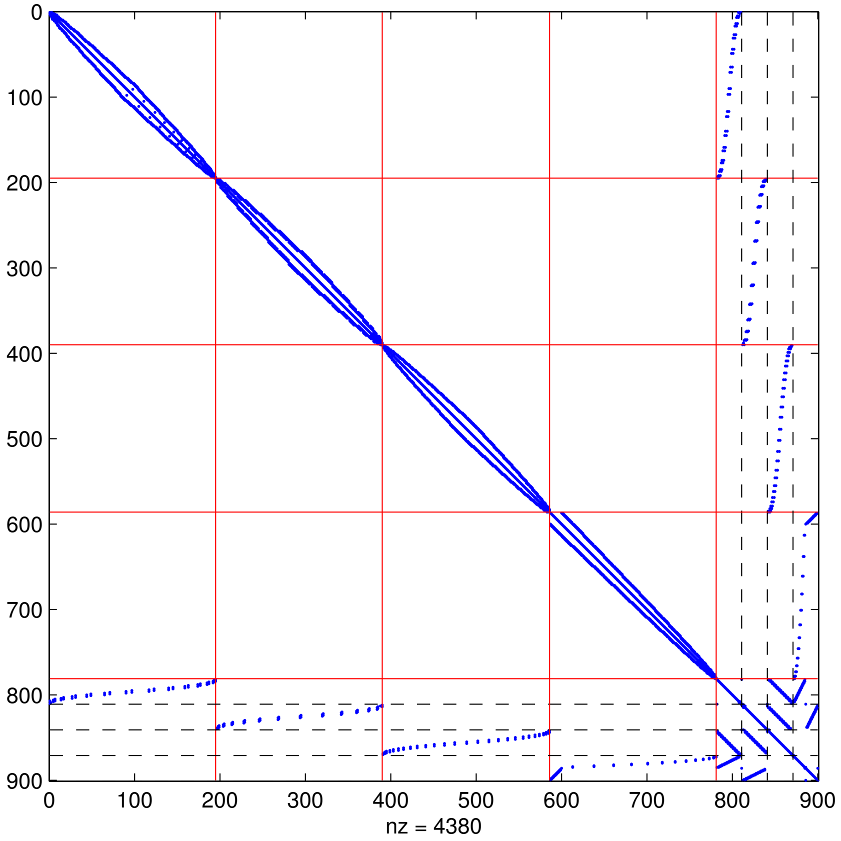
- Result:

$$\underbrace{\begin{pmatrix} B_1 & & & \dots & E_1 \\ & B_2 & & \dots & E_2 \\ \vdots & & \ddots & & \vdots \\ & & & B_p & E_p \\ E_1^T & E_2^T & \dots & E_p^T & C \end{pmatrix}}_{PAP^T} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{pmatrix}$$

Notation:

Write as:

$$A = \begin{pmatrix} B & E \\ E^T & C \end{pmatrix}$$



The spectral Schur complement

- Eliminating the u_i's we get

$$\begin{pmatrix} S_1(\lambda) & E_{12} & \cdots & E_{1p} \\ E_{21} & S_2(\lambda) & \cdots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1}^\top & E_{p2}^\top & \cdots & S_p(\lambda) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = 0$$

- $S_i(\lambda) = C_i - \lambda I - E_i^\top (B_i - \lambda I)^{-1} E_i$
- Interface problem (non-linear): $S(\lambda) y(\lambda) = 0.$
- Top part can be recovered as $u_i = -(B - \lambda I)^{-1} E_i y(\lambda)$.
- See also AMLS [Bennighof, Lehoucq, 2003]

Spectral Schur complement (cont.)

State problem as:

- Find $\sigma \in \mathbb{R}$ such that

One eigenvalue of $S(\sigma) \equiv 0$, or,

➤ $\mu(\sigma) = 0$ where $\mu(\sigma)$ == smallest (|.|) eig of $S(\sigma)$.

➤ Can treat $\mu(\sigma)$ as a function → root-finding problem.

➤ The function $\mu(\sigma)$ is analytic for any $\sigma \notin \Lambda(B)$ with

$$\frac{d\mu(\sigma)}{d\sigma} = -1 - \frac{\|(B - \sigma I)^{-1} E y(\sigma)\|_2^2}{\|y(\sigma)\|_2^2}.$$

Basic algorithm - Newton's scheme

- We can formulate a Newton-based algorithm.

ALGORITHM : 1. *Newton Scheme*

```
Select initial $\sigma$
Repeat:
    Compute $\mu(\sigma)$ = Smallest eigenvalue in modulus
    of $S(\sigma)$ & associated eigenvector $y(\sigma)$
    Set $\eta := \|(B - \sigma I)^{-1} E y(\sigma)\|_2$
    Set $\sigma := \sigma + \mu(\sigma)/(1 + \eta^2)$
Until: $|\mu(\sigma)| \leq \mathrm{tol}$
```

- $\mu(\sigma)$ computed by an inverse iteration scheme.
- Details omitted

Complex Rational Filter + Schur complements

- Goal: DD techniques in contour integral-based methods

$$A - sI = \begin{pmatrix} B - sI & E \\ E^T & C - sI \end{pmatrix} \rightarrow$$

$$(A - sI)^{-1} = \left[\begin{array}{c|c} * & -(B - sI)^{-1} E S(s)^{-1} \\ \hline * & S(s)^{-1} \end{array}\right]$$

- Then, Cauchy integral formula for spectral projector yields:

$$P = \frac{-1}{2i\pi} \int_\Gamma R(s) ds \equiv \left[\begin{array}{c|c} * & -W \\ \hline * & G \end{array}\right] \quad \text{with}$$

$$G = \frac{-1}{2i\pi} \int_\Gamma S(s)^{-1} ds, \quad W = \frac{-1}{2i\pi} \int_\Gamma (B - sI)^{-1} E S(s)^{-1} ds$$

- Advantage: Does not involve inverse of whole matrix

- Let $$\boxed{P = [P_1, P_2] \equiv \left[\begin{array}{c|c} * & -W \\ \hline * & G \end{array}\right]}$$

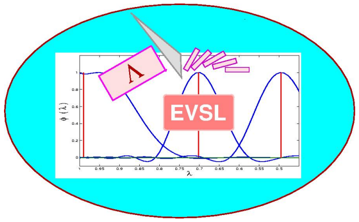
- We know how to compute P_2 or $P_2 \times$ `randn` (s, ns)

Q: How can we recover eigenvectors of A from P_2?

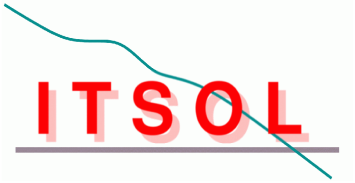
A: Observe: if $P = VV^T$, and $V = \binom{V_u}{V_s}$ then $P_2 = VV_s^T$

- Just capture the range of P_2 : Subspace iteration - Lanczos
- This approach is a one-shot method [no easy way to iterate]
- See: V. Kalantzis, J. Kestyn, E. Polizzi, and YS *PFEAST: A High Performance Eigenvalue Solver Using Three Full Levels of MPI Parallelism* in Proc. Supercomputing'16.

SOFTWARE



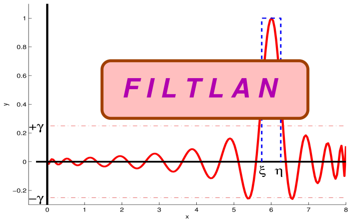
EVSL NEW! a library of (sequential) eigensolvers based on spectrum slicing. **Version 1.0** released on [09/11/2016]
EVSL provides routines for computing eigenvalues located in a given interval, and their associated eigenvectors, of real symmetric matrices. It also provides tools for spectrum slicing, 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.



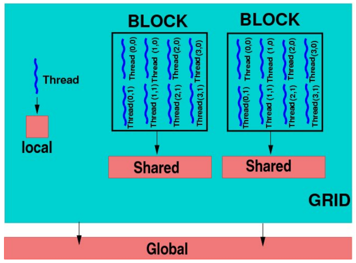
ITSOL a library of (sequential) iterative solvers. **Version 2** released. [11/16/2010]
ITSOL can be viewed as an extension of the ITSOL module in the SPARSKIT package. It is written in C and aims at providing additional preconditioners for solving general sparse linear systems of equations. Preconditioners 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.

---

pARMS **Version 3.2** released. [11/16/2010]
A portable library of distributed-memory sparse iterative solvers. Version 3.2 posted.
Note: PSPARSLIB [a FORTRAN77 portable library of distributed-memory sparse iterative solvers released first circa 1995] is no longer posted. pARMS replaces PSPARSLIB. The older version of pARMS [pARMS_2.2] will remain posted but will have no support.
This work was supported by the Department Of Energy.



FILTLAN **Version 1.0a** released. [08/22/2011]
A Filtered Lanczos package for solving interior and extreme symmetric eigenproblems. Version 1.0a posted. Suppose you want to compute all the eigenvalues of a matrix A that are located in an interval which is a subset deep inside the spectrum of A. The matrix A is symmetric, and may be issued from the discretization of a 3-D problem (e.g., a Poisson operator), so shift-and-invert may not be an option. In this situation a filtered Lanczos approach is ideal and tests reveal that this approach is very effective, especially when the number of eigenvalues-eigenvectors to be computed is very large.
This work was supported by the Department Of Energy.



CUDA_ITSOL [05/13/2011] The CUDA Iterative Solver package. This is a package for performing various sparse matrix operations and, more importantly, for solving sparse linear systems of equations. It is written under CUDA. The package was developed by Ruipeng Li [PhD Student, Univ. of Minnesota].
See this technical report for details on the methods implemented and more.
This work was supported by the Department Of Energy.

Conclusion

Part I: Polynomial filtering

- Polnym. Filter. appealing when # of eigenvectors to be computed is large and when Matvecs are inexpensive
- Will not work too well for generalized eigenvalue problem
- Will not work well for spectra with very large outliers.

Part II: Rational filtering

- We must rethink the way we view Rational filtering - away from Cauchy and into approximation of functions. LS approach is flexible, easy to implement, easy to understand.

Part III: Domain Decomposition

➤ We *must* combine DD with any filtering technique [rational or polynomial]

➤ Many ideas still to explore in Domain Decomposition for interior eigenvalue problems

➤ `EVSL` code available here:

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

➤ Fully parallel version (MPI) in the works