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

MATH + X Symposium
Houston, Jan 19, 2017

Background. Origins of Eigenvalue Problems

- Structural Engineering [$Ku = \lambda Mu$] (Goal: frequency response)
- Electronic structure calculations [Schrödinger equation..]
- Stability analysis [e.g., electrical networks, mechanical system,..]
- Bifurcation analysis [e.g., in fluid flow]

➤ Large eigenvalue problems in quantum chemistry use up biggest portion of the time in supercomputer centers

Background. New applications in data analytics

- Machine learning problems often require a (partial) *Singular Value Decomposition* -
- Somewhat different issues in this case:
  - Very large matrices, update the SVD
  - Compute dominant singular values/vectors
  - Many problems of approximating a matrix by one of lower rank (Dimension reduction, ...)
- But: Methods for computing SVD somewhat similar to those for standard eigenvalue problems

Background. The Problem (s)

We consider the eigenvalue problem

$$\boldsymbol{Ax = \lambda x}$$

where $\boldsymbol{A}$ is symmetric real (or Hermitian complex)

- Also: $\boldsymbol{Ax = \lambda Bx}$ where $\boldsymbol{B}$ is symmetric positive definite, $\boldsymbol{A}$ is symmetric or nonsymmetric
- What to compute:
  - A few $\boldsymbol{\lambda}_i$'s with smallest or largest real parts;
  - All $\boldsymbol{\lambda}_i$'s in a certain region of $\mathbb{C}$;
  - A few of the dominant eigenvalues;
  - All $\boldsymbol{\lambda}_i$'s (rare).

Background. The main tools

Projection process: (a) Build a 'good' subspace $K = \mathbf{span}(V)$; (b) get approximate eigenpairs by a Rayleigh-Ritz process:

$$V^T(A - \tilde{\lambda} I)Vy = 0$$

- $\tilde{\lambda}$ = Ritz value, $\tilde{u} = Vy$ = Ritz vector.
- Two common choices for K:
  1) Power subspace $K = \mathbf{span}\{A^k X_0\}$; or $\mathbf{span}\{P_k(A) X_0\}$;
  2) Krylov subspace $K = \mathbf{span}\{v, Av, \cdots, A^{k-1}v\}$

Background. The main tools (cont)

Shift-and-invert:

➤ If we want eigenvalues near σ, replace A by $(A - \sigma I)^{-1}$.

Example: power method: $v_j = Av_{j-1}/$scaling replaced by

$$v_j = \frac{(A - \sigma I)^{-1} v_{j-1}}{\text{scaling}}$$

➤ Works well for computing *a few* eigenvalues near σ/

➤ Used in commercial package NASTRAN (for decades!)

➤ Requires factoring $(A - \sigma I)$ (or $(A - \sigma B)$ in generalized case.) But convergence will be much faster.

➤ A solve each time - Factorization done once (ideally).

Background. The main tools (cont)

Deflation:

➤ Once eigenvectors converge remove them from the picture

Restarting Strategies :

➤ Restart projection process by using information gathered in previous steps

---

➤ ALL available methods use some combination of these ingredients.

[e.g. ARPACK: Arnoldi/Lanczos + 'implicit restarts' + shift-and-invert (option).]

Large problems in applications

➤ 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.

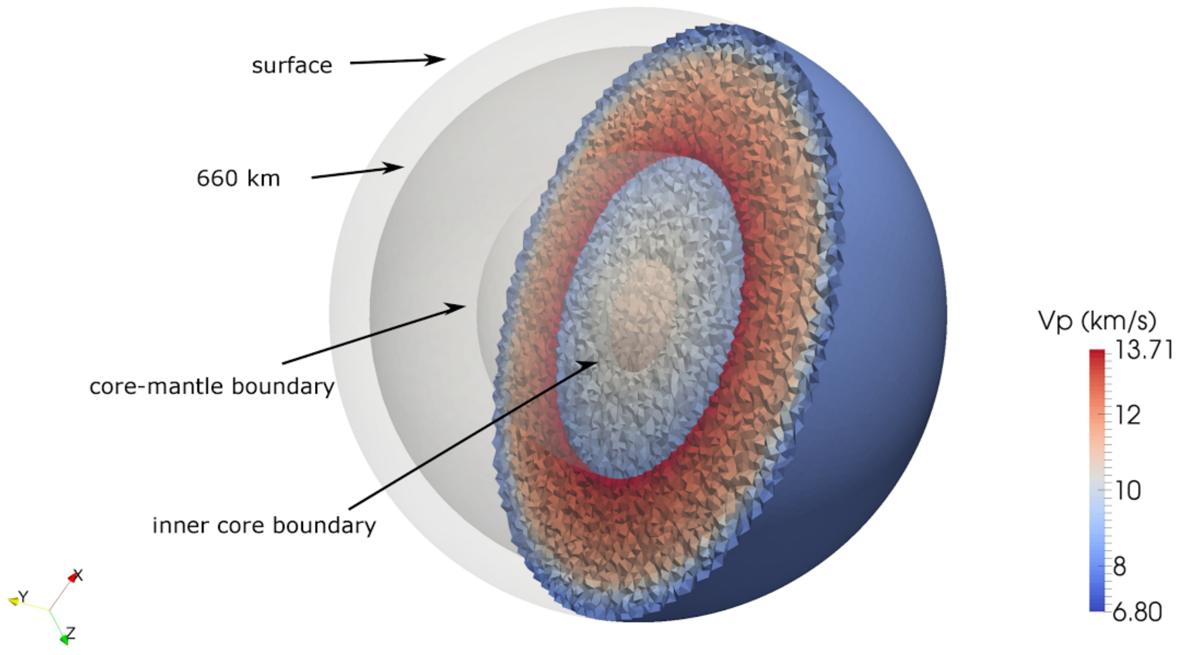
➤ Density Functional Theory in electronic structure calculations: *'ground states'*

➤ *Excited states* involve transitions and invariably lead to much more complex computations. $\longrightarrow$ Large matrices, *many* eigen-pairs to compute

Illustration:

'Hamiltonian of size $n \sim 10^6$ *get 10% of bands'*

Computing earth normal modes (J. Shi & M. V. De Hoop)



- FEM model leads to a generalized eigenvalue problem:

$$\begin{bmatrix} A_{CM} & & & B_{CMB} \\ & A_{IC} & & B_{ICB} \\ & & 0 & A_{dp} \\ B_{CMB}^T & B_{IMB}^T & A_{dp}^T & A_{OC} \end{bmatrix} \begin{bmatrix} u^{CM} \\ u^{IC} \\ u^{OC} \\ p \end{bmatrix} = \omega^2 \begin{bmatrix} M_{CM} & & & \\ & M_{IC} & & \\ & & M_{OC} & \\ & & & 0 \end{bmatrix} \begin{bmatrix} u^{CM} \\ u^{IC} \\ u^{OC} \\ p \end{bmatrix}$$

- Want all eigen-values/vectors inside a given interval
- Issue: ‘mass’ matrix has a large null space..
- Solution: change formulation of matrix problem.

➤ ... Work in progress.

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,..), e.g., ARPACK code, PRIMME.
  - Block Algorithms [Block Lanczos, TraceMin, LOBPCG, SlepSc,...]
  - + 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

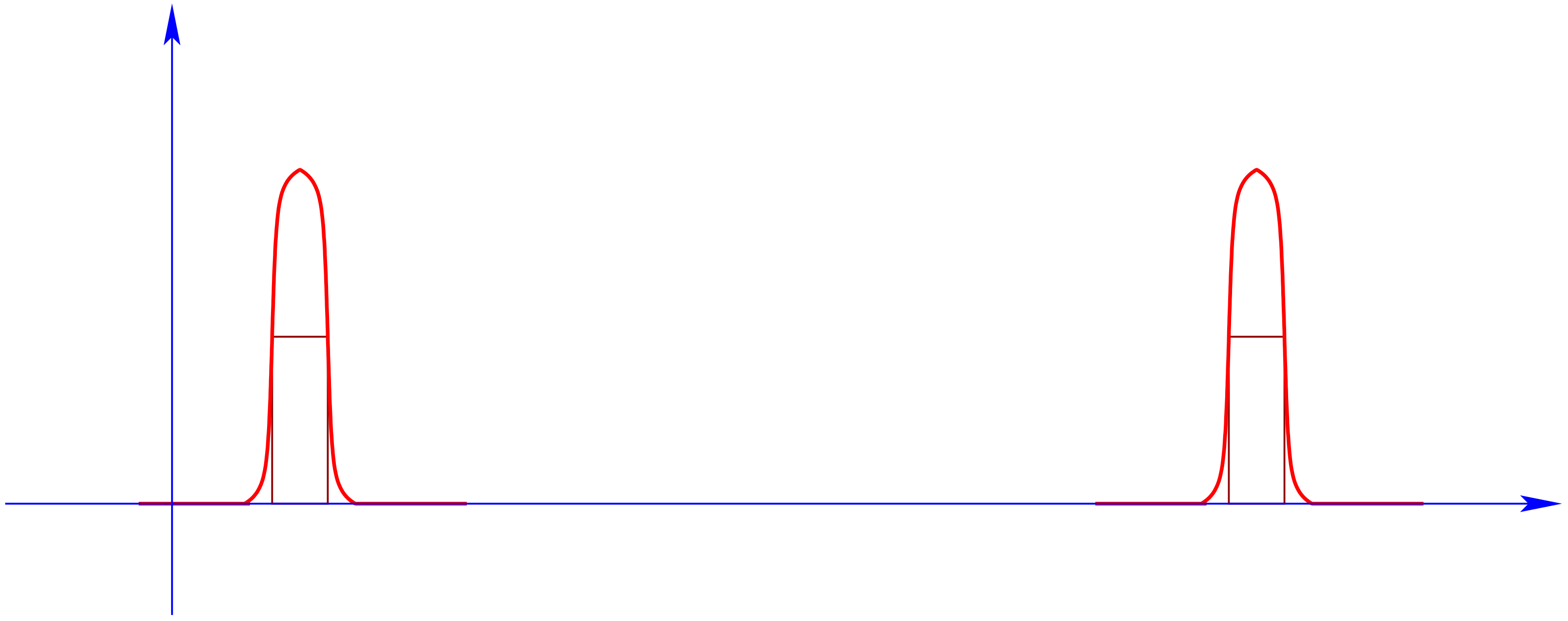
Issues with shift-and invert (and related approaches)

- ➤ Issue 1: factorization may be too expensive
  - ● Can use iterative methods?
- ➤ Issue 2: Iterative techniques often fail –
  - ● Reason: Highly indefinite problems.

➤ Alternative to shift-and-invert: ‘Spectrum slicing’ with Polynomial filtering

Spectrum slicing for computing many eigenvalues

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

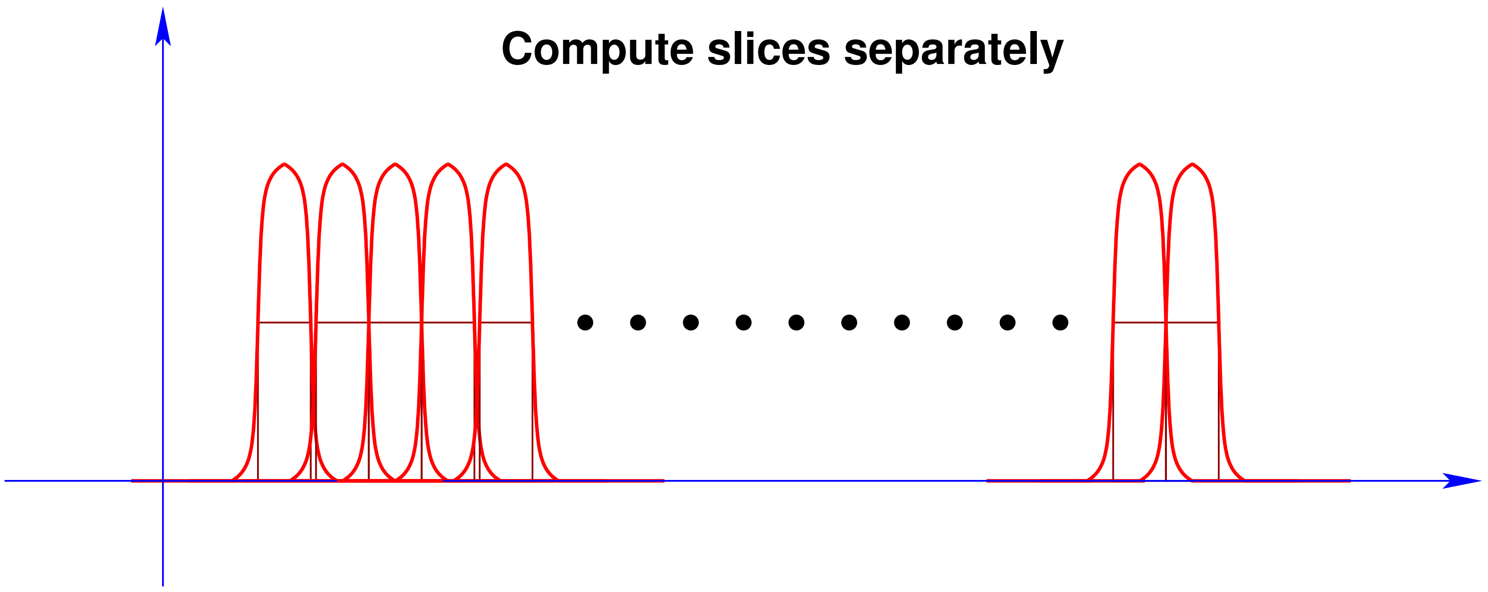


➤ Idea: Get the spectrum by ‘slices’ or ’windows’ [e.g., a few hundreds or thousands of pairs at a time]

➤ 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

➤ Expensive (memory+arithmetic) for some (e.g. large 3D) problems.

➤ First Alternative: Polynomial filtering

Polynomial filtering

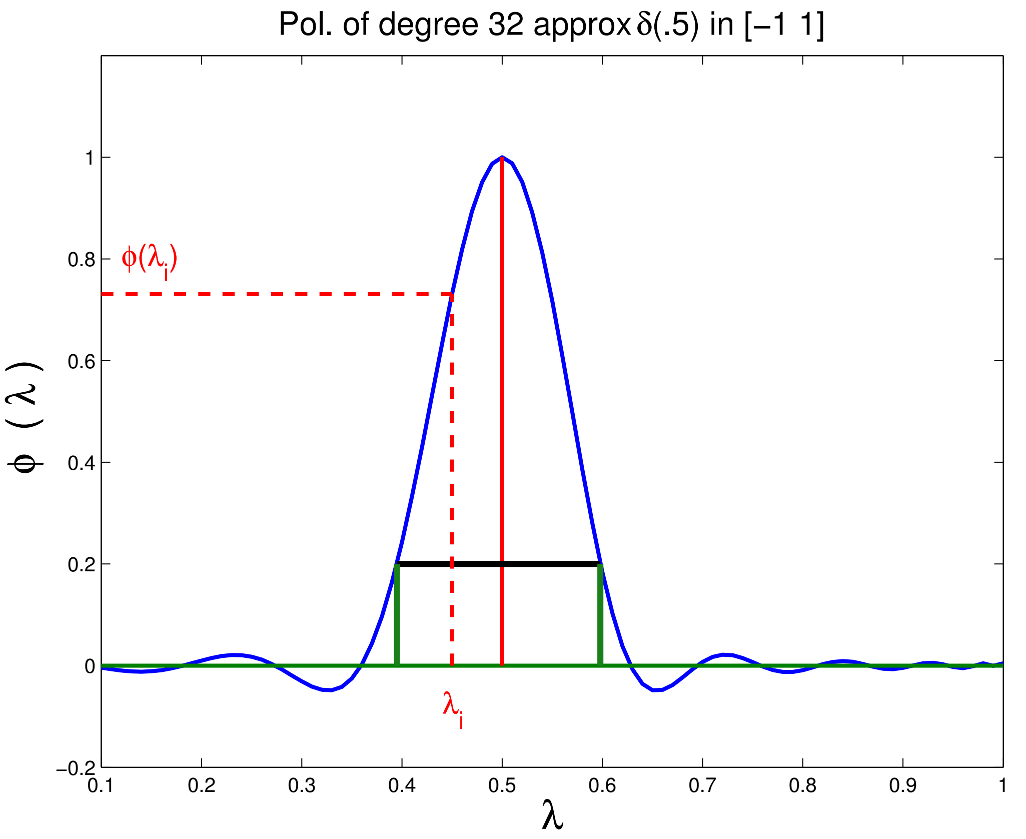
- Apply Lanczos or Subspace iteration to: $\boxed{M = \phi(A)}$ where $\phi(t)$ is a polynomial
- Each *matvec* $y = Av$ is replaced by $y = \phi(A)v$
- Eigenvalues in high part of filter will be computed first
- Old (forgotten) idea. But new context is *very* favorable
- Consider Subspace Iteration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

Matlab Script

```
for iter=1:maxit
        Y = PolFilt(A, V, mu);
        [V, R] = qr(Y,0);
%— Rayleigh-Ritz with A
        C = V'*(A*V);
        [X, D] = eig(C);
        d = diag(D);
%— get e-values closest to center of interval
        [∼, idx] = sort(abs(d-gam),'ascend');
%— get their residuals
        ...
%— if enough pairs have converged exit
        ...
end
```

What polynomials?

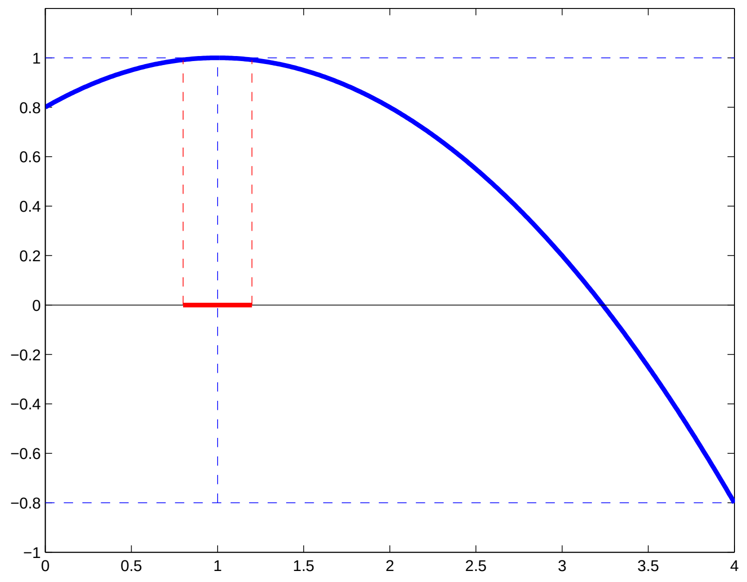
- For end-intervals: use standard Chebyshev polynomials
- For inside intervals: several choices



- Recall the main goal:

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

Use of quadratics: An old idea



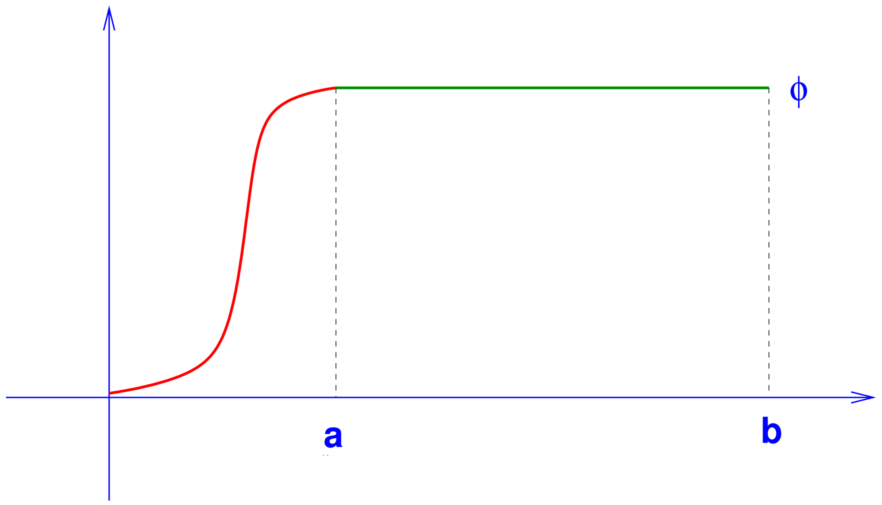
- We want to compute eigenvalues near $\sigma = 1$ of a matrix A with $\Lambda(A) \subseteq [0,\ 4]$.
- Use the simple transform: $p_2(t) = 1 - \alpha(t - \sigma)^2$.
- For $\alpha = .2, \sigma = 1$ you get $\longrightarrow$
- Use Subs. Iter. with $M = p_2(A)$.
- Eigenvalues near σ become the dominant ones – so Subspace Iteration will work – but...
- ... they are now poorly separated $\longrightarrow$ slow convergence.

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]

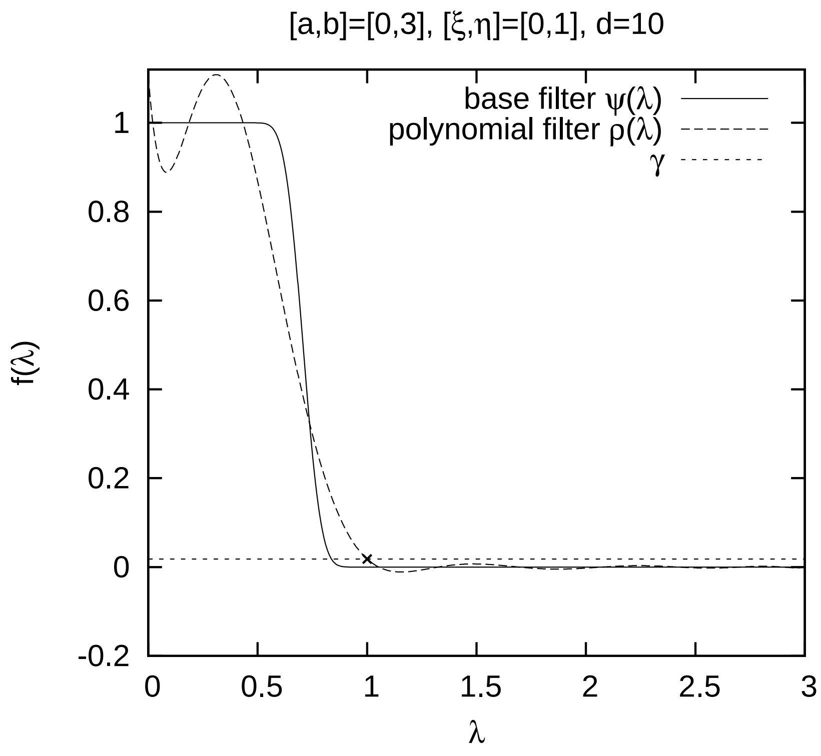


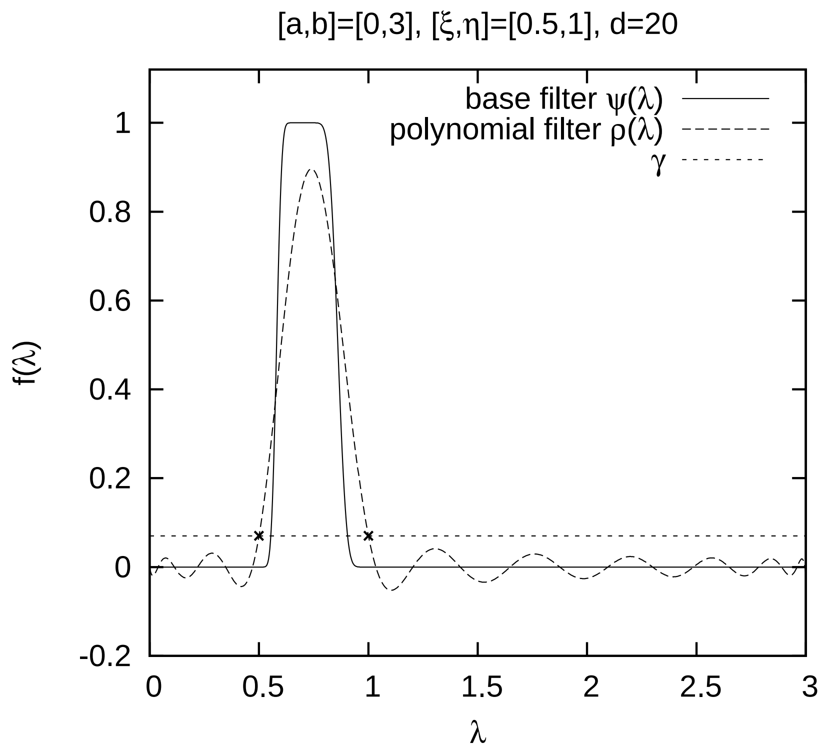
- Then approximate base filter by degree k polynomial in a least-squares sense.
- No numerical integration needed

Main advantage: very flexible.

➤ Details skipped

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

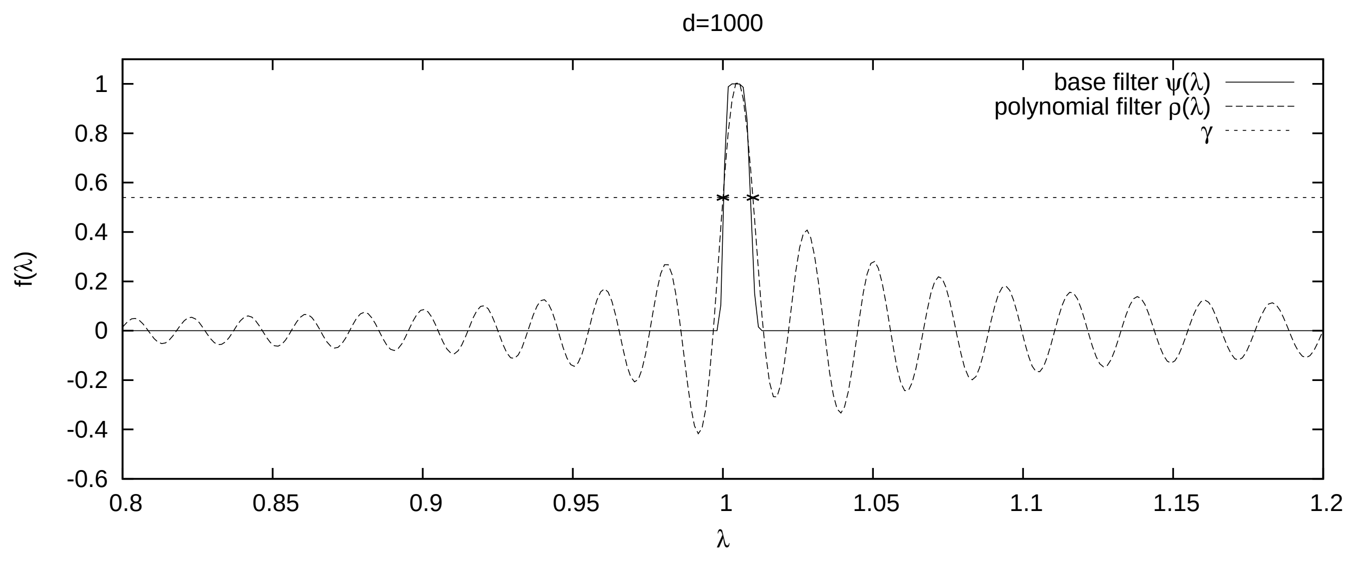




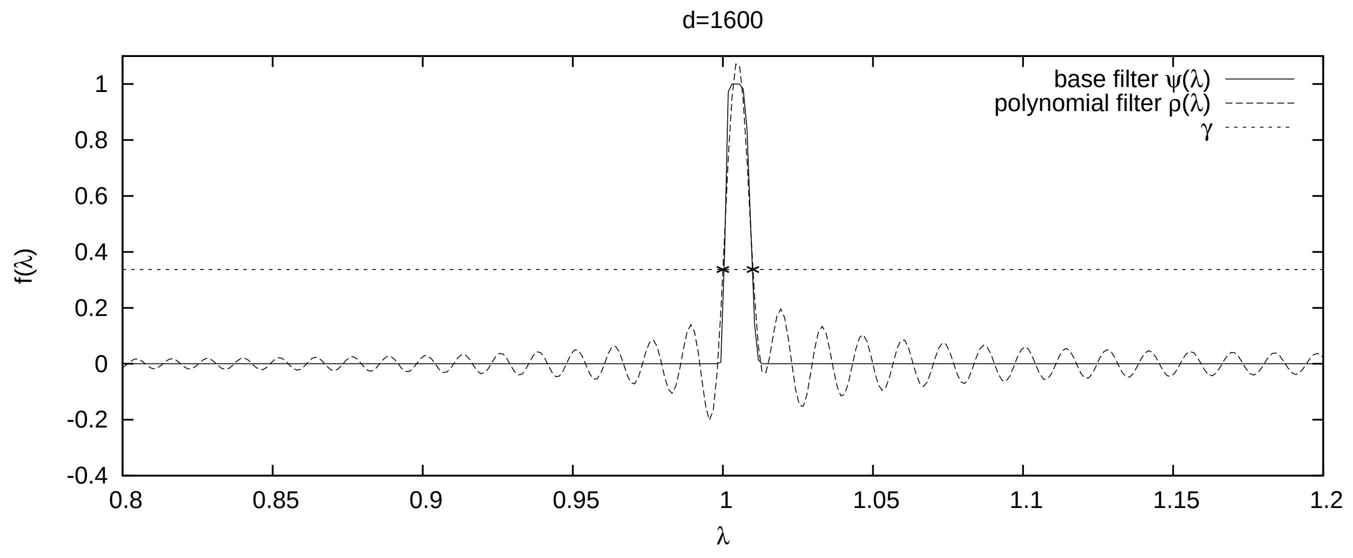
- ➤ See Reference on Lanczos + pol. filtering: Bekas, Kokiopoulou, YS (2008) for motivation, etc.
- ➤ H.-r Fang and YS “Filtlan” paper [SISC,2012] and code

Misconception: High degree polynomials are bad

Degree 1000 (zoom)



Degree 1600 (zoom)

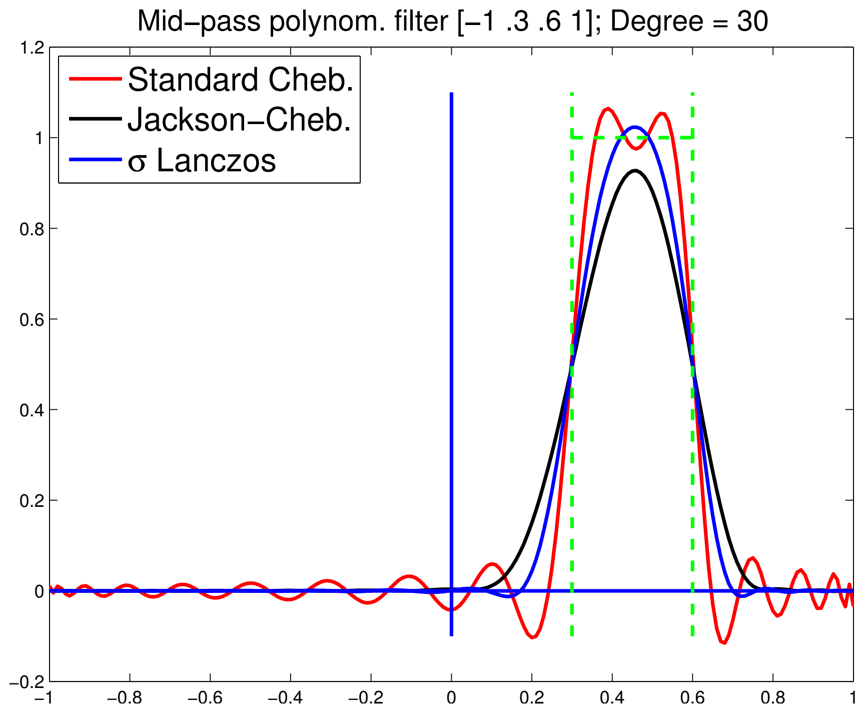


Simpler: Step-function Chebyshev + Jackson damping

➤ 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

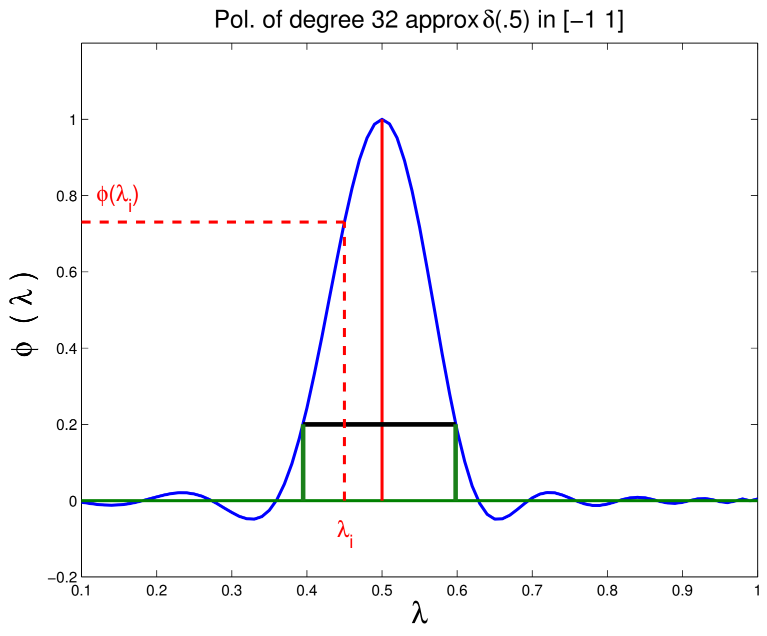


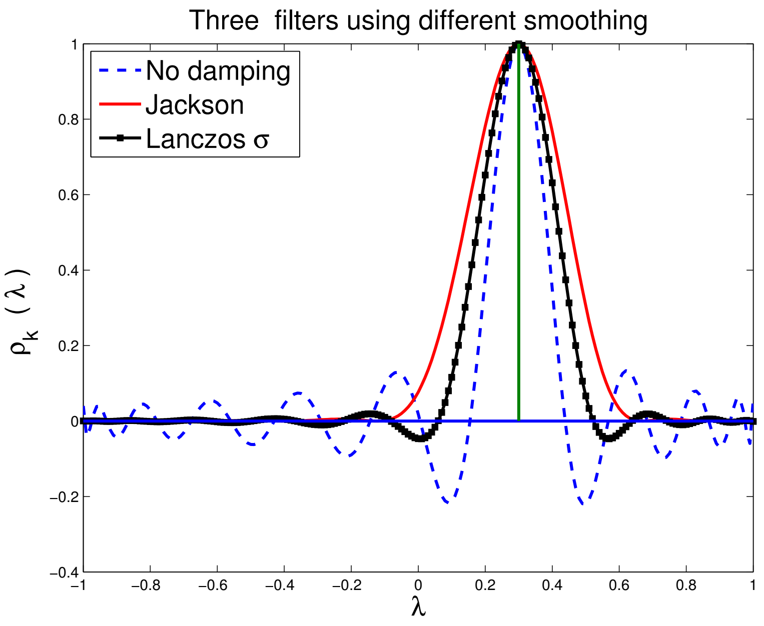
➤ G. Schofield, J. R. Chelikowsky and YS, CPC, 183, ('11)

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. $\longrightarrow$





$\longleftarrow$ Can use same damping: Jackson, Lanczos $\boldsymbol{\sigma}$ damping, or none.

Theory

The Chebyshev expansion of δ_γ is

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

➤ 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

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

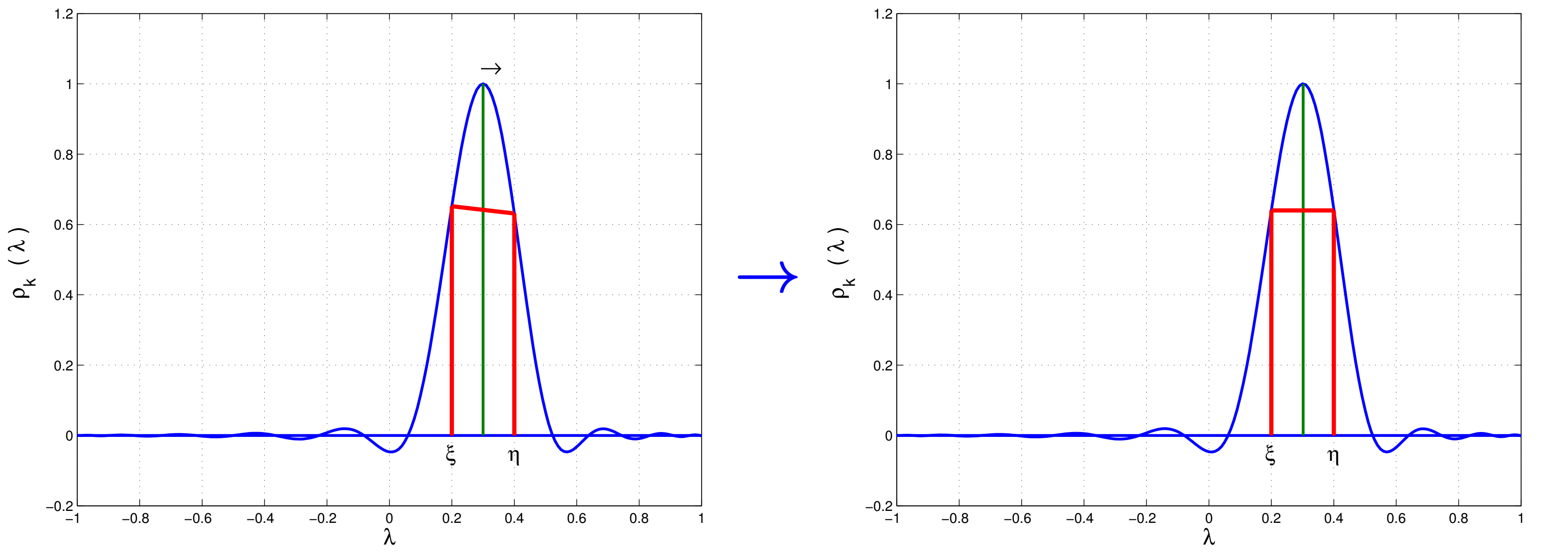
$$\mu_k = \begin{cases} 1/2 & \text{if } k == 0 \\ \cos(k \cos^{-1}(\gamma)) & \text{otherwise} \end{cases}$$

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

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

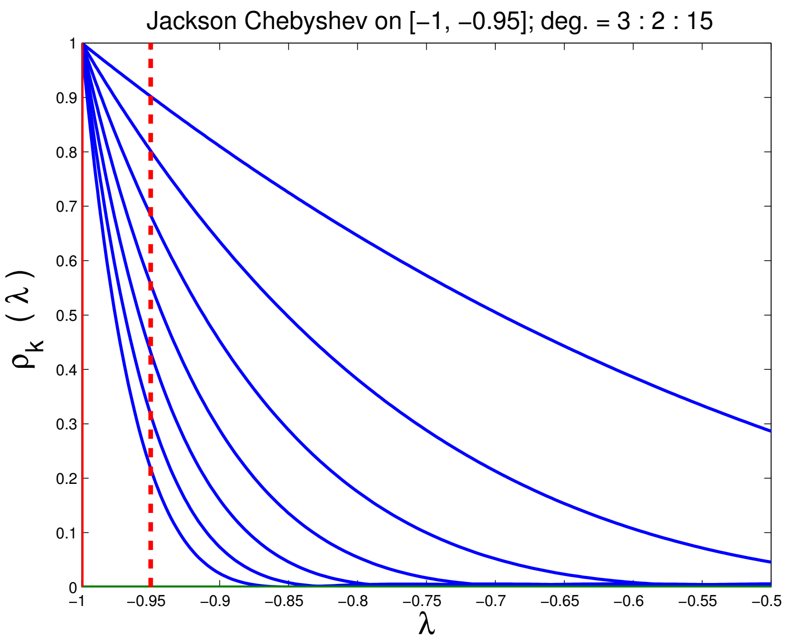
Issue # one: 'balance the filter'

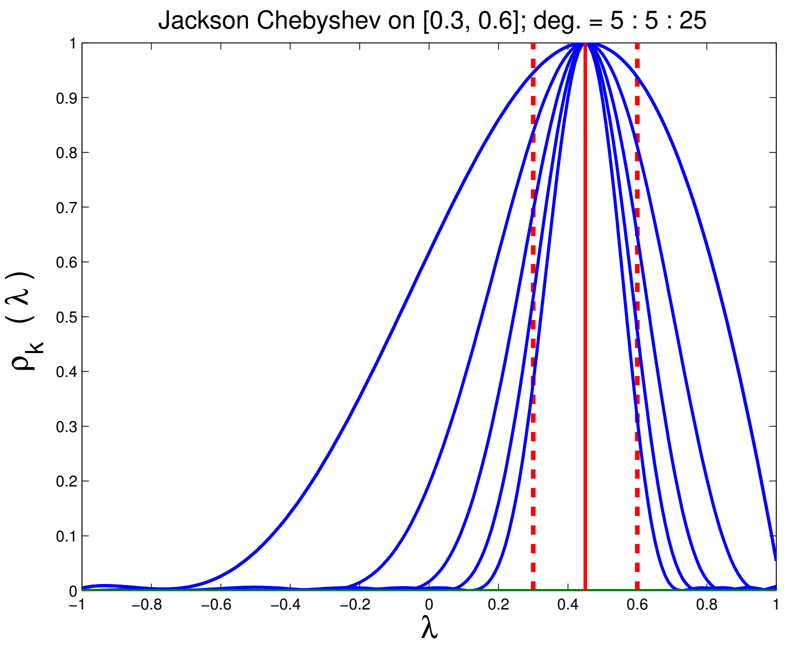
- To facilitate the selection of *'wanted'* eigenvalues [Select $\boldsymbol{\lambda}$'s such that $\phi(\boldsymbol{\lambda}) >$ bar] we need to ...
- ... find γ so that $\phi(\boldsymbol{\xi}) == \phi(\boldsymbol{\eta})$



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)

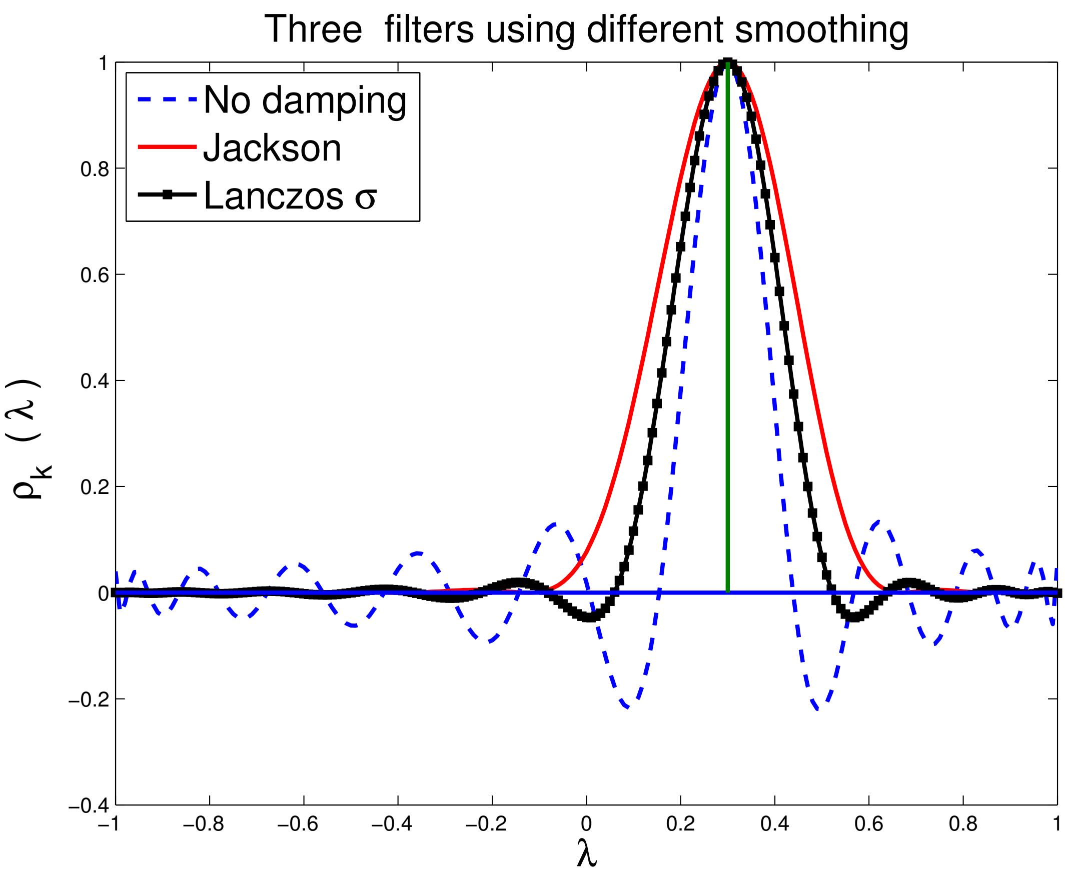




- 1) Start low (e.g. 2); 2) Increase degree until value (s) at the boundary (ies) become small enough –
- Eventually w'll use criterion based on derivatives at ξ & η

Issue # Three : Gibbs oscillations

➤ Discontinuous 'function' approximated $\rightarrow$ Gibbs oscillations



➤ Three options:

- No damping
- Jackson damping
- Lanczos σ damping

➤ Good compromise: Lanczos σ damping

Backround: The Lanczos Algorithm

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

➤ ... 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:

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

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

Projection: Lanczos vs. Subspace iteration

➤ Subspace iteration is quite appealing in some applications (e.g., electronic structure): $\boxed{\text{Can re-use previous subspace.}}$

➤ 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.

Polynomial filtered Lanczos

- Use the Lanczos algorithm with A replaced by $p_k(A)$, where $p_k(t) =$ polynomial of degree k
- Idea not new (and not too popular in the past)

What is new?

1. Very large problems;
2. (tens of) Thousands of eigenvalues;
3. Parallelism.

- Combine with spectrum slicing
- Main attraction: reduce cost of orthogonalization

*Hypothetical scenario: large A, *many* wanted eigenpairs*

- Assume A has size $10M$
- ... and you want to compute 50,000 eigenvalues/vectors (huge for numerical analysits, not for physicists) ...
- ... in the lower part of the spectrum - or the middle.
- By (any) standard method you will need to orthogonalize at least 50K vectors of size 10M. Then:
  - Space needed: $\approx 4 \times 10^{12}$ b = 4TB *just for the basis*
  - Orthogonalization cost: 5×10^{16} = 50 PetaOPS.
  - At step k, each orthogonalization step costs $\approx 4kn$
  - This is $\approx 200,000n$ for k close to $50,000$.

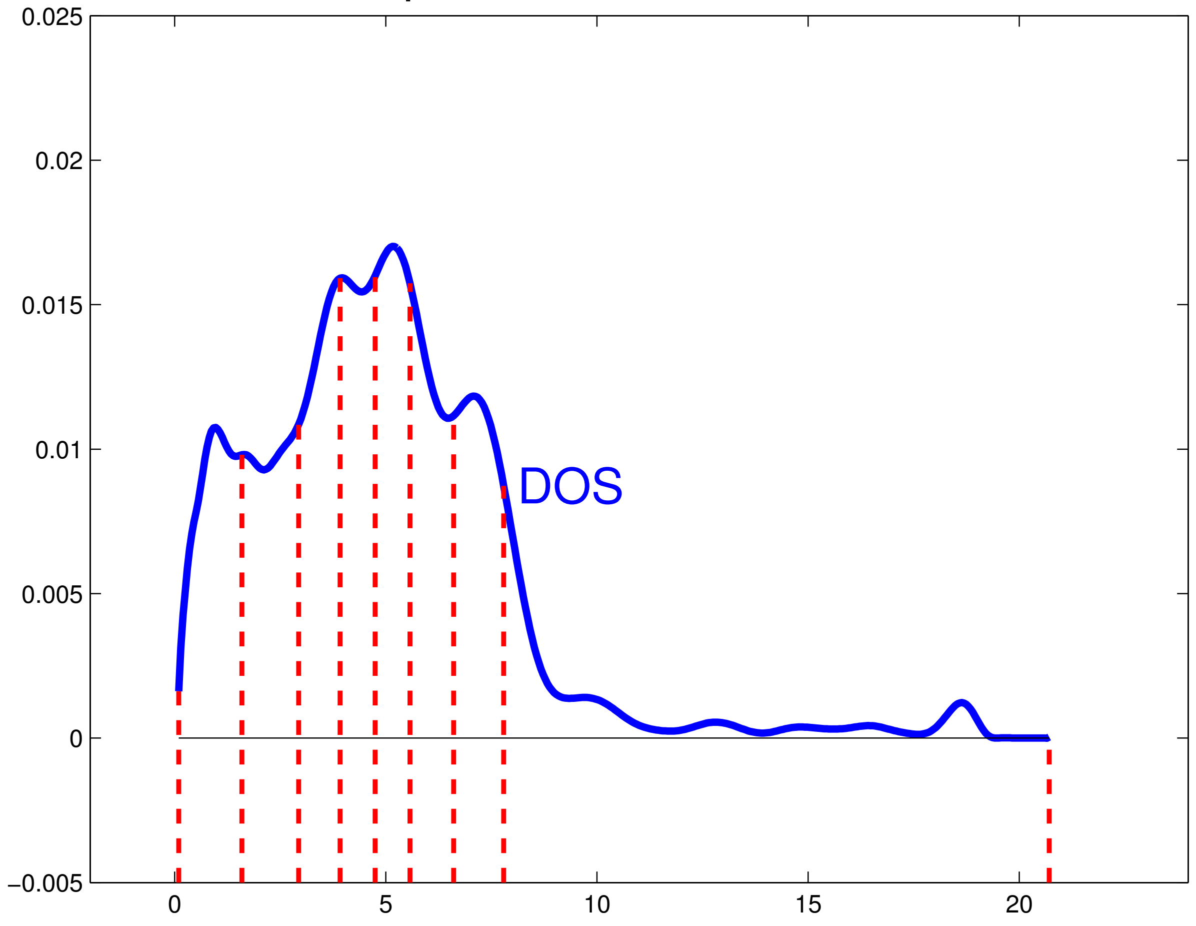
How do I slice a spectrum?

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]*
- ● $\boxed{\text{Interesting instance of a tool from physics used in linear algebra.}}$
- ➤ 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..

Slice spectrum into 8 with the DOS



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

A little 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
- $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$ 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

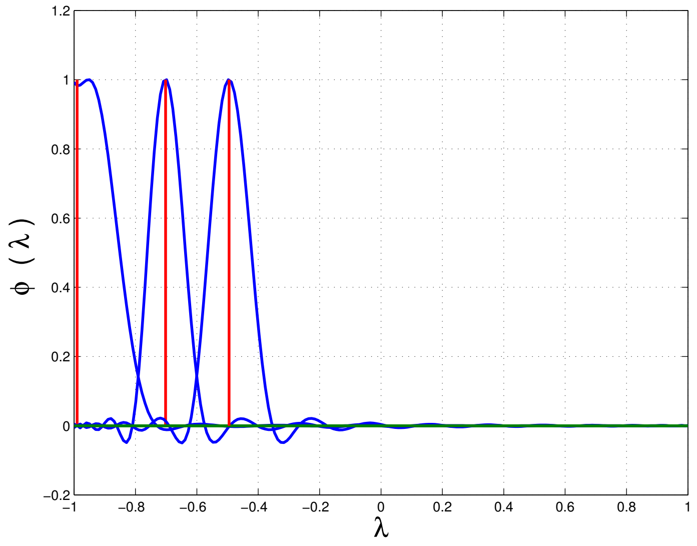
- Used by Chemists to calculate the DOS – see Silver and Röder'94 , Wang '94, Drabold-Sankey'93, + others
- Basic idea: expand DOS into Chebyshev polynomials
- Coefficients γ_k lead to evaluating Tr $(T_k(A))$
- Use trace estimators [discovered independently] to get traces
- Details skipped

Spectrum Slicing and the EVSL project

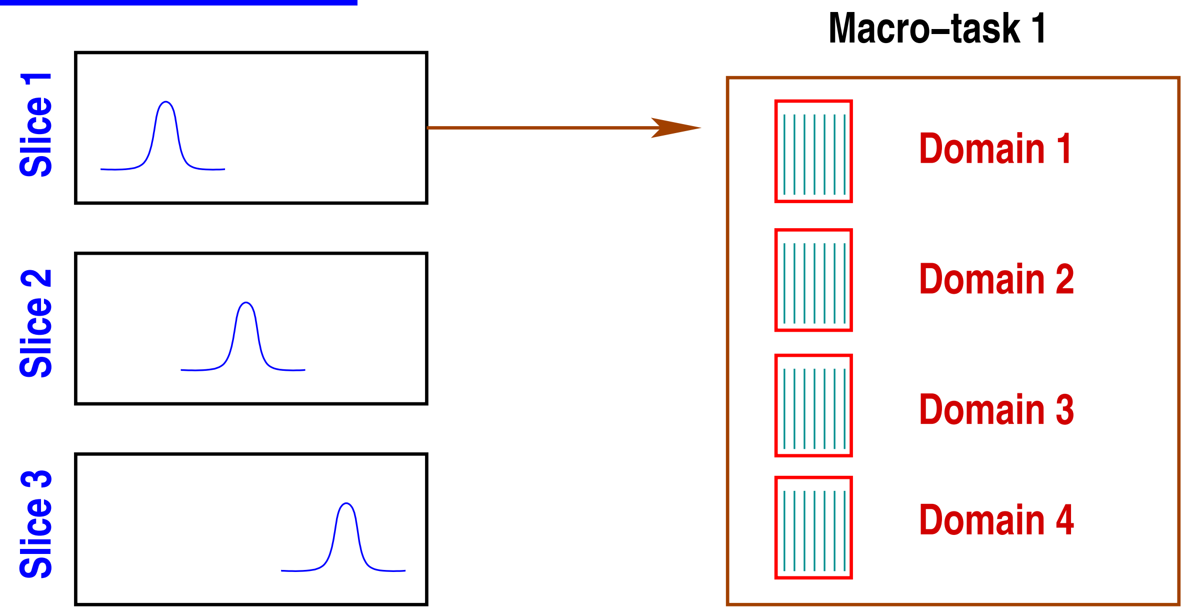
- EVSL uses polynomial and rational filters
- Each can be appealing in different situations.

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 $(60^3 \to n = 216,000)$ *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	$[a, b]$	$[\xi, \eta]$	$\nu_{[\xi,\eta]}$
$\mathbf{Ge_{87}H_{76}}$	$\mathbf{112,985}$	$\mathbf{7.9}M$	$[-1.21, 32.76]$	$[-0.64, -0.0053]$	212
$\mathbf{Ge_{99}H_{100}}$	$\mathbf{112,985}$	$\mathbf{8.5}M$	$[-1.22, 32.70]$	$[-0.65, -0.0096]$	250
$\mathbf{Si_{41}Ge_{41}H_{72}}$	$\mathbf{185,639}$	$\mathbf{15.0}M$	$[-1.12, 49.82]$	$[-0.64, -0.0028]$	218
$\mathbf{Si_{87}H_{76}}$	$\mathbf{240,369}$	$\mathbf{10.6}M$	$[-1.19, 43.07]$	$[-0.66, -0.0300]$	213
$\mathbf{Ga_{41}As_{41}H_{72}}$	$\mathbf{268,096}$	$\mathbf{18.5}M$	$[-1.25, 1301]$	$[-0.64, -0.0000]$	201

Numerical results for PARSEC matrices

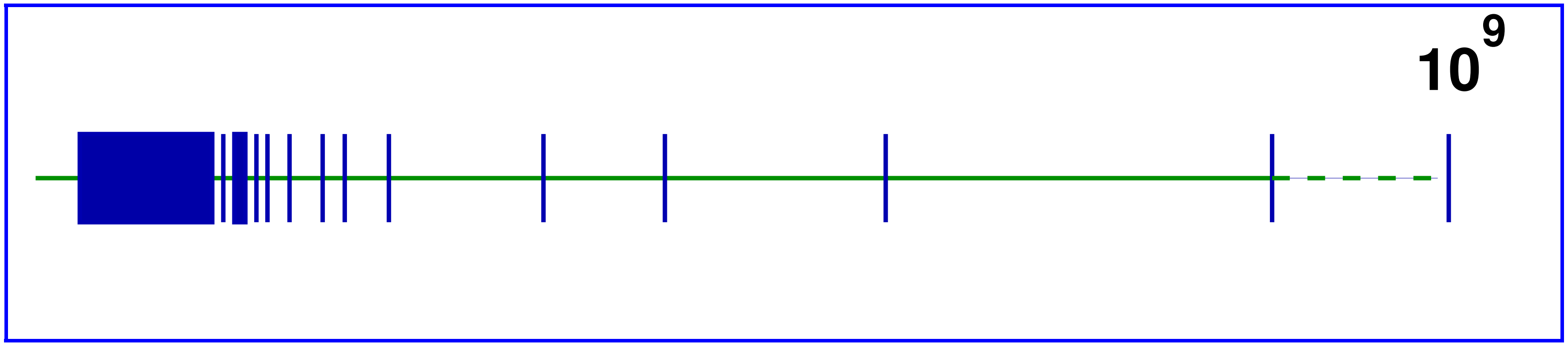
Matrix	deg	iter	matvec	CPU time (sec)		residual	
				matvec	total	max	avg
$\mathbf{Ge_{87}H_{76}}$	26	1431	37482	282.70	395.91	9.40×10^{-09}	2.55×10^{-10}
$\mathbf{Ge_{99}H_{100}}$	26	1615	42330	338.76	488.91	9.10×10^{-09}	2.26×10^{-10}
$\mathbf{Si_{41}Ge_{41}H_{72}}$	35	1420	50032	702.32	891.98	3.80×10^{-09}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

RATIONAL FILTERS

Why use rational filters?

** Joint work with Yuanzhe Xi

➤ Consider a spectrum like this one:



➤ Polynomial filtering utterly ineffective for this case

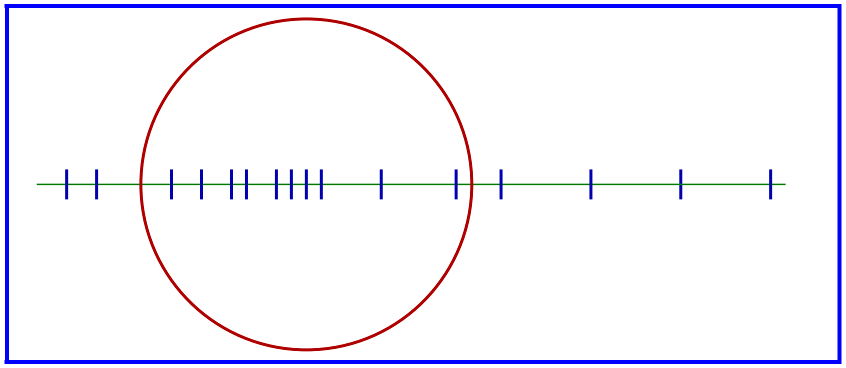
➤ Second issue: situation when Matrix-vector products are expensive

➤ Generalized eigenvalue problems.

- Alternative is to use rational filters: $\phi(z) = \sum_j \frac{\alpha_j}{z-\sigma_j}$

$\phi(A) = \sum_j \alpha_j (A - \sigma_j I)^{-1}$ $\rightarrow$ We now need to solve linear systems

- Tool: Cauchy integral representations of spectral projectors

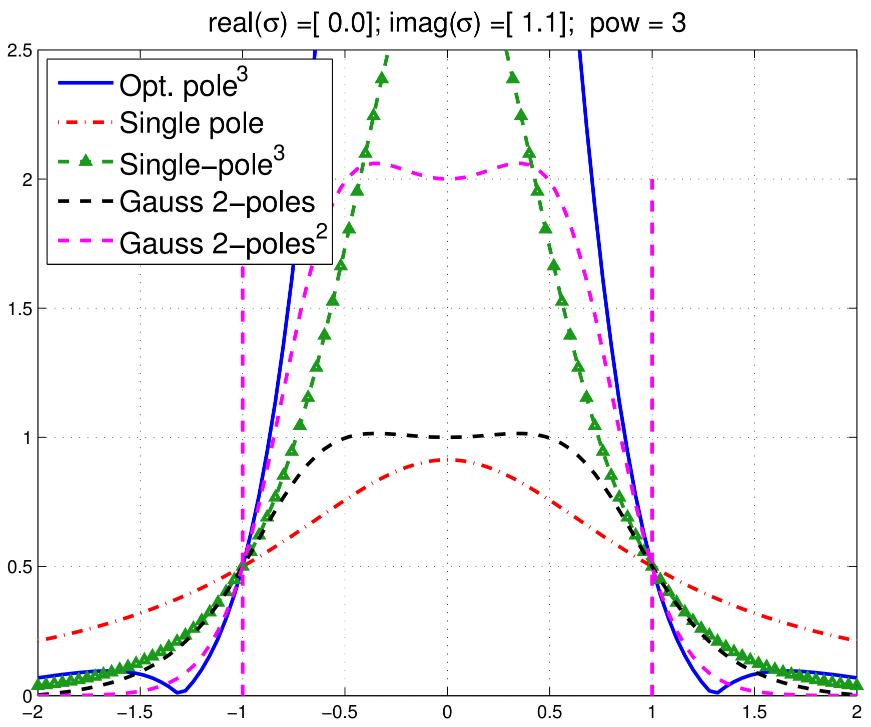


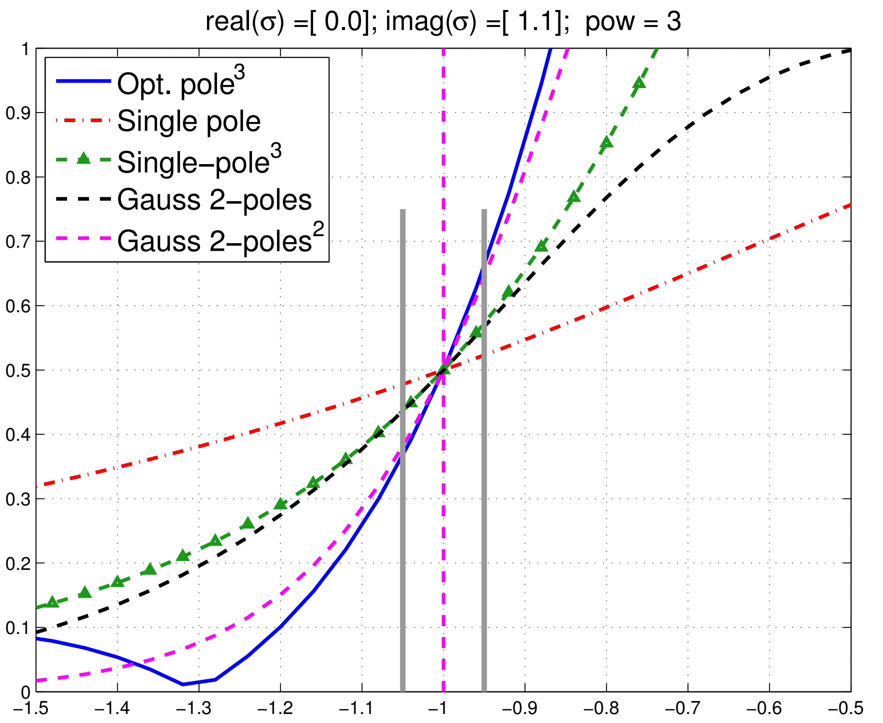
$$P = \frac{-1}{2i\pi} \int_\Gamma (A - sI)^{-1} ds$$

- 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



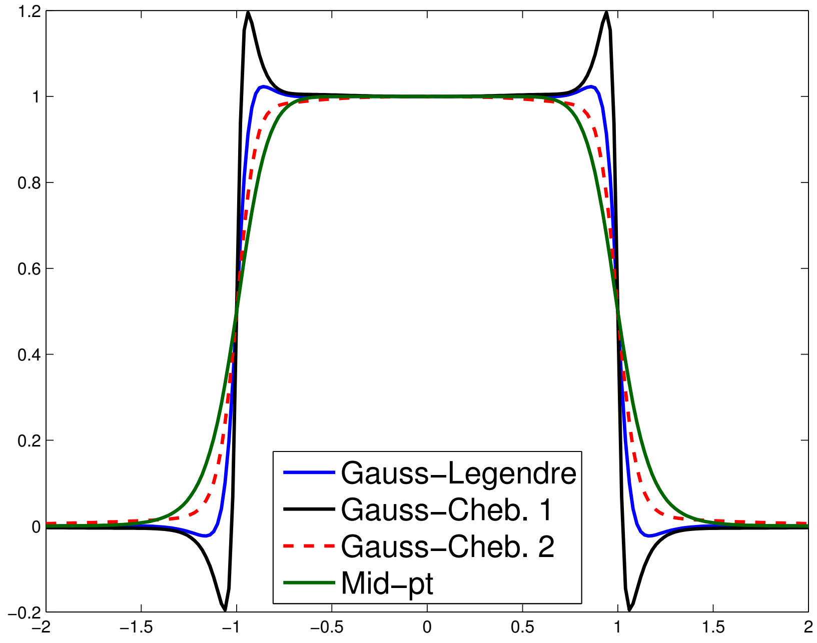


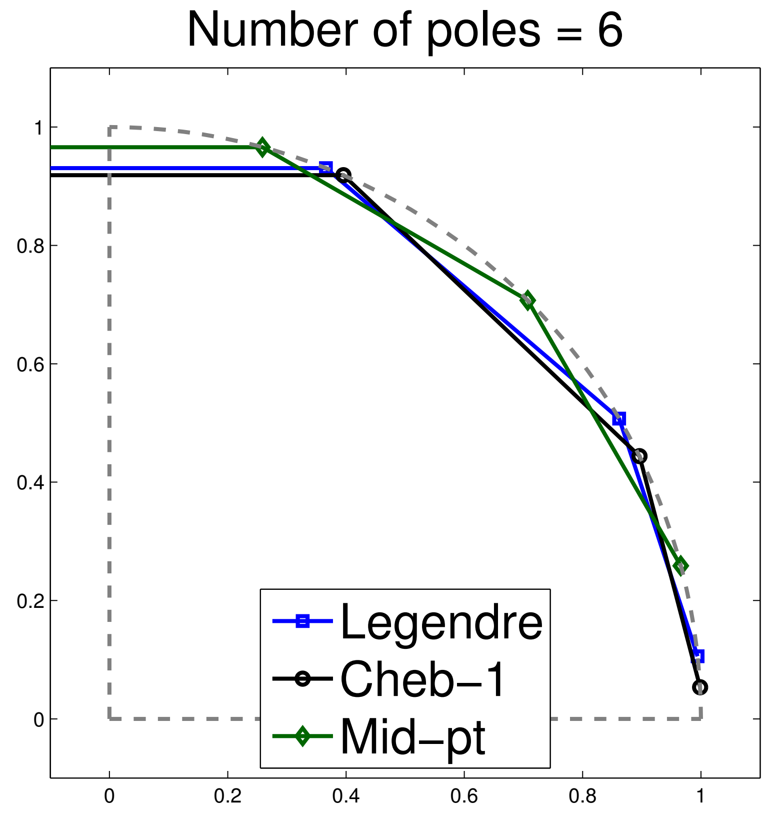
- Assume subspace iteration is used with above filters. Which filter will give better convergence?

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

The 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 $\mathbf{43 \times 53}$ grid. ($\boldsymbol{n = 2279}$)

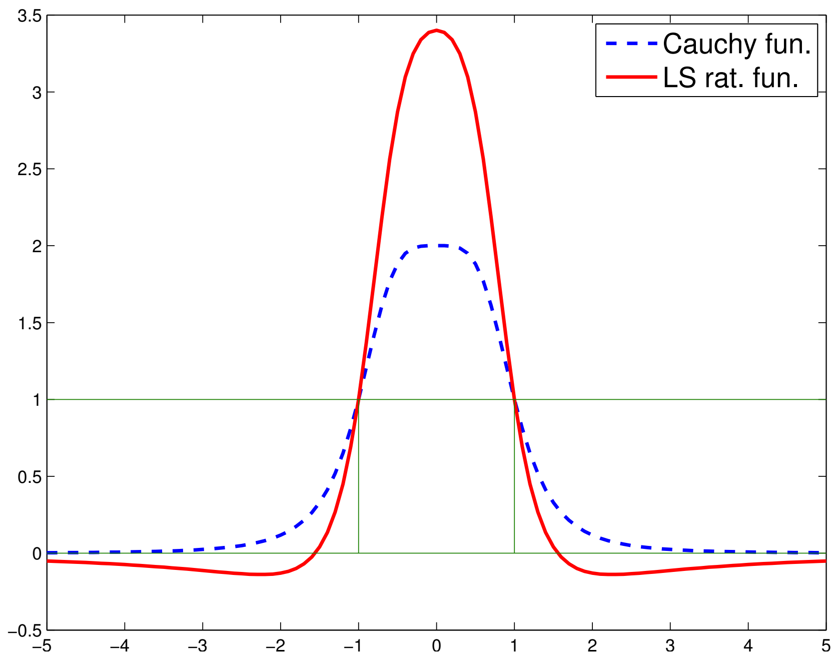
➤ Take 4 poles obtained from mid-point rule($\boldsymbol{N_c = 2}$ on each 1/2 plane)

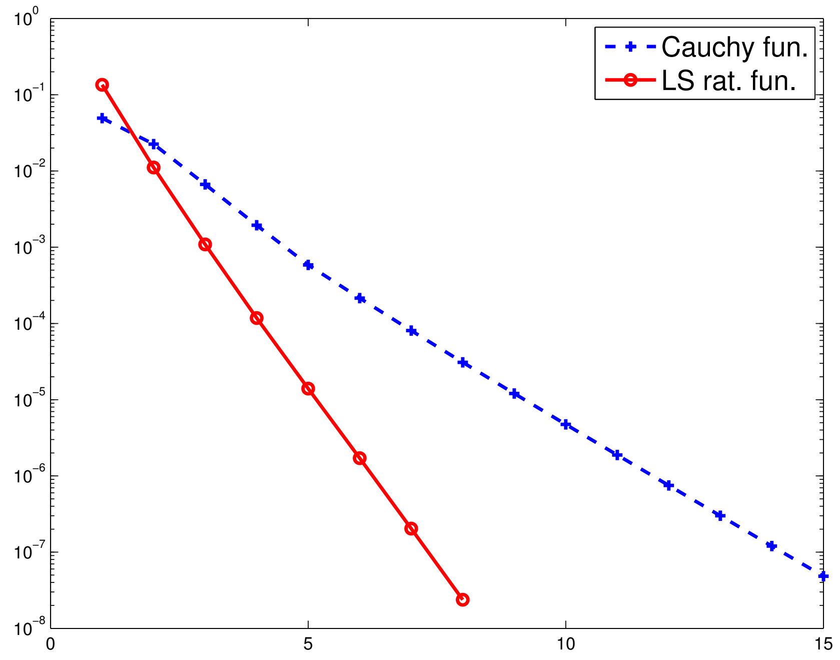
➤ Want: eigenvalues inside $\mathbf{[0,\ 0.2]}$. There are $\boldsymbol{nev = 31}$ of them.

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

➤ Use subspace of dim $\boldsymbol{nev + 6}$

➤ $\boldsymbol{\beta = 0.2}$





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

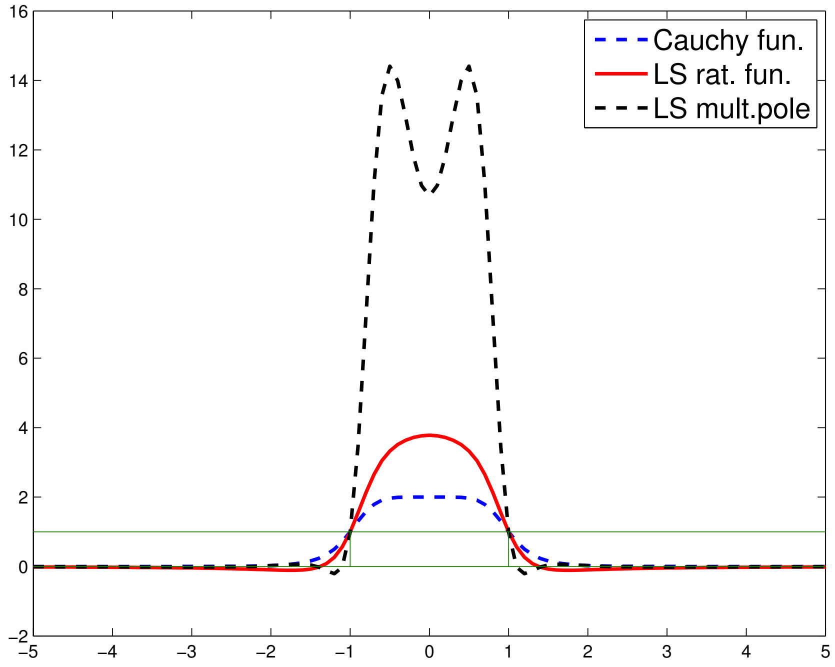
- Other advantages:

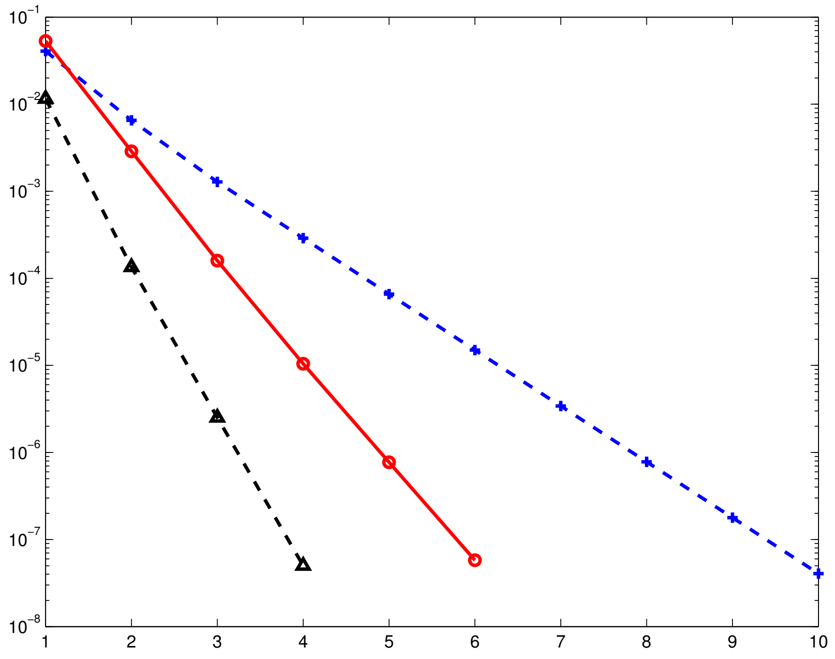
- Can select poles far away from real axis $\rightarrow$ faster iterative solvers [E. Di Napoli et al.]
- Very flexible – can be adapted to many situations
- Can use multiple poles (!)

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

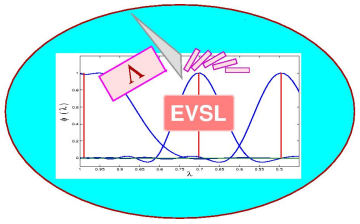
Better rational filters: Example

- Take same example as before 43×53 Laplacean
- Now take 6 poles [3×2 midpoint rule]
- Repeat each pole [double poles.]

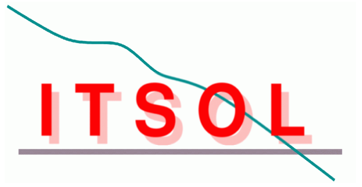




SOFTWARE



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.

Conclusion

- Polynomial Filtering appealing when # of eigenpairs to be computed is large and Matvecs are cheap
- May not work well for generalized eigenvalue problems
- Will not work well for spectra with large outliers.
- Alternative: Rational filtering
- Both approaches implemented in EVSL
- Current work: test this on the earth normal modes problem.
- `EVSL` code available here:

  `www.cs.umn.edu/~saad/software/EVSL`
- Fully parallel version (MPI) in the works