UNIVERSITY OF MINNESOTA TWIN CITIES

# Polynomial filtering for interior eigenvalue problems

*Yousef Saad*

*Department of Computer Science and Engineering*

*University of Minnesota*

*SIAM CSE*

*Boston – March 1, 2013*

## *First:*

- Joint work with: Haw-ren Fang
- Grady Schoefield and Jim Chelikowsky [UT Austin] [windowing into PARSEC]
- Work supported in part by NSF (to 2012) and now by DOE

## *Introduction*

*Q:* How do you compute eigenvalues in the middle of the spectrum of a large Hermitian matrix?

*A:* Method of choice: Shift and invert + some projection process (Lanczos, subspace iteration..)

*Main steps:*

1) Select a shift (or sequence of shifts) $\sigma$;
2) Factor $A - \sigma I$: $A - \sigma I = LDL^T$
3) Apply Lanczos algorithm to $(A - \sigma I)^{-1}$

- Solves with $A - \sigma I$ carried out using factorization
- Limitation: factorization

Q: What if factoring $A$ is too expensive (e.g., Large 3-D similation)?

A: Obvious answer: Use iterative solvers ...

➤ However: systems are highly indefinite $\rightarrow$ Wont work too well.

➤ Digression: Still lots of work to do in iterative solution methods for highly indefinite linear systems

➤ Other common characteristic:

Need a very large number of eigenvalues and eigenvectors

➤ Applications: Excited states in quantum physics: TDDFT, GW, ...

➤ Or just plain Density Functional Theory (DFT)

➤ Number of wanted eigenvectors is equal to number of occupied states – [ == the number of valence electrons in DFT]

➤ An example: in real-space code (PARSEC), you can have a Hamiltonian of size a few Millions, and number of ev's in the tens of thousands.

## *Polynomial filtered Lanczos*

- Possible solution: Use Lanczos with polynomial filtering.
- 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.

- Most important factor is 2.
- Main rationale of polynomial filtering : reduce the cost of orthogonalization
- Important application: compute the spectrum by pieces ['spectrum slicing' a term coined by B. Parlett]

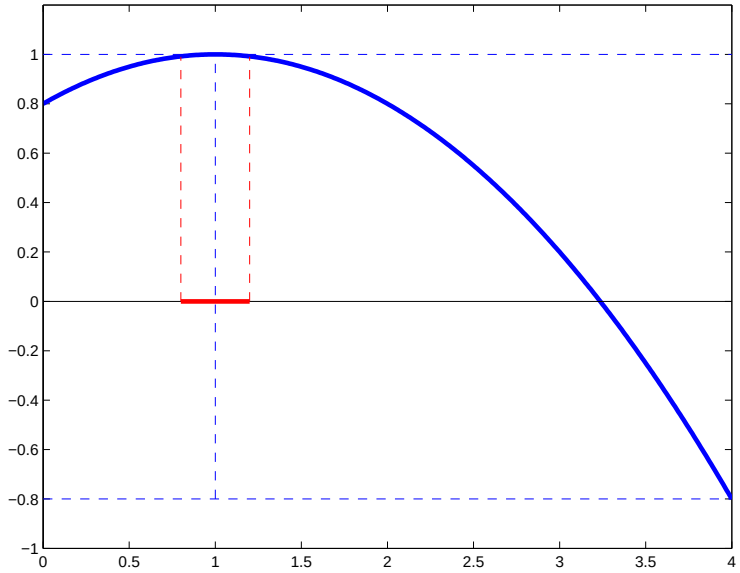
## *Introduction: What is filtered Lanczos?*

➤ In short: we just replace $(A - \sigma I)^{-1}$ in S.I. Lanczos by $p_k(A)$ where $p_k(t) =$ polynomial of degree $k$

➤ 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 $\longrightarrow$

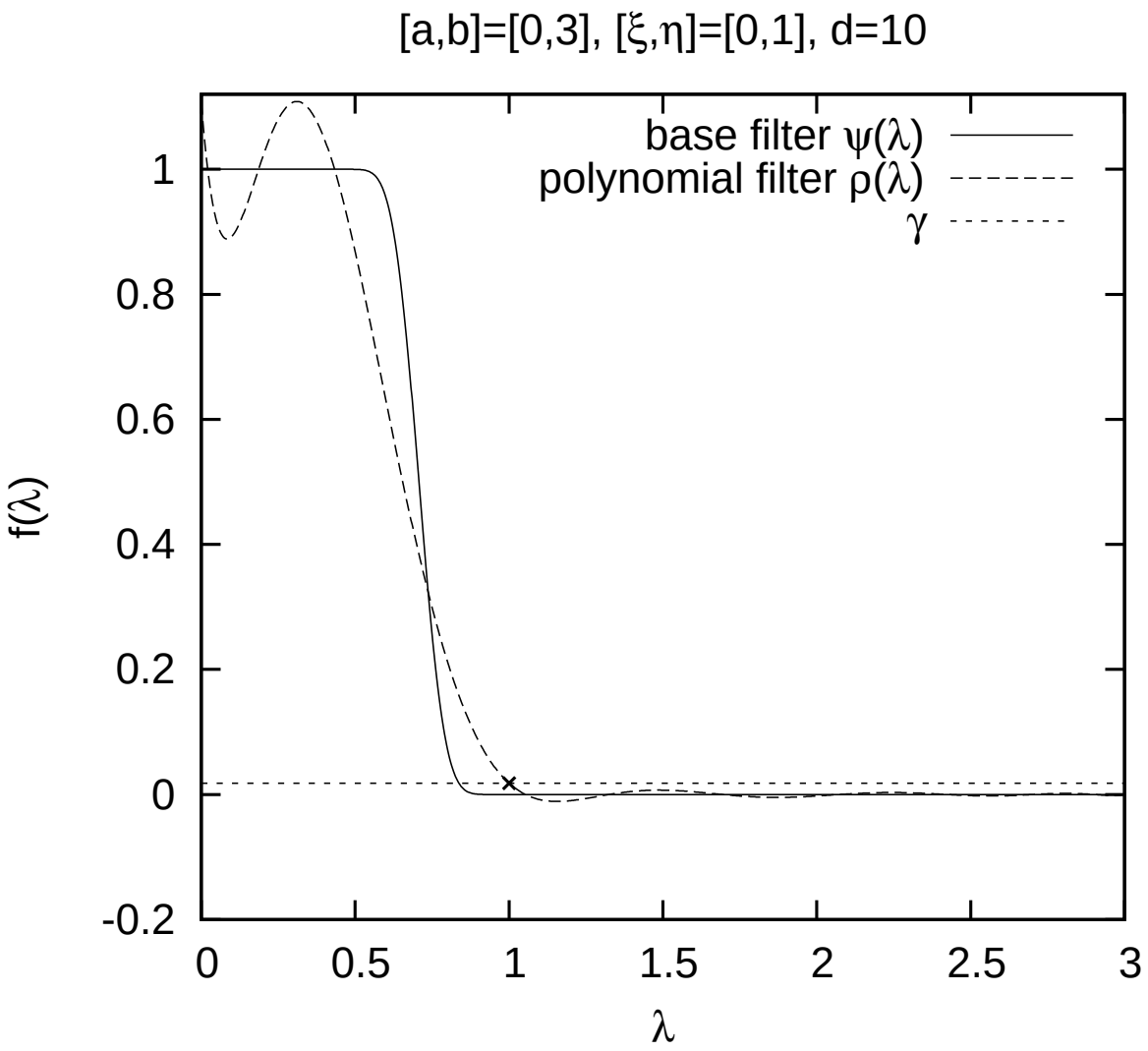



➤ Use Lanczos with $B = p_2(A)$.

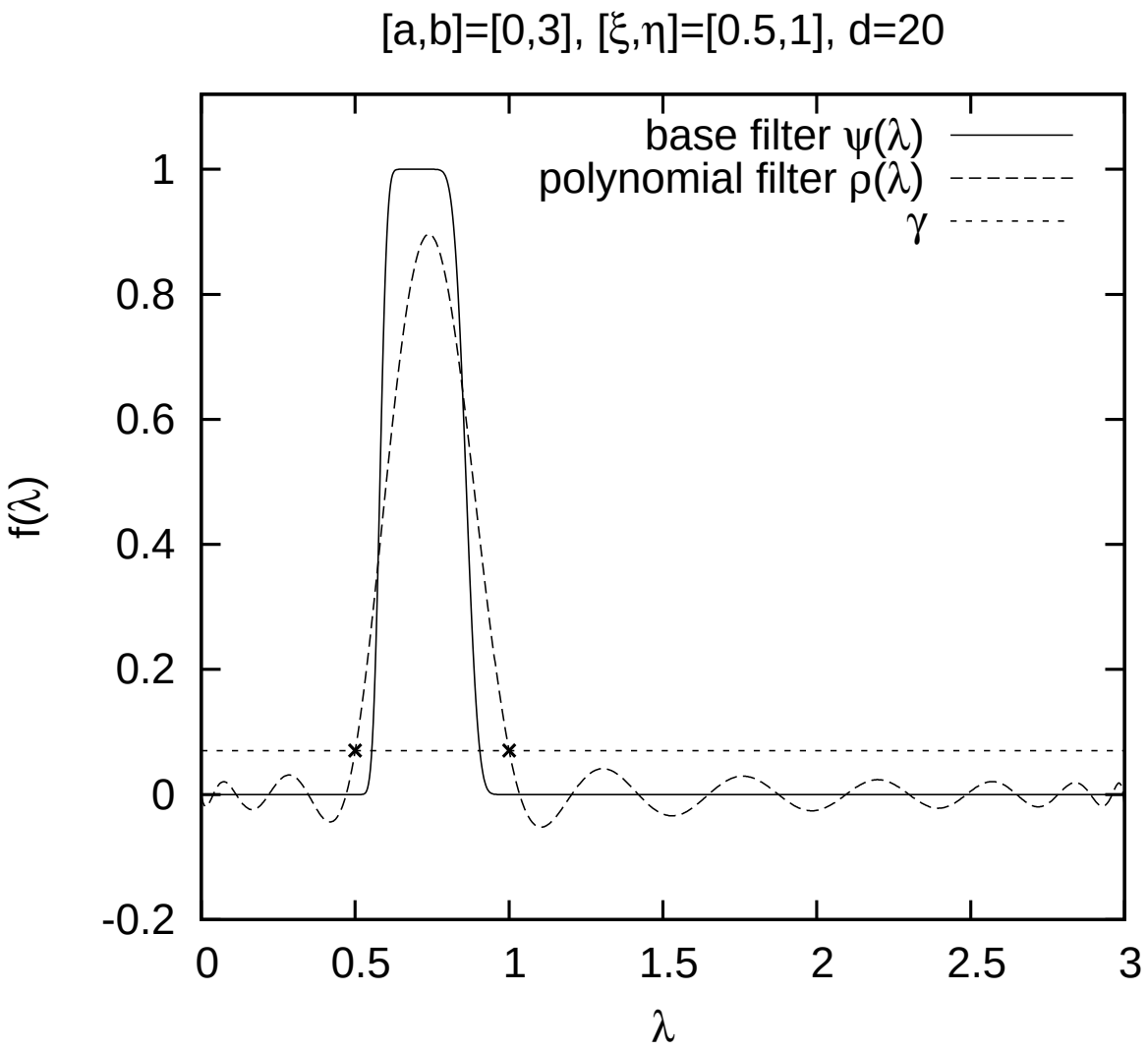
➤ Eigenvalues near $\sigma$ become the dominant ones – so Lanczos will work – but...

➤ ... they are now poorly separated $\rightarrow$ slow convergence.

# *A low-pass filter*

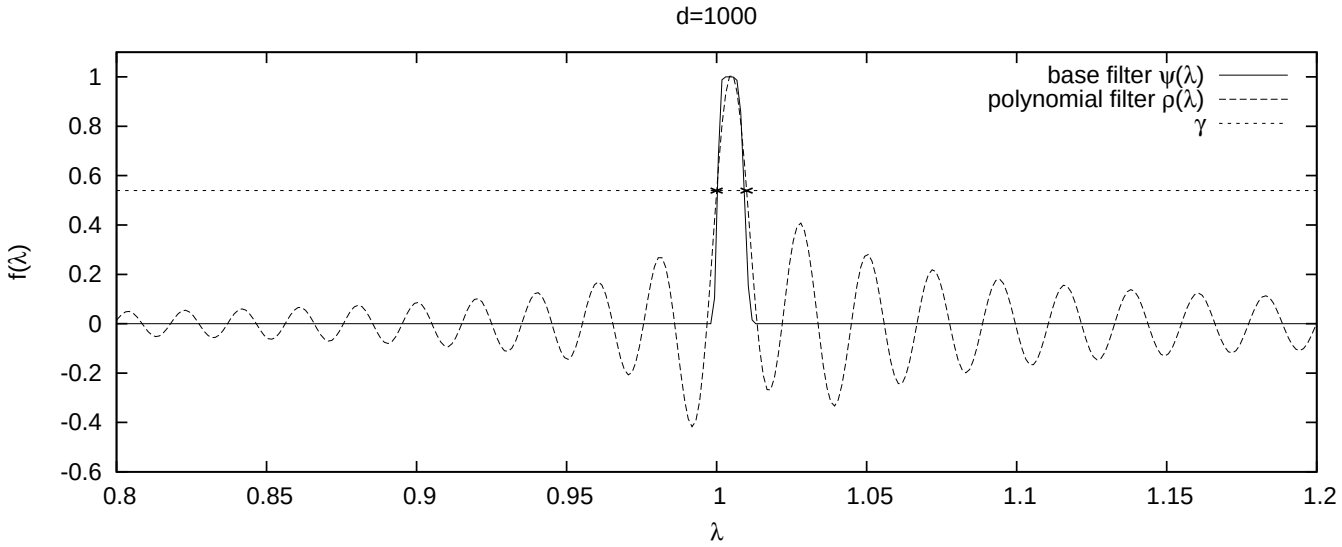


# *A mid-pass filter*

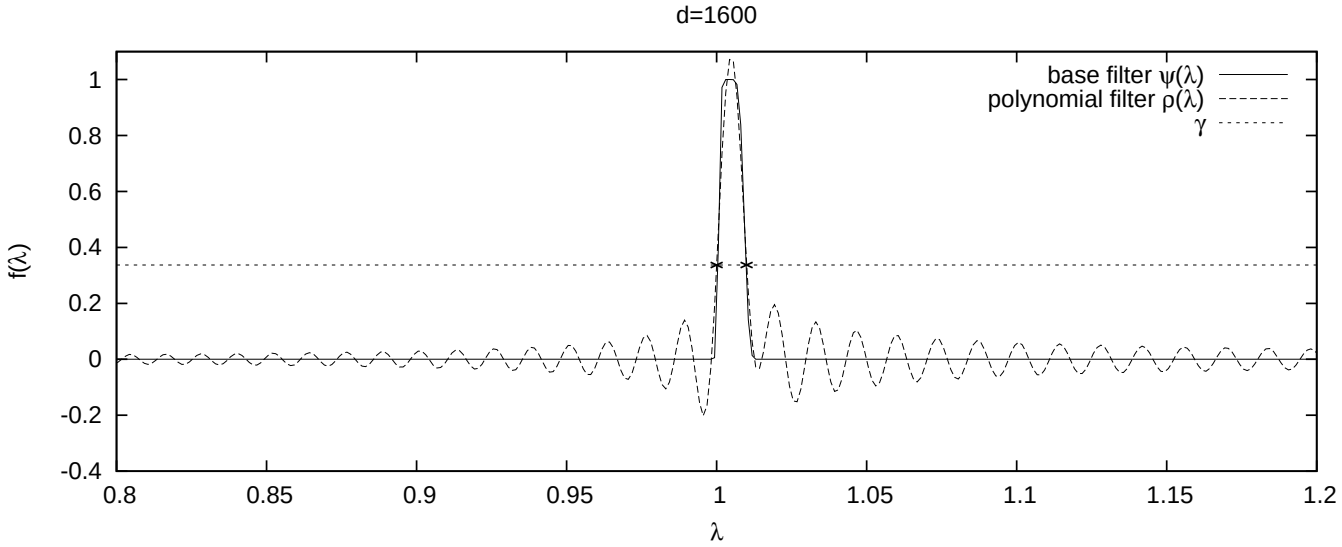


# *Misconception: High degree polynomials are bad*

Degree 1000 (zoom)




Degree 1600 (zoom)



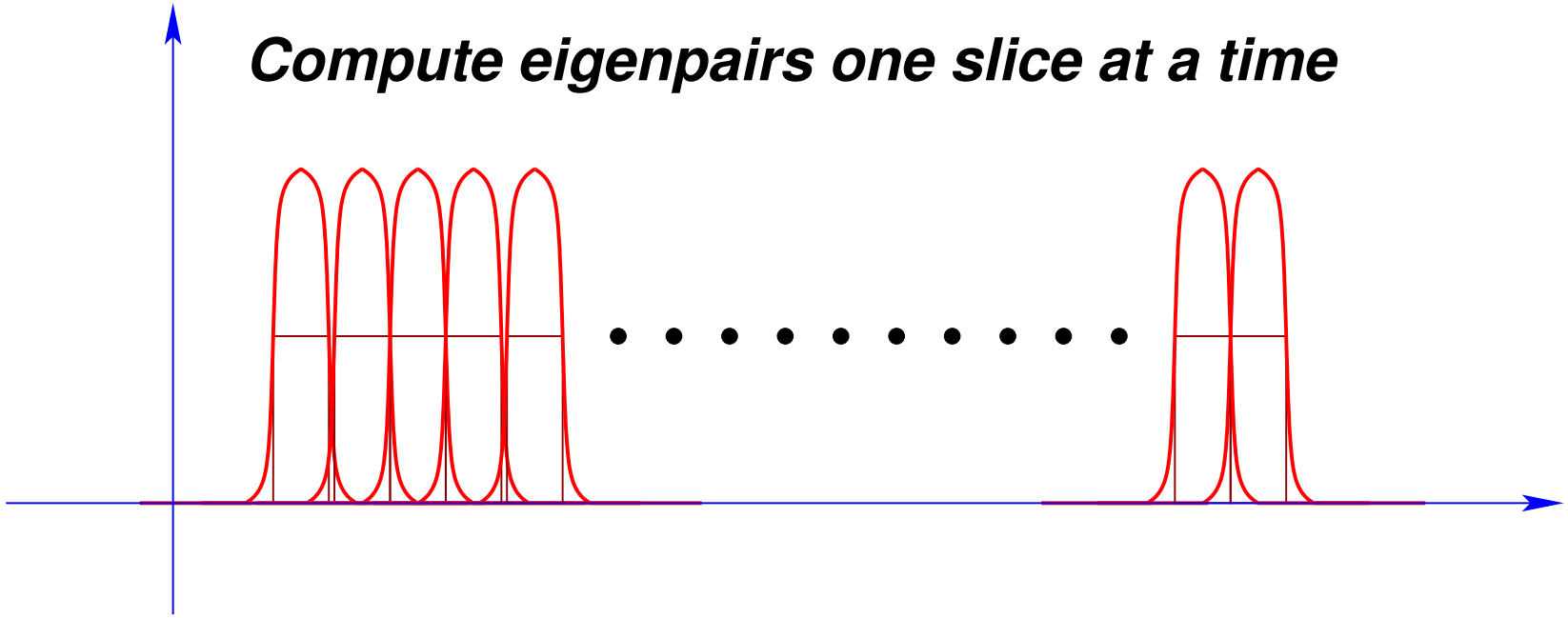
## *Hypothetical scenario: large $A$, zillions of wanted e-values*

- Assume $A$ has size $10M$ (Not huge by todays standard)
- ... 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 methods you will need to orthogonalize at least 50K vectors of size 10M –
- Space is an issue: $4 \times 10^{12}$ bytes = 4TB of mem *just for the basis*
- Orthogonalization is also an issue: $5 \times 10^{16}$ = 50 PetaOPS.
- Toward the end, at step $k$, each orthogonalization step costs about $\approx 4kn \approx 200,000n$ for $k$ close to $50,000$.

## *The alternative: 'Spectrum slicing' or 'windowing'*

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

- Idea: Get the spectrum by 'slices' or 'windows'
- Can get a few hundreds or thousands of vectors at a time.




- Deceivingly simple looking idea.
- Issues:
  - Deal with interfaces : duplicate/missing eigenvalues
  - Window size [need estimate of eigenvalues]
  - polynomial degree

## *Spectrum slicing in PARSEC*

- Being implemented in our code:

Pseudopotential Algorithm for Real-Space Electronic Calculations (PARSEC)

- See :

'*A Spectrum Slicing Method for the Kohn-Sham Problem*', G. Schofield, J. R. Chelikowsky and YS, Computer Physics Comm., vol 183, pp. 487-505.

- Refer to this paper for details on windowing and 'initial proof of concept'

## *Computing the polynomials: Jackson-Chebyshev*

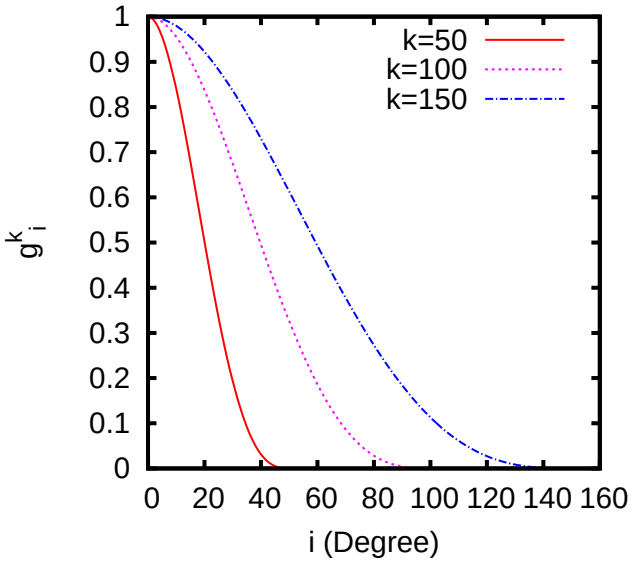
Chebyshev-Jackson approximation of a function $f$:

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

$$\gamma_i = \frac{2-\delta_{i0}}{\pi} \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx \quad \delta_{i0} = \text{ Kronecker symbol}$$

The $g_i^k$'s attenuate higher order terms in the sum.

Attenuation coefficient $g_i^k$ for $k$=50,100,150 $\rightarrow$



Let $\alpha_k = \dfrac{\pi}{k+2}$, then :

$$g_i^k = \frac{\left(1 - \frac{i}{k+2}\right) \sin(\alpha_k) \cos(i\alpha_k) + \frac{1}{k+2} \cos(\alpha_k) \sin(i\alpha_k)}{\sin(\alpha_k)}$$

See

'*Electronic structure calculations in plane-wave codes without diagonalization.*' Laurent O. Jay, Hanchul Kim, YS, and James R. Chelikowsky. *Computer Physics Communications*, 118:21–30, 1999.

## The expansion coefficients $\gamma_i$

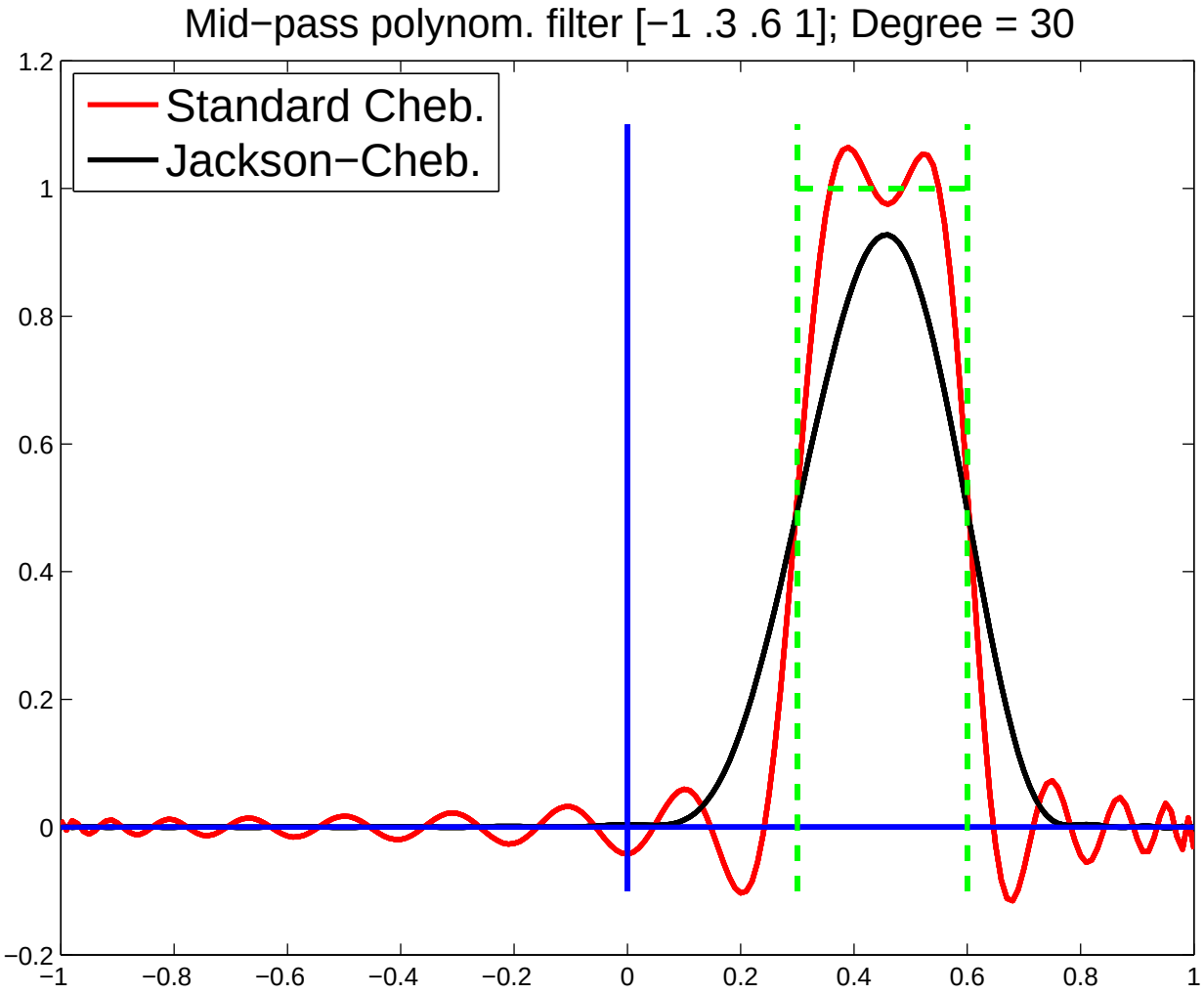
When $f(x)$ is a step function on $[a, b] \subseteq [-1\ 1]$:

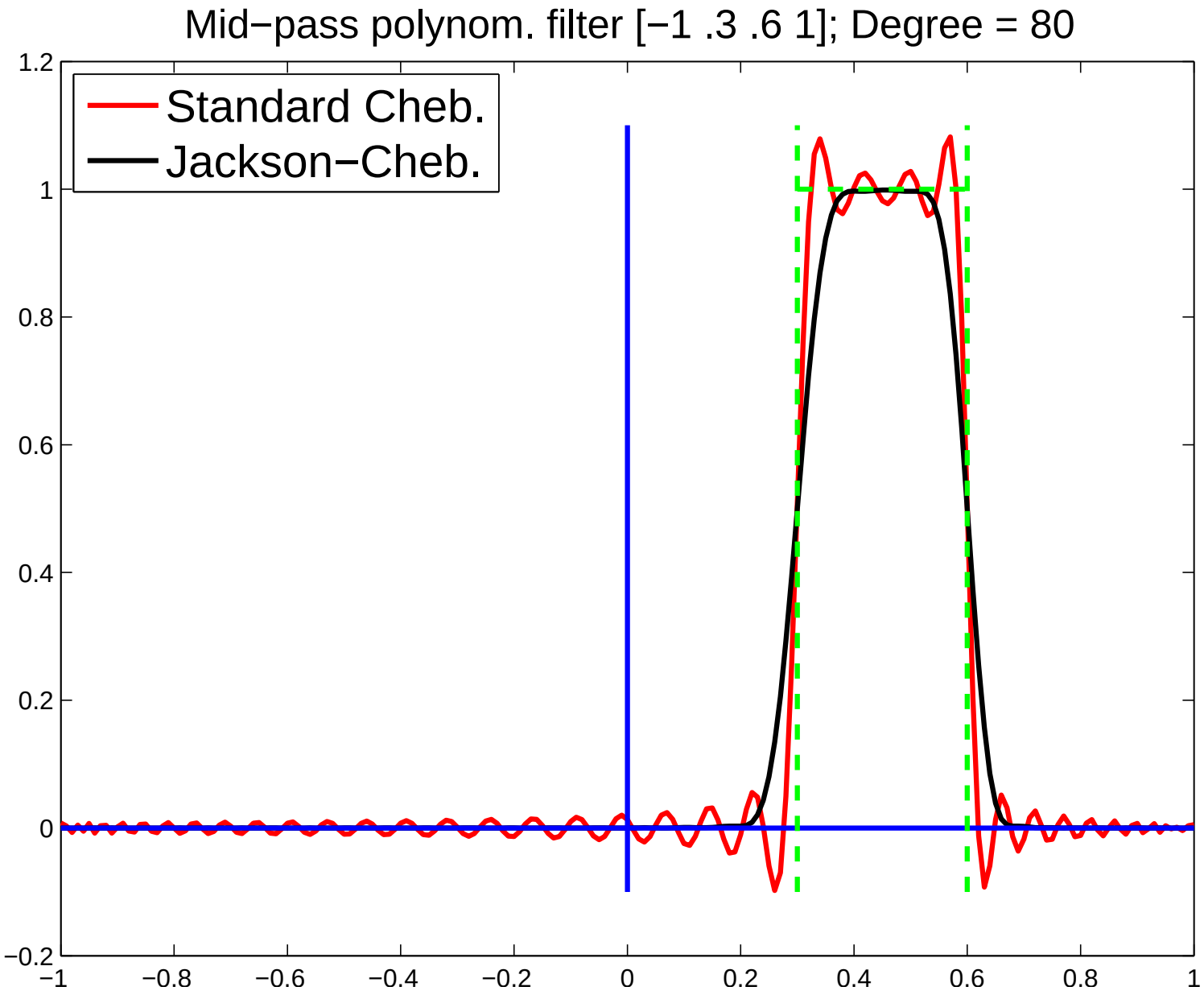
$$\gamma_i = \begin{cases} \dfrac{1}{\pi}\left(\arccos(a) - \arccos(b)\right) : i = 0 \\ \dfrac{2}{\pi}\left(\dfrac{\sin(i\arccos(a)) - \sin(i\arccos(b))}{i}\right) : i > 0 \end{cases}$$

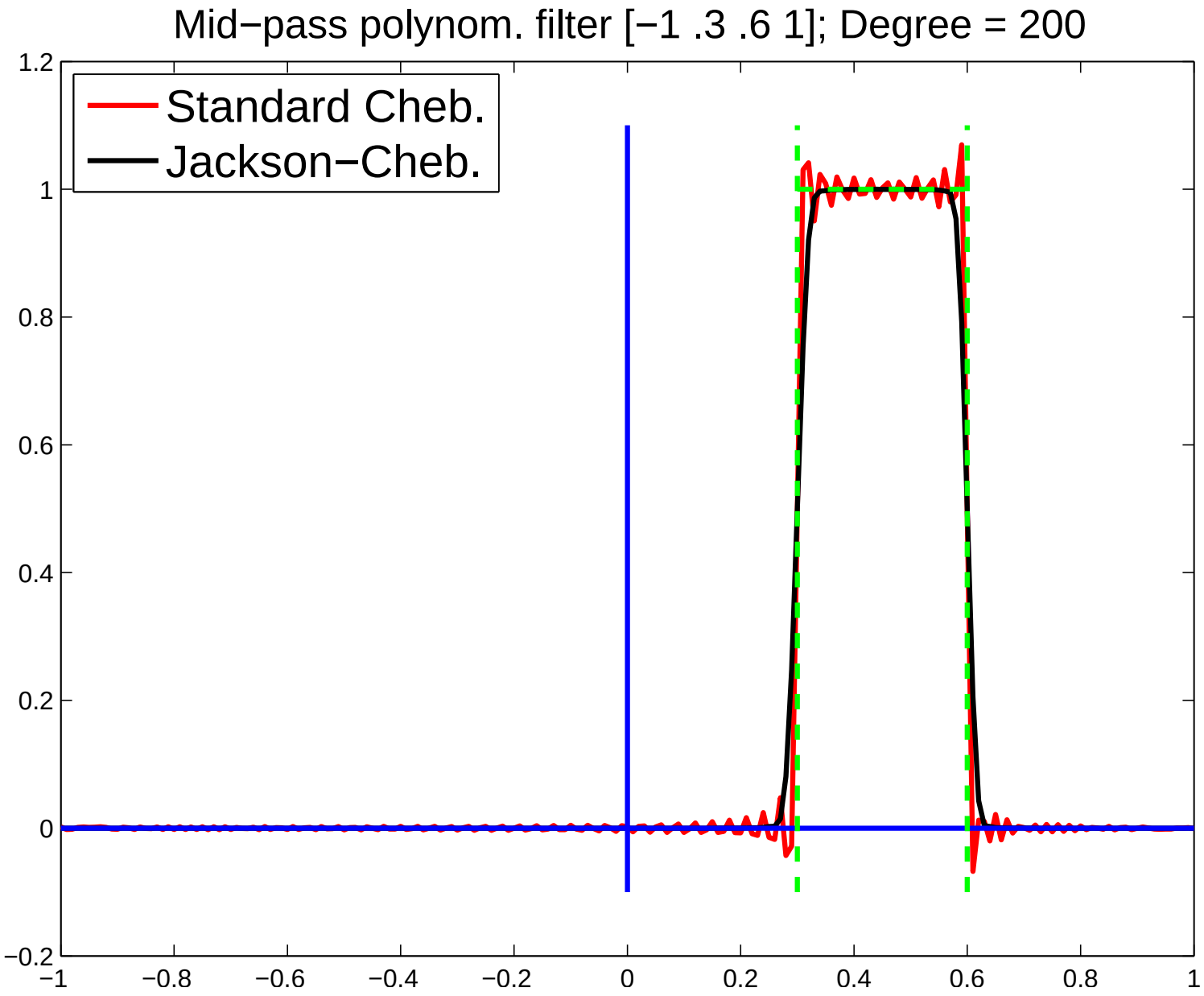
- A few examples follow –

# *Computing the polynomials: Jackson-Chebyshev*

- Polynomials of degree 30 for $[a,\ b] = [.3,\ .6]$



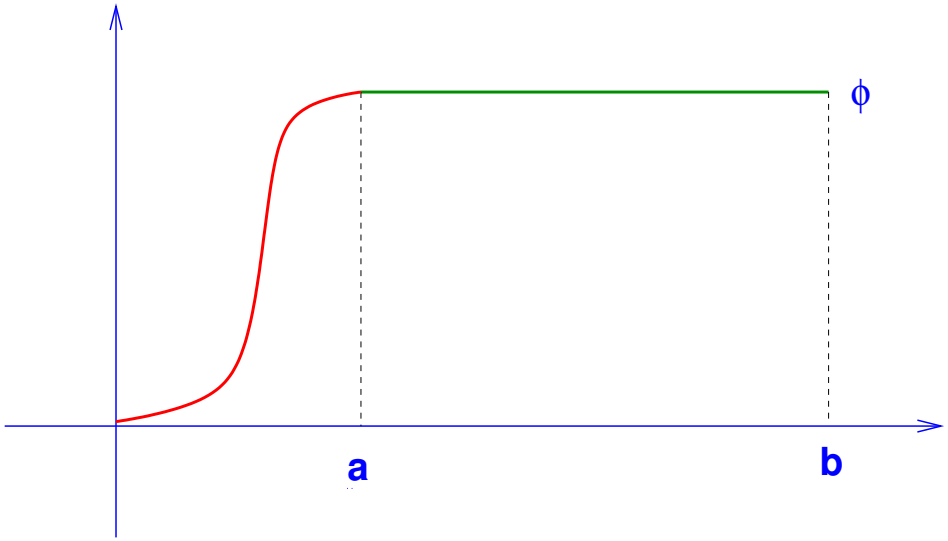

Mid−pass polynom. filter [−1 .3 .6 1]; Degree = 80
Standard Cheb.
Jackson−Cheb.
1.2
1
0.8
0.6
0.4
0.2
0
−0.2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1


Mid−pass polynom. filter [−1 .3 .6 1]; Degree = 200
Standard Cheb.
Jackson−Cheb.
1.2
1
0.8
0.6
0.4
0.2
0
−0.2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

# *How to get the polynomial filter? Second approach*

*Idea:*

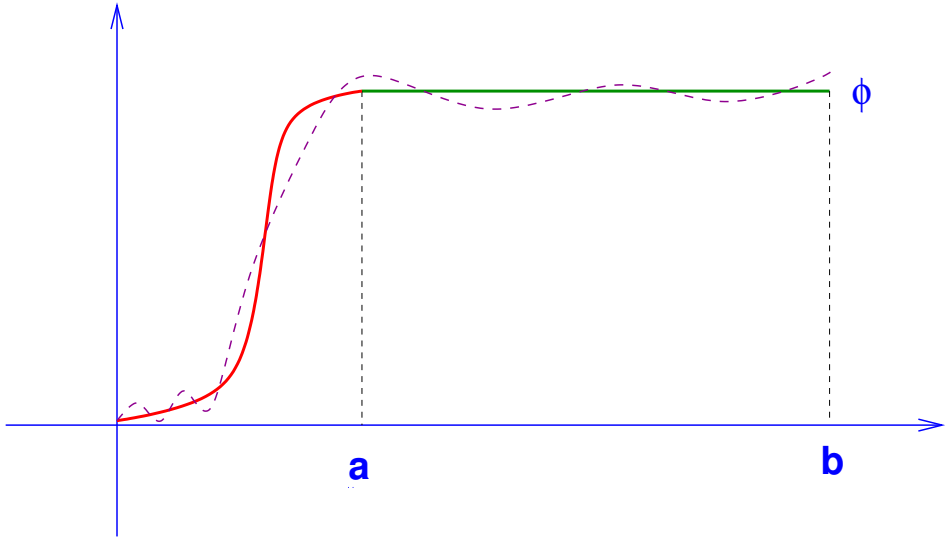
- First select an "ideal filter"
- e.g., a piecewise polynomial function




➤ For example $\phi$ = Hermite interpolating pol. in [0,a], and $\phi = 1$ in [a, b]

➤ Referred to as the 'Base filter'

- Then approximate base filter by degree $k$ polynomial in a least-squares sense.
- Can do this without numerical integration




*Main advantage:* Extremely flexible.

*Method:* Build a sequence of polynomials $\phi_k$ which approximate the ideal PP filter $\phi$, in the $L_2$ sense.

➤ Again 2 implementations

➤ Define $\phi_k \equiv$ the least-squares polynomial approximation to $\phi$:

$$\phi_k(t) = \sum_{j=1}^{k} \langle \phi, \mathcal{P}_j \rangle \mathcal{P}_j(t),$$

where $\{\mathcal{P}_j\}$ is a basis of polynomials that is orthonormal for some $L_2$ inner-product.

➤ Method 1: Use Stieljes procedure to computing orthogonal polynomials

## ALGORITHM : 1. Stieljes

1. $\mathcal{P}_0 \equiv 0,$
2. $\beta_1 = \|\mathcal{S}_0\|_{\langle \quad \rangle},$
3. $\mathcal{P}_1(t) = \frac{1}{\beta_1}\mathcal{S}_0(t),$
4. *For* $j = 2, \ldots, m$ *Do*
5. $\quad \alpha_j = \langle t\,\mathcal{P}_j, \mathcal{P}_j \rangle,$
6. $\quad \mathcal{S}_j(t) = t\,\mathcal{P}_j(t) - \alpha_j \mathcal{P}_j(t) - \beta_j \mathcal{P}_{j-1}(t),$
7. $\quad \beta_{j+1} = \|\mathcal{S}_j\|_{\langle \quad \rangle},$
8. $\quad \mathcal{P}_{j+1}(t) = \frac{1}{\beta_{j+1}}\mathcal{S}_j(t).$
9. *EndDo*

## *Computation of Stieljes coefficients*

*Problem:* To compute the scalars $\alpha_j$ and $\beta_{j+1}$, of 3-term recurrence (Stieljes) + the expansion coefficients $\gamma_j$. Need to avoid numerical integration.

*Solution:* define orthogonal polynomials over two (or more) disjoint intervals – see similar work YS'83:

YS, '*Iterative solution of indefinite symmetric systems by methods using orthogonal polynomials over two disjoint intervals*', SIAM Journal on Numerical Analysis, 20 (1983), pp. 784–811.

E. KOKIOPOULOU AND YS, '*Polynomial Filtering in Latent Semantic Indexing for Information Retrieval*', in *Proc. ACM-SIGIR Conference on research and development in information retrieval, Sheffield, UK*, (2004)

- Let large interval be $[0, b]$ – should contain $\Lambda(B)$
- Assume 2 subintervals. On subinterval $[a_{l-1}, a_l]$, $l = 1, 2$ define the inner-product $\langle \psi_1, \psi_2 \rangle_{a_{l-1}, a_l}$ by

$$\langle \psi_1, \psi_2 \rangle_{a_{l-1}, a_l} = \int_{a_{l-1}}^{a_l} \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t - a_{l-1})(a_l - t)}}\, dt.$$

- Then define the inner product on $[0, b]$ by

$$\langle \psi_1, \psi_2 \rangle = \int_0^a \frac{\psi_1(t)\psi_2(t)}{\sqrt{t(a - t)}}\, dt + \rho \int_a^b \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t - a)(b - t)}}\, dt.$$

- To avoid numerical integration, use a basis of Chebyshev polynomials on interval [YS'83]

## *Mehod 2 : Filtered CG/CR - like polynomial iterations*

*Want:* a CG-like (or CR-like) algorithms for which the inderlying residual polynomial or solution polynomial are Least-squares filter polynomials

- Seek $s$ to minimize $\|\phi(\lambda) - \lambda\, s(\lambda)\|_w$ with respect to a certain norm $\|.\|_w$.
- Equivalently, minimize $\|(1-\phi) - (1-\lambda\, s(\lambda))\|_w$ over all polynomials $s$ of degree $\leq k$.
- Focus on second view-point (residual polynomial)
- goal is to make $r(\lambda) \equiv 1 - \lambda s(\lambda)$ close to $1 - \phi$.

## *Recall: Conjugate Residual Algorithm*

ALGORITHM : 2. *Conjugate Residual Algorithm*

1. *Compute* $r_0 := b - Ax_0$, $p_0 := r_0$
2. *For* $j = 0, 1, \ldots,$ *until convergence Do:*
3. $\quad \alpha_j := (r_j, Ar_j)/(Ap_j, Ap_j)$
4. $\quad x_{j+1} := x_j + \alpha_j p_j$
5. $\quad r_{j+1} := r_j - \alpha_j Ap_j$
6. $\quad \beta_j := (r_{j+1}, Ar_{j+1})/(r_j, Ar_j)$
7. $\quad p_{j+1} := r_{j+1} + \beta_j p_j$
8. $\quad$ *Compute* $Ap_{j+1} = Ar_{j+1} + \beta_j Ap_j$
9. *EndDo*

➤ Think in terms of polynomial iteration

## ALGORITHM : 3. *Filtered CR polynomial Iteration*

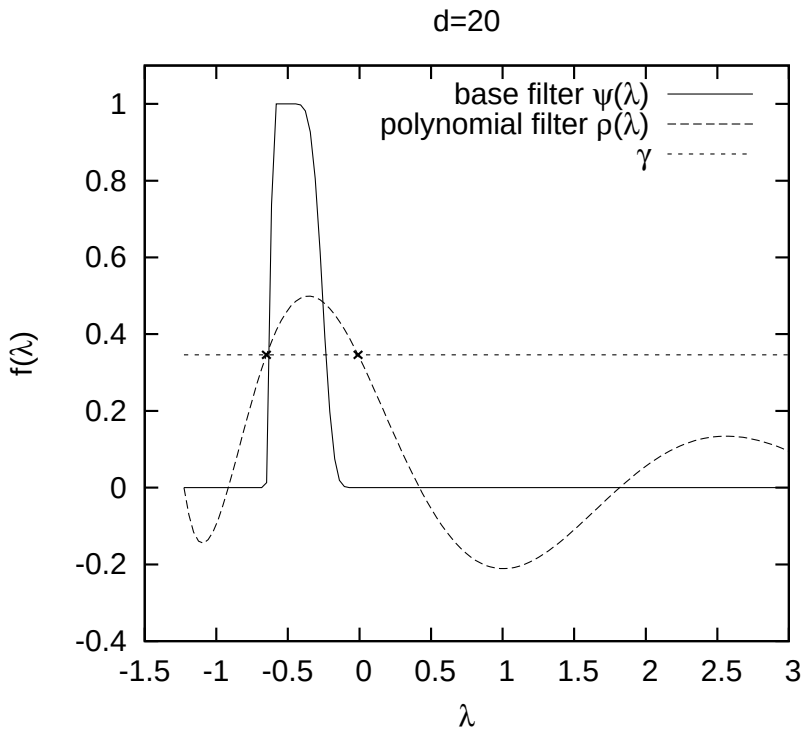
1. *Compute* $\tilde{r}_0 := b - Ax_0$, $p_0 := \tilde{r}_0$ $\boxed{\pi_0 = \rho_0 = 1}$
1.a. $\boxed{\text{Compute } \lambda\pi_0}$
2. *For* $j = 0, 1, \ldots,$ *until convergence Do:*
3. $\tilde{\alpha}_j := < \rho_j, \lambda\rho_j >_w \; / \; < \lambda\pi_j, \lambda\pi_j >_w$
3.a. $\alpha_j := \tilde{\alpha}_j - < 1 - \phi, \lambda\pi_j >_w \; / \; < \lambda\pi_j, \lambda\pi_j >_w$
4. $x_{j+1} := x_j + \alpha_j p_j$
5. $\tilde{r}_{j+1} := \tilde{r}_j - \tilde{\alpha}_j A p_j$ $\boxed{\rho_{j+1} = \rho_j - \tilde{\alpha}_j \lambda\pi_j}$
6. $\beta_j := < \rho_{j+1}, \lambda\rho_{j+1} >_w \; / \; < \rho_j, \lambda\rho_j >_w$
7. $p_{j+1} := r_{j+1} + \beta_j p_j$ $\boxed{\pi_{j+1} := \rho_{j+1} + \beta_j \pi_j}$
8. $\boxed{\text{Compute } \lambda\pi_{j+1}}$
9. *EndDo*

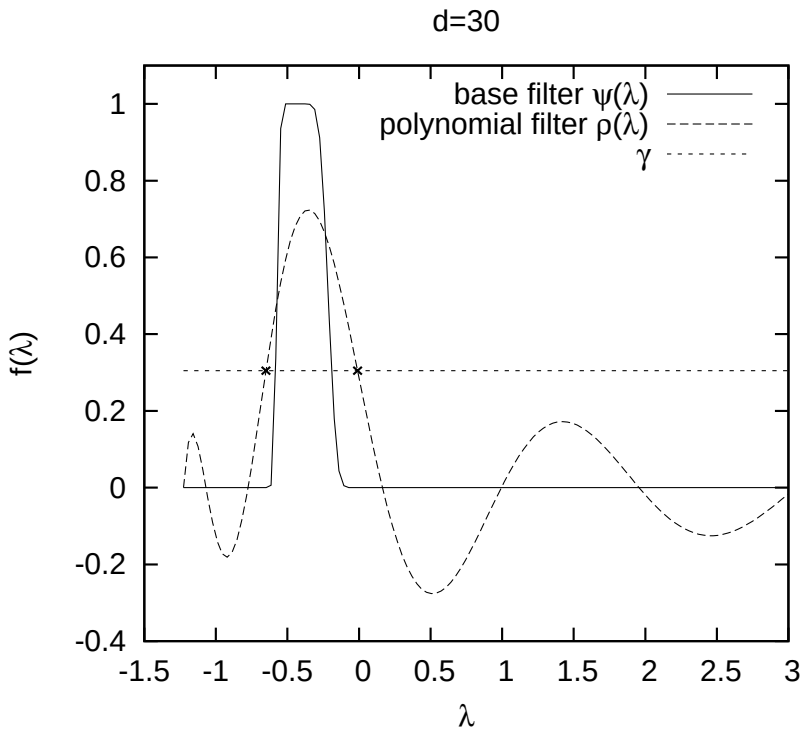
➤ All polynomials expressed in Chebyshev basis - cost of algorithm is negligible [ $O(k^2)$ for deg. $k$. ]

# *A few mid-pass filters of various degrees*

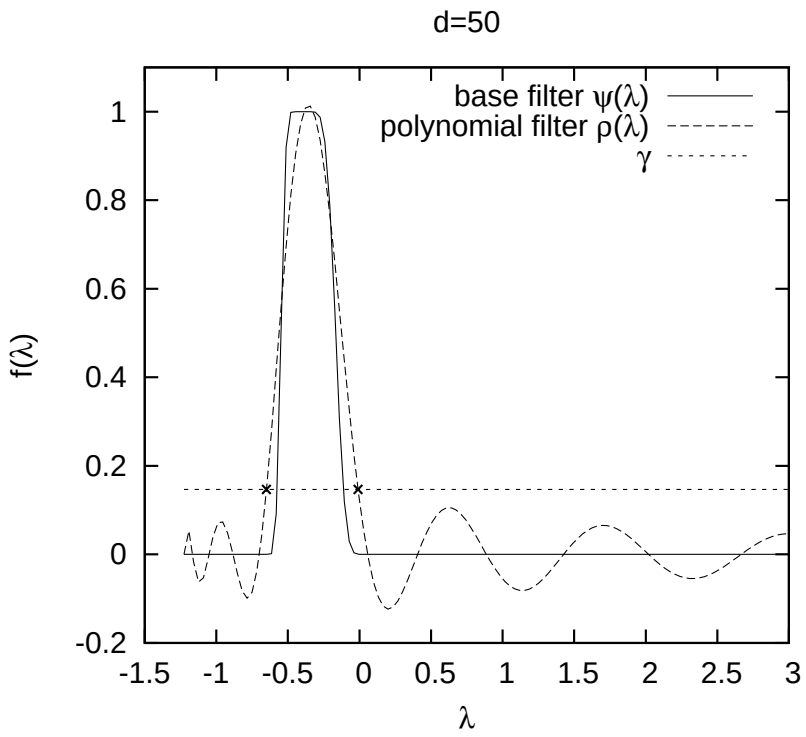
Four examples of middle-pass filters $\psi(\lambda)$ and their polynomial approximations $\rho(\lambda)$.

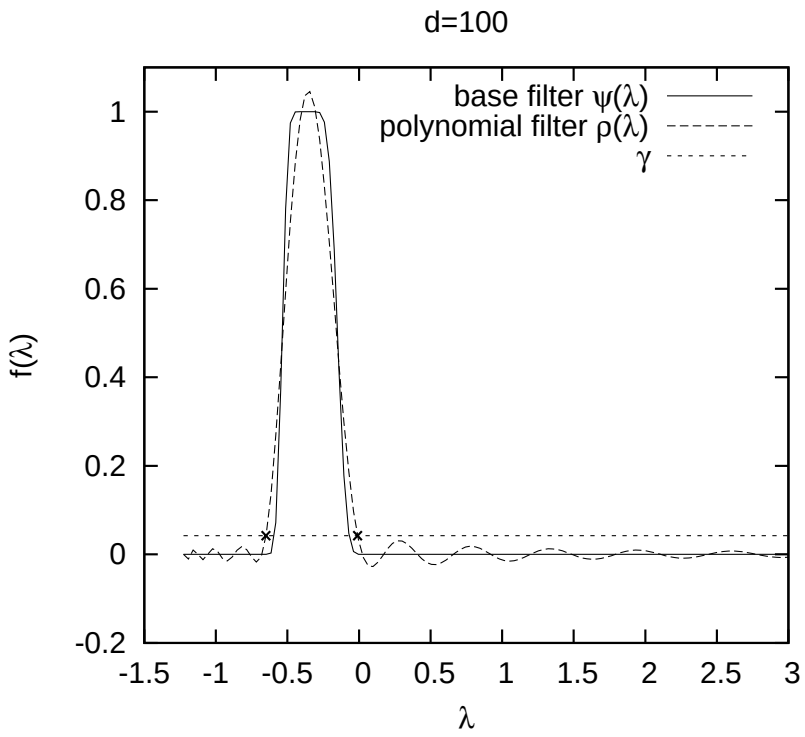
- Degrees 20 and 30



# Degrees 50 and 100

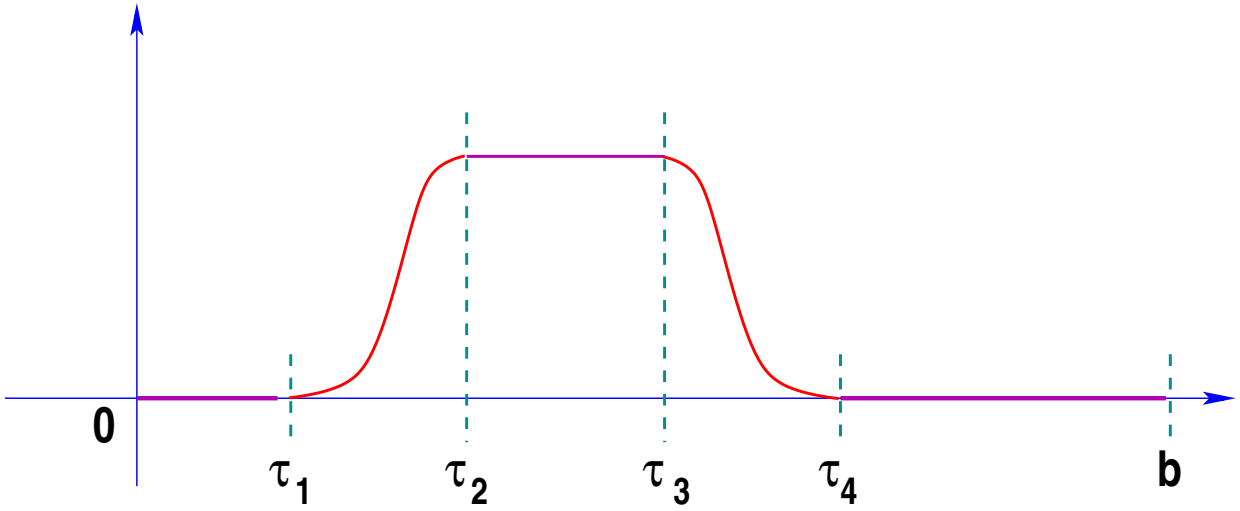


## *Base Filter to build a Mid-Pass filter polynomial*

We partition $[0, b]$ into five sub-intervals,

$$[0, b] = [0, 1][\tau_1, \tau_2] \cup [\tau_2, \tau_3] \cup [\tau_3, \tau_4] \cup [\tau_4, b]$$




- Set: $\psi(t) = 0$ in $[0, \tau_1] \cup [\tau_4, b]$ and $\psi(t) = 1$ in $[\tau_2, \tau_3]$
- Use standard Hermite interpolation to get 'brigde' functions in $[\tau_1, \tau_2]$ and $[\tau_3, \tau_4]$

## *References*


'*A Filtered Lanczos Procedure for Extreme and Interior Eigenvalue Problems*', H. R. Fang and YS, SISC 34(4) A2220-2246 (2012). For details on window-less implementation (one slice) + code

'*Computation of Large Invariant Subspaces Using Polynomial Filtered Lanczos Iterations with Applications in Density Functional Theory*', C. Bekas and E. Kokiopoulou and YS, SIMAX 30(1), 397-418 (2008).

'*Filtered Conjugate Residual-type Algorithms with Applications*', YS; SIMAX 28 pp. 845-870 (2006)

## *Tests – Test matrices*

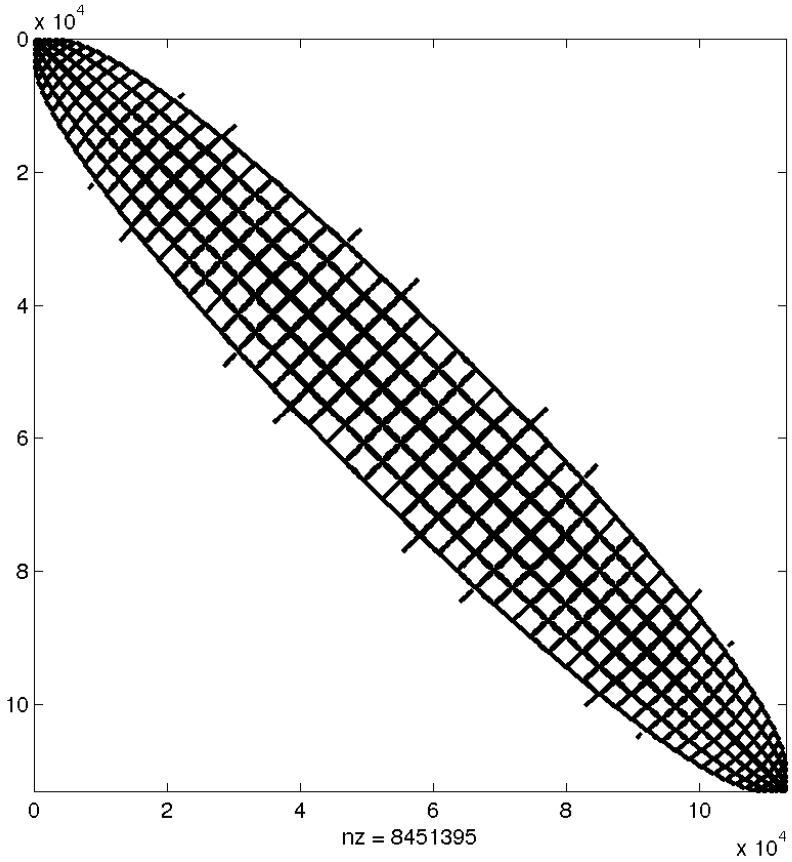
➤ Experiments performed in sequential mode: on two dual-core AMD Opteron(tm) Processors 2214 @ 2.2GHz and 16GB memory.

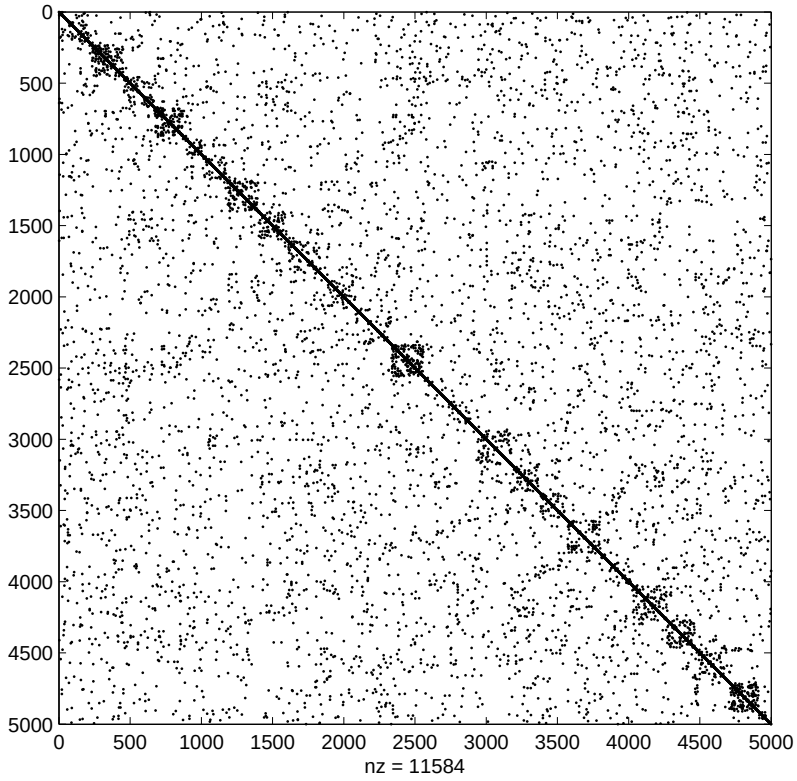
*Test matrices:*

* Five Hamiltonians from electronic structure calculations,

* An integer matrix named `Andrews`, and

* A discretized Laplacian (FD)


x 10^4
0
2
4
6
8
10
0
2
4
6
8
10
nz = 8451395
x 10^4


0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
nz = 11584

# Matrix characteristics

| matrix | $n$ | $nnz$ | $\frac{nnz}{n}$ | full eigen-range $[a, b]$ | Fermi $n_0$ |
|---|---|---|---|---|---|
| GE87H76 | 112,985 | 7,892,195 | 69.85 | $[-1.2140, 32.764]$ | 212 |
| Ge99H100 | 112,985 | 8,451,395 | 74.80 | $[-1.2264, 32.703]$ | 248 |
| SI41Ge41H72 | 185,639 | 15,011,265 | 80.86 | $[-1.2135, 49.818]$ | 200 |
| Si87H76 | 240,369 | 10,661,631 | 44.36 | $[-1.1963, 43.074]$ | 212 |
| Ga41As41H72 | 268,096 | 18,488,476 | 68.96 | $[-1.2501, 1300.9]$ | 200 |
| `Andrews` | 60,000 | 760,154 | 12.67 | $[0, 36.485]$ | N/A |
| `Laplacian` | 1,000,000 | 6,940,000 | 6.94 | $[0.00290, 11.997]$ | N/A |

# Experimental set-up

| matrix | eigen-interval $[\xi, \eta]$ | # eig in $[\xi, \eta]$ | # eig in $[a, \eta]$ | $\frac{\eta-\xi}{b-a}$ | $\frac{\eta-a}{b-a}$ |
|---|---|---|---|---|---|
| GE87H76 | $[-0.645, -0.0053]$ | 212 | 318 | 0.0188 | 0.0356 |
| Ge99H100 | $[-0.65, -0.0096]$ | 250 | 372 | 0.0189 | 0.0359 |
| SI41Ge41H72 | $[-0.64, -0.00282]$ | 218 | 318 | 0.0125 | 0.0237 |
| Si87H76 | $[-0.66, -0.33]$ | 212 | 317 | 0.0075 | 0.0196 |
| Ga41As41H72 | $[-0.64, 0.0]$ | 201 | 301 | 0.0005 | 0.0010 |
| `Andrews` | $[4, 5]$ | 1,844 | 3,751 | 0.0274 | 0.1370 |
| `Laplacian` | $[1, 1.01]$ | 276 | >17,000 | 0.0008 | 0.0044 |

# Results for Ge99H100 -set 1 of stats

| method | degree | # iter | # matvecs | memory |
|---|---|---|---|---|
| filt. Lan. (high-pass) | $d = 20$ | 1,020 | 20,400 | 1,117 |
| | $d = 30$ | 710 | 21,300 | 806 |
| | $d = 50$ | 470 | 23,500 | 508 |
| | $d = 100$ | 340 | 34,000 | 440 |
| filt. Lan. (low-pass) | $d = 10$ | 770 | 7,700 | 806 |
| | $d = 20$ | 600 | 12,000 | 688 |
| | $d = 30$ | 530 | 15,900 | 590 |
| | $d = 50$ | 470 | 23,500 | 508 |
| Part. $\perp$ Lanczos | | 5,140 | 5,140 | 4,883 |
| ARPACK | | 6,233 | 6,233 | 1,073 |

# Results for Ge99H100 -CPU times (sec.)

| method | degree | $\rho(A)v$ | reorth | eigvec | total |
|---|---|---|---|---|---|
| filt. Lan. (high-pass) | $d = 20$ | 1,283 | 77 | 23 | 1,417 |
| | $d = 30$ | 1,343 | 55 | 14 | 1,440 |
| | $d = 50$ | 1,411 | 32 | 9 | 1,479 |
| | $d = 100$ | 1,866 | 26 | 7 | 1,930 |
| filt. Lan. (low-pass) | $d = 10$ | 483 | 124 | 21 | 668 |
| | $d = 20$ | 663 | 57 | 21 | 777 |
| | $d = 30$ | 1,017 | 49 | 15 | 1,123 |
| | $d = 50$ | 1,254 | 26 | 13 | 1,342 |
| Part. $\perp$ Lanczos | | 234 | 1,460 | 793 | 2,962 |
| ARPACK | | 298 | †17,503 | †666 | 18,468 |

# Results for Andrews - set 1 of stats

| method | degree | # iter | # matvecs | memory |
|---|---|---|---|---|
| filt. Lan. (mid-pass) | $d = 20$ | 9,440 | 188,800 | 4,829 |
| | $d = 30$ | 6,040 | 180,120 | 2,799 |
| | $d = 50$ | 3,800 | 190,000 | 1,947 |
| | $d = 100$ | 2,360 | 236,000 | 1,131 |
| filt. Lan. (high-pass) | $d = 10$ | 5,990 | 59,900 | 2,799 |
| | $d = 20$ | 4,780 | 95,600 | 2,334 |
| | $d = 30$ | 4,360 | 130,800 | 2,334 |
| | $d = 50$ | 4,690 | 234,500 | 2,334 |
| Part. $\perp$ Lanczos | | 22,345 | 22,345 | 10,312 |
| ARPACK | | 30,716 | 30,716 | 6,129 |

# Results for Andrews - CPU times (sec.)

| method | degree | $\rho(A)v$ | reorth | eigvec | total |
|---|---|---|---|---|---|
| filt. Lan. (mid-pass) | $d = 20$ | 2,797 | 192 | 4,834 | 9,840 |
| | $d = 30$ | 2,429 | 115 | 2,151 | 5,279 |
| | $d = 50$ | 3,040 | 65 | 521 | 3,810 |
| | $d = 100$ | 3,757 | 93 | 220 | 4,147 |
| filt. Lan. (high-pass) | $d = 10$ | 1,152 | 2,911 | 2,391 | 7,050 |
| | $d = 20$ | 1,335 | 1,718 | 1,472 | 4,874 |
| | $d = 30$ | 1,806 | 1,218 | 1,274 | 4,576 |
| | $d = 50$ | 3,187 | 1,032 | 1,383 | 5,918 |
| Part. $\perp$ Lanczos | | 217 | 30,455 | 64,223 | 112,664 |
| ARPACK | | 345 | †423,492 | †18,094 | 441,934 |

# Results for Laplacian – set 1 of stats

| method | degree | # iter | # matvecs | memory |
|---|---|---|---|---|
| | $600$ | 1,400 | 840,000 | 10,913 |
| mid-pass filter | $1,000$ | 950 | 950,000 | 7,640 |
| | $1,600$ | 710 | 1,136,000 | 6,358 |

# Results for Laplacian – CPU times

| method | degree | $\rho(A)v$ | reorth | eigvec | total |
|---|---|---|---|---|---|
| | $600$ | 97,817 | 927 | 241 | 99,279 |
| mid-pass filter | $1,000$ | 119,242 | 773 | 162 | 120,384 |
| | $1,600$ | 169,741 | 722 | 119 | 170,856 |

## *Conclusion*

- Quite appealing general approach when number of eigenvectors to be computed is large
- and when Matvec is not too expensise
- Will not work too well for generalized eigenvalue problem
- Code available here

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