Updating and downdating algorithms for polynomial least squares fitting

Gérard MEURANT

CEA/DIF

October 6, 2007

Introduction

- Given nodes t_j, weights w_j^2 and values y_j, $j = 1, \ldots, m$ find a polynomial q of degree $n \leq m$ s.t.

$$\sum_{j=1}^{m} (y_j - q(t_j))^2 \, w_j^2$$

is minimized

Introduction

- Given nodes t_j, weights w_j^2 and values y_j, $j = 1, \ldots, m$ find a polynomial q of degree $n \leq m$ s.t.

$$\sum_{j=1}^{m} (y_j - q(t_j))^2 \, w_j^2$$

  is minimized

- The nodes and weights are associated with a discrete inner product

$$\langle f, g \rangle_m = \sum_{j=1}^{m} f(t_j) g(t_j) w_j^2$$

Introduction

- Given nodes t_j, weights w_j^2 and values y_j, $j = 1, \ldots, m$ find a polynomial q of degree $n \leq m$ s.t.

$$\sum_{j=1}^{m} (y_j - q(t_j))^2 \, w_j^2$$

  is minimized

- The nodes and weights are associated with a discrete inner product

$$\langle f, g \rangle_m = \sum_{j=1}^{m} f(t_j) g(t_j) w_j^2$$

- The solution q can be obtained using the orthogonal polynomials p_k associated with this inner product

Introduction (2)

- We use the three-term recurrence relation to represent the orthogonal polynomials p_k

Introduction (2)

- We use the three-term recurrence relation to represent the orthogonal polynomials p_k
- This gives symmetric tridiagonal (Jacobi) matrices J_k with coefficients α_i, β_i

Introduction (2)

- We use the three-term recurrence relation to represent the orthogonal polynomials p_k
- This gives symmetric tridiagonal (Jacobi) matrices J_k with coefficients α_i, β_i
- The inverse eigenvalue problem is to construct J_m from the spectral data t_j, w_j, $j = 1, \ldots, m$

Introduction (2)

- We use the three-term recurrence relation to represent the orthogonal polynomials p_k
- This gives symmetric tridiagonal (Jacobi) matrices J_k with coefficients α_i, β_i
- The inverse eigenvalue problem is to construct J_m from the spectral data $t_j, w_j, j = 1, \ldots, m$
- The eigenvalues of J_m must be the nodes t_j and the first elements of the eigenvectors must be w_j

Solution of the least squares problem

- Eigenvalues of J_m: $\theta_j^{(m)} = t_j$ (Ritz values)

Solution of the least squares problem

- Eigenvalues of J_m: $\theta_j^{(m)} = t_j$ (Ritz values)
- $p^m(\theta_j^{(m)}) = \begin{pmatrix} p_0(\theta_j^{(m)}) & \cdots & p_{m-1}(\theta_j^{(m)}) \end{pmatrix}^T$ is an eigenvector of J_m

Solution of the least squares problem

- Eigenvalues of J_m: $\theta_j^{(m)} = t_j$ (Ritz values)
- $p^m(\theta_j^{(m)}) = \begin{pmatrix} p_0(\theta_j^{(m)}) & \cdots & p_{m-1}(\theta_j^{(m)}) \end{pmatrix}^T$ is an eigenvector of J_m
- $P_m = \begin{pmatrix} p^m(\theta_1^{(m)}) & \cdots & p^m(\theta_m^{(m)}) \end{pmatrix}$

Solution of the least squares problem

- Eigenvalues of J_m: $\theta_j^{(m)} = t_j$ (Ritz values)
- $p^m(\theta_j^{(m)}) = \begin{pmatrix} p_0(\theta_j^{(m)}) & \cdots & p_{m-1}(\theta_j^{(m)}) \end{pmatrix}^T$ is an eigenvector of J_m
- $P_m = \begin{pmatrix} p^m(\theta_1^{(m)}) & \cdots & p^m(\theta_m^{(m)}) \end{pmatrix}$
- $J_m P_m = P_m \Theta_m$, $P_m^T P_m = D_m^{-2}$ diagonal

Solution of the least squares problem

- Eigenvalues of J_m: $\theta_j^{(m)} = t_j$ (Ritz values)
- $p^m(\theta_j^{(m)}) = \begin{pmatrix} p_0(\theta_j^{(m)}) & \cdots & p_{m-1}(\theta_j^{(m)}) \end{pmatrix}^T$ is an eigenvector of J_m
- $P_m = \begin{pmatrix} p^m(\theta_1^{(m)}) & \cdots & p^m(\theta_m^{(m)}) \end{pmatrix}$
- $J_m P_m = P_m \Theta_m$, $P_m^T P_m = D_m^{-2}$ diagonal
- solution $q(t^m) = P_m^T d^m$, $t^m = (t_j)_{j=1,\ldots,m}$

Solution of the least squares problem

- Eigenvalues of J_m: $\theta_j^{(m)} = t_j$ (Ritz values)
- $p^m(\theta_j^{(m)}) = \begin{pmatrix} p_0(\theta_j^{(m)}) & \cdots & p_{m-1}(\theta_j^{(m)}) \end{pmatrix}^T$ is an eigenvector of J_m
- $P_m = \begin{pmatrix} p^m(\theta_1^{(m)}) & \cdots & p^m(\theta_m^{(m)}) \end{pmatrix}$
- $J_m P_m = P_m \Theta_m$, $P_m^T P_m = D_m^{-2}$ diagonal
- solution $q(t^m) = P_m^T d^m$, $t^m = (t_j)_{j=1,\ldots,m}$
- $\sum_{j=1}^m (y_j - q(t_j))^2 w_j^2 = \| P_m D_m^2 y^m - d^m \|^2$

Solution of the least squares problem

- Eigenvalues of J_m: $\theta_j^{(m)} = t_j$ (Ritz values)
- $p^m(\theta_j^{(m)}) = \begin{pmatrix} p_0(\theta_j^{(m)}) & \cdots & p_{m-1}(\theta_j^{(m)}) \end{pmatrix}^T$ is an eigenvector of J_m
- $P_m = \begin{pmatrix} p^m(\theta_1^{(m)}) & \cdots & p^m(\theta_m^{(m)}) \end{pmatrix}$
- $J_m P_m = P_m \Theta_m$, $P_m^T P_m = D_m^{-2}$ diagonal
- solution $q(t^m) = P_m^T d^m$, $t^m = (t_j)_{j=1,\ldots,m}$
- $\sum_{j=1}^m (y_j - q(t_j))^2 w_j^2 = \| P_m D_m^2 y^m - d^m \|^2$
- $d^m = P_m D_m^2 y^m$, $\quad d^n = P_{n,m} D_m^2 y^m$, $n < m$

Updating and downdating

- Assume we know the solution up to index m

Updating and downdating

- Assume we know the solution up to index m
- Updating: add a new triplet of data $\{t_{m+1}, w_{m+1}, y_{m+1}\}$ and compute the solution without restarting from scratch

Updating and downdating

- Assume we know the solution up to index m
- Updating: add a new triplet of data $\{t_{m+1}, w_{m+1}, y_{m+1}\}$ and compute the solution without restarting from scratch
- Downdating: remove a triplet of data

Updating and downdating

- Assume we know the solution up to index m
- Updating: add a new triplet of data $\{t_{m+1}, w_{m+1}, y_{m+1}\}$ and compute the solution without restarting from scratch
- Downdating: remove a triplet of data
- Problem: compute the orthogonal polynomials having t_j and w_j

Updating and downdating

- Assume we know the solution up to index m
- Updating: add a new triplet of data $\{t_{m+1}, w_{m+1}, y_{m+1}\}$ and compute the solution without restarting from scratch
- Downdating: remove a triplet of data
- Problem: compute the orthogonal polynomials having t_j and w_j
- This is done by constructing the Jacobi matrix defined by the coefficients of the three-term recurrence

Construction of the Jacobi matrix J_m

- This can be done with the Stieltjes or the Lanczos algorithms with a diagonal matrix (nodes) and a starting vector (weights)

Construction of the Jacobi matrix J_m

- This can be done with the Stieltjes or the Lanczos algorithms with a diagonal matrix (nodes) and a starting vector (weights)
- Reconstruction algorithms:

Construction of the Jacobi matrix J_m

- This can be done with the Stieltjes or the Lanczos algorithms with a diagonal matrix (nodes) and a starting vector (weights)
- Reconstruction algorithms:
  - De Boor and Golub

Construction of the Jacobi matrix J_m

- This can be done with the Stieltjes or the Lanczos algorithms with a diagonal matrix (nodes) and a starting vector (weights)
- Reconstruction algorithms:
  - De Boor and Golub
  - Gragg and Harrod

Construction of the Jacobi matrix J_m

- This can be done with the Stieltjes or the Lanczos algorithms with a diagonal matrix (nodes) and a starting vector (weights)
- Reconstruction algorithms:
  - De Boor and Golub
  - Gragg and Harrod
  - Gautschi

Construction of the Jacobi matrix J_m

- This can be done with the Stieltjes or the Lanczos algorithms with a diagonal matrix (nodes) and a starting vector (weights)
- Reconstruction algorithms:
  - De Boor and Golub
  - Gragg and Harrod
  - Gautschi
  - Reichel

Construction of the Jacobi matrix J_m

- This can be done with the Stieltjes or the Lanczos algorithms with a diagonal matrix (nodes) and a starting vector (weights)
- Reconstruction algorithms:
  - De Boor and Golub
  - Gragg and Harrod
  - Gautschi
  - Reichel
  - Laurie

Updating

This was considered by Elhay, Golub and Kautsky (1991)

Theorem

Let $\sigma_m = (w_1^2 + \cdots + w_m^2)^{1/2}$. *The solution is*

$$J_{m+1} = Q \begin{pmatrix} J_m & 0 \\ 0 & t_{m+1} \end{pmatrix} Q^T$$

$$\sigma_{m+1} = (\sigma_m^2 + w_{m+1}^2)^{1/2}$$

$$d^{m+1} = Q \begin{pmatrix} d^m \\ w_{m+1} y_{m+1} \end{pmatrix}$$

Q orthogonal matrix uniquely determined by requiring J_{m+1} *to be tridiagonal and Q to be such that*

$$Q(\sigma_m e^1 + w_{m+1} e^{m+1}) = \sigma_{m+1} e^1$$

Updating (2)

- $Q = R_m R_{m-1} \cdots R_1$, R_j: rotation between rows j and $m+1$

Updating (2)

- $Q = R_m R_{m-1} \cdots R_1$, R_j: rotation between rows j and $m+1$
- R_1 computed to have $Q(\sigma_m e^1 + w_{m+1} e^{m+1}) = \sigma_{m+1} e^1$

Updating (2)

- $Q = R_m R_{m-1} \cdots R_1$, R_j: rotation between rows j and $m+1$
- R_1 computed to have $Q(\sigma_m e^1 + w_{m+1} e^{m+1}) = \sigma_{m+1} e^1$
- This creates nonzero elements in the last row (and last column) that are chased by subsequent rotations $R_2, \ldots, R_m$ to make J_{m+1} tridiagonal

Downdating

- This is similar to zeroing a weight. It can be done by hyperbolic rotations

Downdating

- This is similar to zeroing a weight. It can be done by hyperbolic rotations
- This does not work so well... (see example)

Downdating

- This is similar to zeroing a weight. It can be done by hyperbolic rotations
- This does not work so well... (see example)

Example:

$$t_j = -1+2(j-1)/(N-1), w_j = 1/\sqrt{N}, y_j = 1.5+\sin(4t_j),\ j = 1, \ldots, N$$

Downdating

- This is similar to zeroing a weight. It can be done by hyperbolic rotations
- This does not work so well... (see example)

Example:

$$t_j = -1+2(j-1)/(N-1), w_j = 1/\sqrt{N}, y_j = 1.5+\sin(4t_j),\ j = 1, \ldots, N$$

Starting from t_1, w_1 we recursively build the Jacobi matrices by adding one node at a time up to N

Downdating

- This is similar to zeroing a weight. It can be done by hyperbolic rotations
- This does not work so well... (see example)

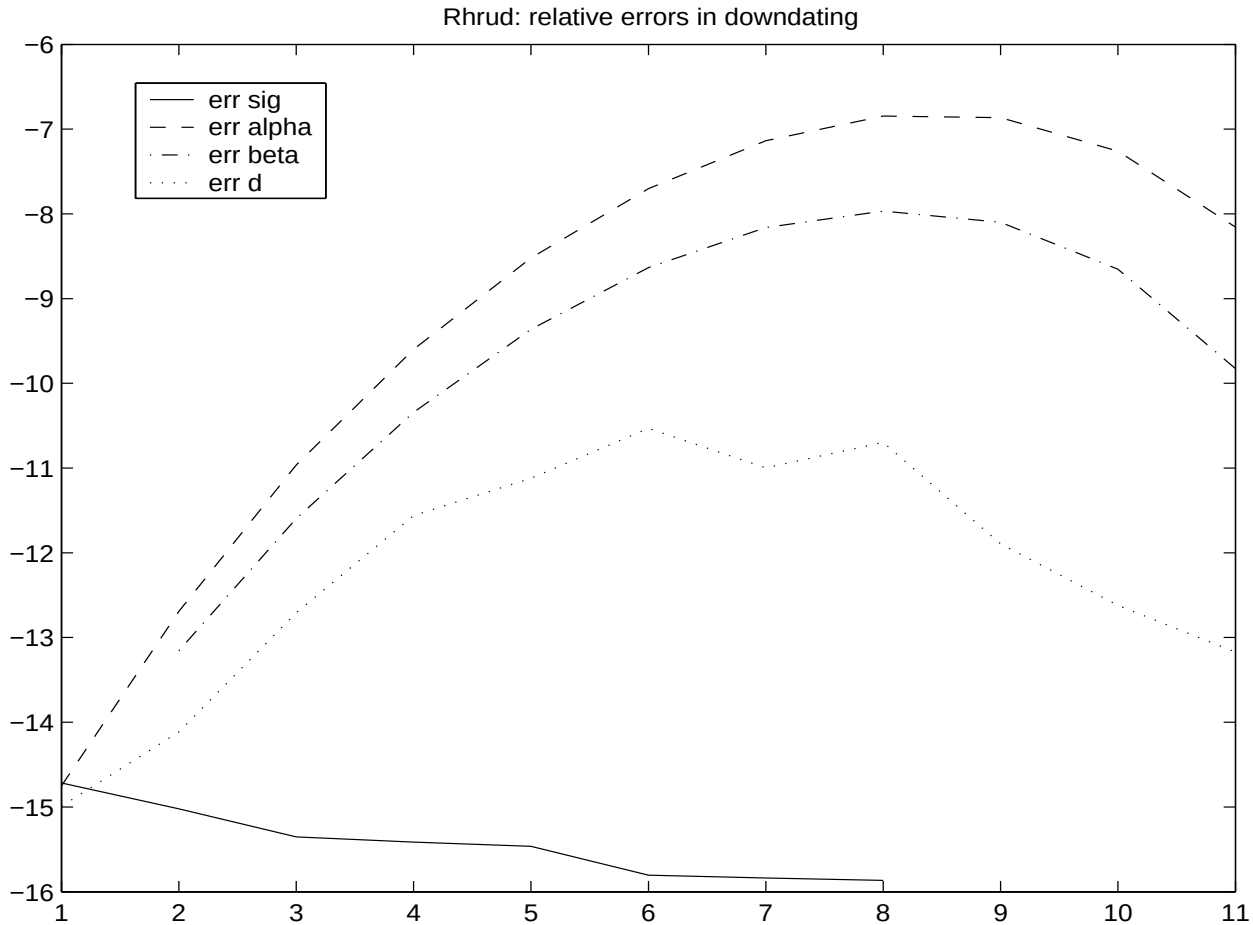
Example:

$$t_j = -1+2(j-1)/(N-1), w_j = 1/\sqrt{N}, y_j = 1.5+\sin(4t_j),\ j = 1, \ldots, N$$

Starting from t_1, w_1 we recursively build the Jacobi matrices by adding one node at a time up to N

We downdate by removing one node at a time and compare $(\alpha, \beta, \sigma, d)$ to what we obtained when updating

Updating with rot. – Downdating with hyp. rot.



Example 1: $\log_{10}$ of relative errors when downdating, $N = 12$

Reversed rotations

Theorem (Elhay, Golub, Kautsky)

Given σ_{m+1}, J_{m+1} and d^{m+1}, the solution when removing the triplet $\{t_{m+1}, w_{m+1}, y_{m+1}\}$ from the data is

$$\begin{pmatrix} \tilde{J}_m & 0 \\ 0 & t_{m+1} \end{pmatrix} = Q J_{m+1} Q^T$$

$$\tilde{\sigma}_m = (\sigma_m^2 - w_{m+1}^2)^{1/2}, \quad \begin{pmatrix} \tilde{d}^m \\ w_{m+1} y_{m+1} \end{pmatrix} = Q d^{m+1}$$

where the orthogonal matrix Q is uniquely determined by requiring $\tilde{J}_m$ to be tridiagonal and such that

$$Q(\sigma_{m+1} e^1 - w_{m+1} q) = \tilde{\sigma}_m e^1$$

q being the normalized eigenvector of J_{m+1} corresponding to t_{m+1} scaled to have a positive first element.

REV

- Seems fine... However, we have to know the eigenvector corresponding to the eigenvalue to be removed

REV

- Seems fine... However, we have to know the eigenvector corresponding to the eigenvalue to be removed
- Elhay, Golub and Kautsky used an eigenvector q computed by solving a linear system with J_{m+1}

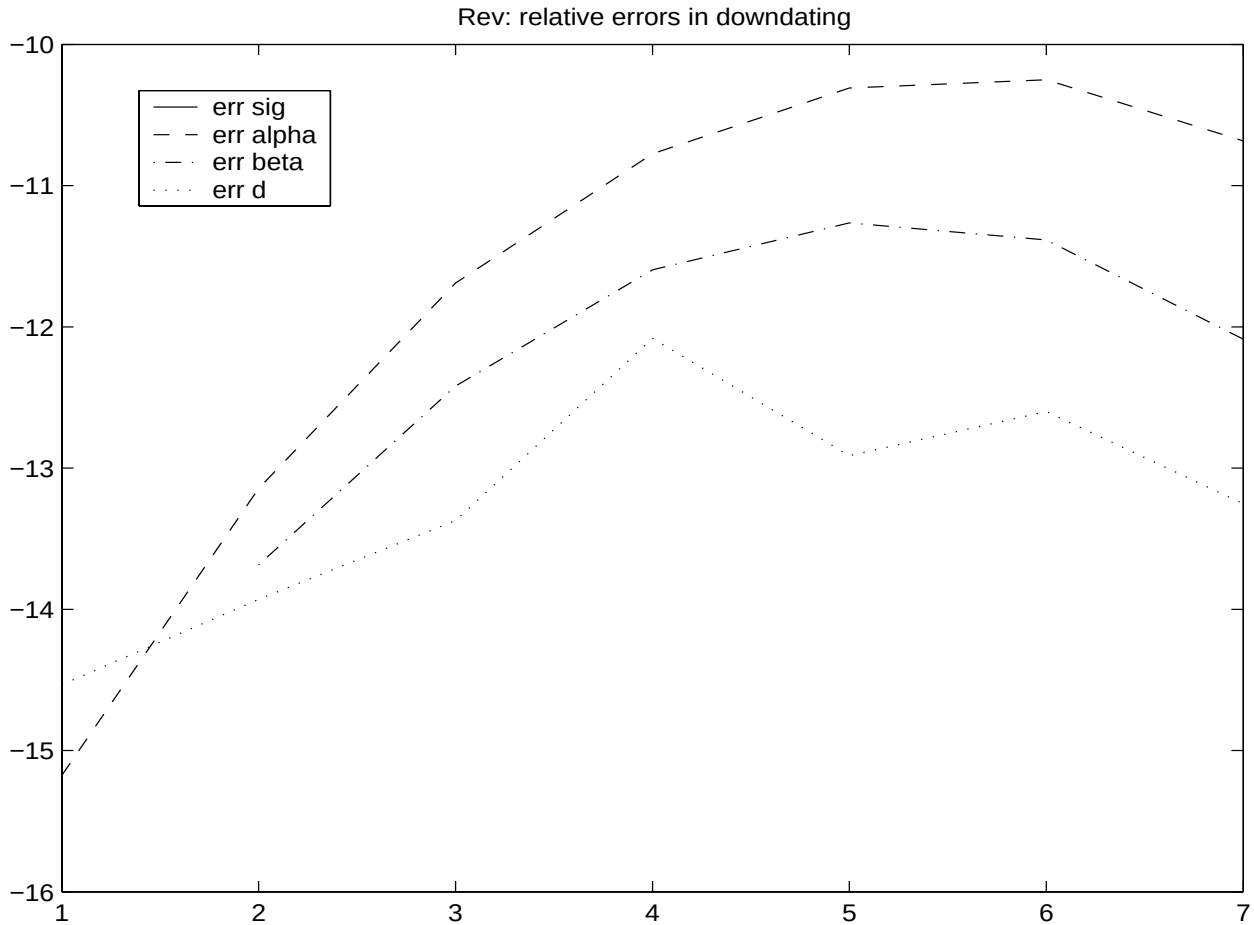
REV

- Seems fine... However, we have to know the eigenvector corresponding to the eigenvalue to be removed
- Elhay, Golub and Kautsky used an eigenvector q computed by solving a linear system with J_{m+1}
- This does not work well...

REV

- Seems fine... However, we have to know the eigenvector corresponding to the eigenvalue to be removed
- Elhay, Golub and Kautsky used an eigenvector q computed by solving a linear system with J_{m+1}
- This does not work well...
- All these computations break down (sqrt of negative numbers) when n is moderately large

Updating with rot. – Downdating with REV.



Example 1: $\log_{10}$ of relative errors when downdating, $N = 8$

Computing the eigenvector

- The eigenvector computed in EGK is not accurate

Computing the eigenvector

- The eigenvector computed in EGK is not accurate
- When updating one node at a time and then downdating, one can incrementally compute the eigenvectors by using the rotations Q

Computing the eigenvector

- The eigenvector computed in EGK is not accurate
- When updating one node at a time and then downdating, one can incrementally compute the eigenvectors by using the rotations Q
- The results are much better...

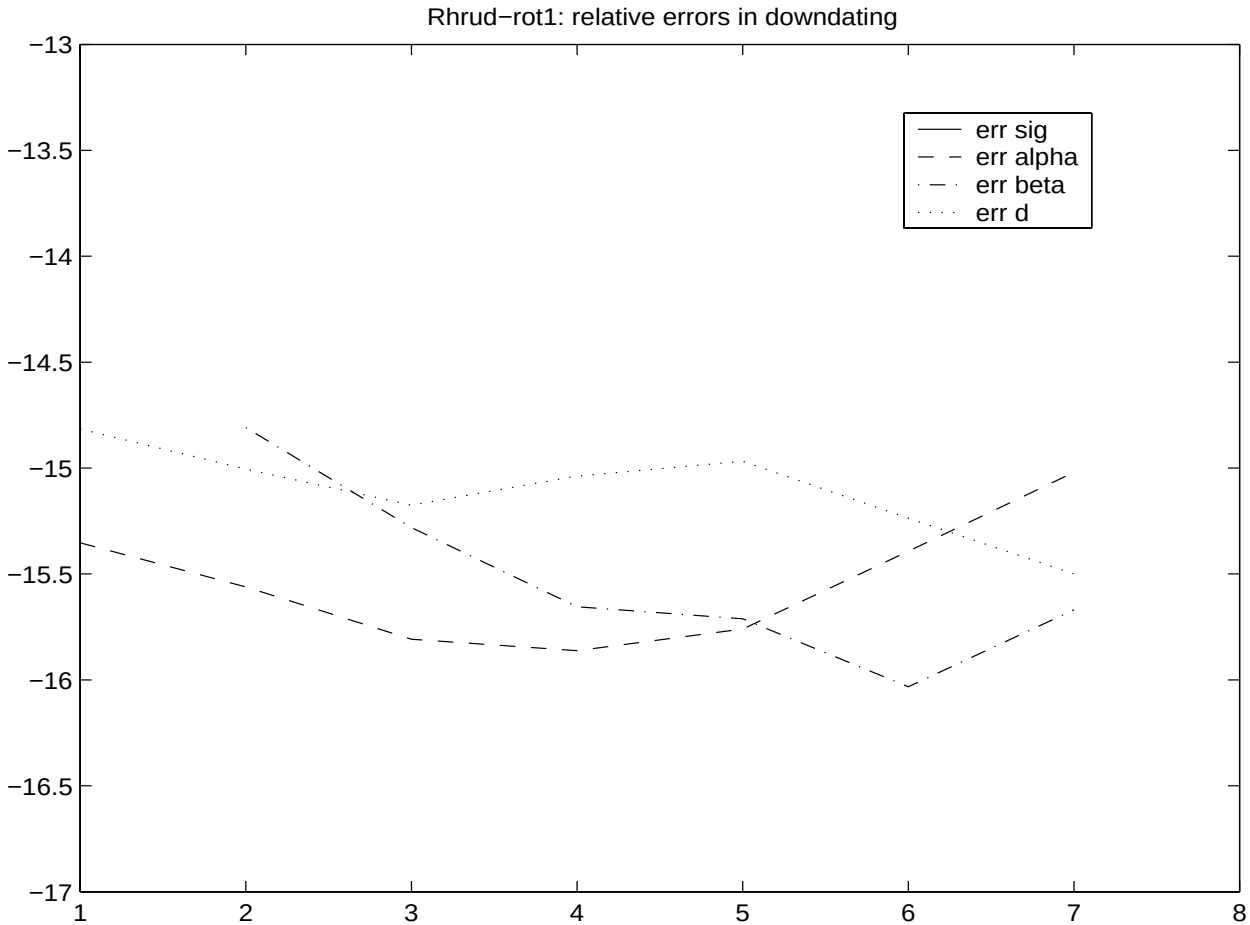
Computing the eigenvector

- The eigenvector computed in EGK is not accurate
- When updating one node at a time and then downdating, one can incrementally compute the eigenvectors by using the rotations Q
- The results are much better...
- We can use large data sets without problem

Computing the eigenvector

- The eigenvector computed in EGK is not accurate
- When updating one node at a time and then downdating, one can incrementally compute the eigenvectors by using the rotations Q
- The results are much better...
- We can use large data sets without problem
- However, this is too costly

Updating with rot. – Downdating with eigenvectors



Example 1: $\log_{10}$ of relative errors when downdating, $N = 8$

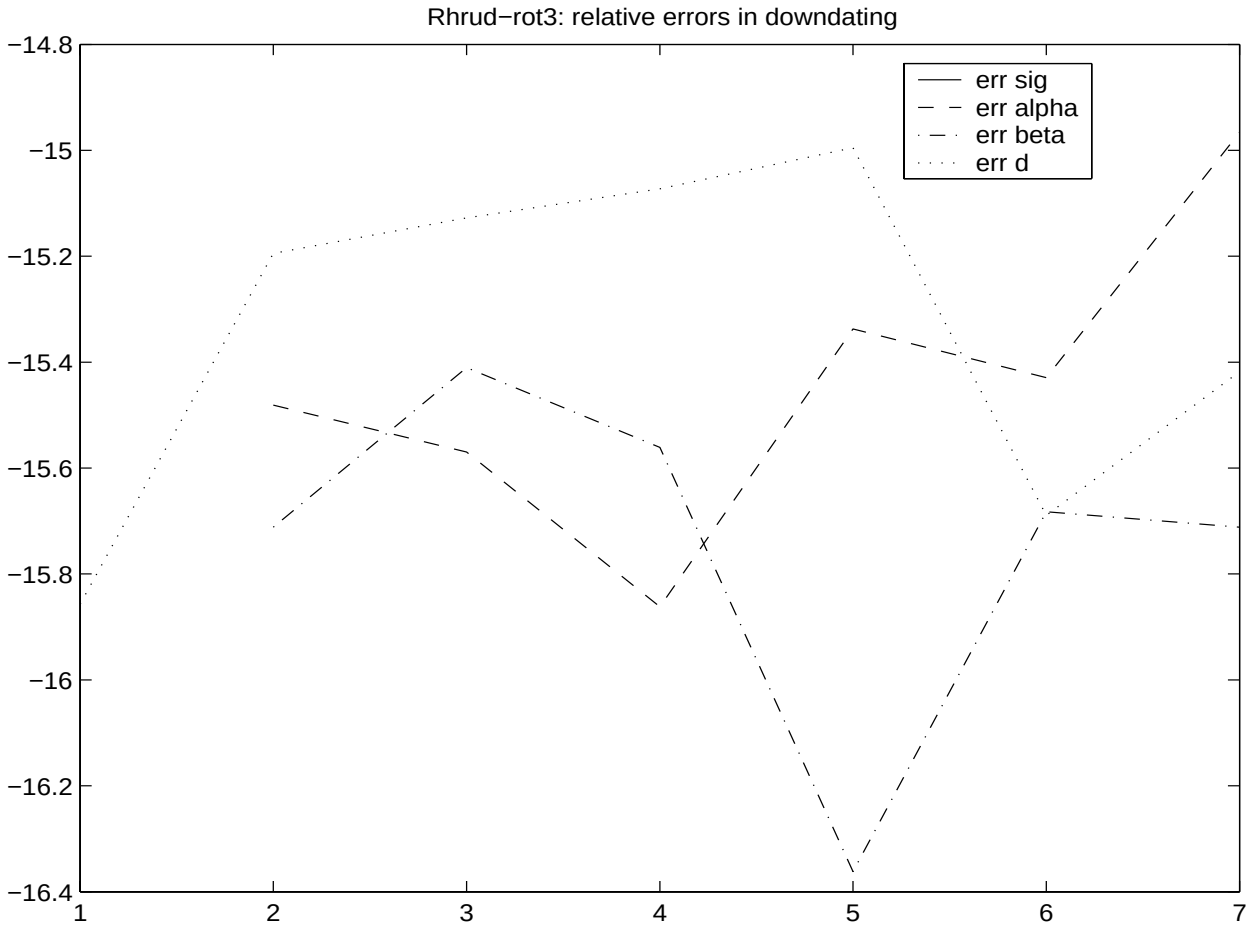
Computing the eigenvector (2)

- Another possibility is to use what is done in MR^3 (Dhillon-Parlett) to compute an accurate eigenvector

Computing the eigenvector (2)

- Another possibility is to use what is done in MR^3 (Dhillon-Parlett) to compute an accurate eigenvector
- Results are even better...

Updating with rot. – Downdating with MR^3



Example 1: $\log_{10}$ of relative errors when downdating, $N = 8$

Example: sliding window

- Assume we know the data for $j = 1, \ldots, N$

Example: sliding window

- Assume we know the data for $j = 1, \ldots, N$
- Let $Y_k = \{t_j, w_j, y_j\}_{j=k}^{k+M-1}$, $k = 1, \ldots, N - M + 1$

Example: sliding window

- Assume we know the data for $j = 1, \ldots, N$
- Let $Y_k = \{t_j, w_j, y_j\}_{j=k}^{k+M-1}$, $k = 1, \ldots, N - M + 1$
- A least squares fit of dim $n \leq M$ is computed for Y_1

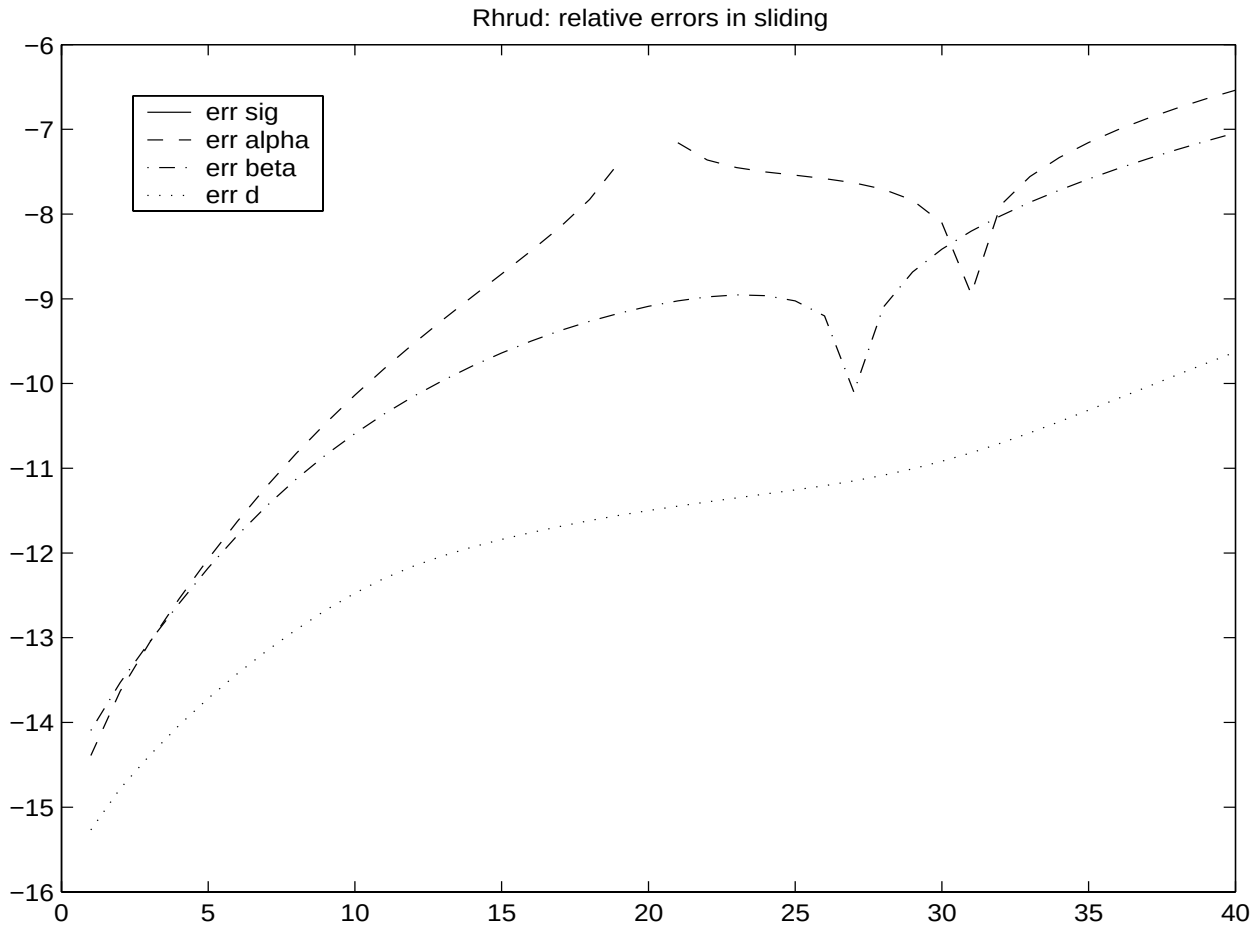
Example: sliding window

- Assume we know the data for $j = 1, \ldots, N$
- Let $Y_k = \{t_j, w_j, y_j\}_{j=k}^{k+M-1}$, $k = 1, \ldots, N - M + 1$
- A least squares fit of dim $n \leq M$ is computed for Y_1
- For $k = 2, \ldots, N - M + 1$ we add $\{t_{k+M}, w_{k+M}, y_{k+M}\}$ and then remove $\{t_{k-1}, w_{k-1}, y_{k-1}\}$

Example: sliding window

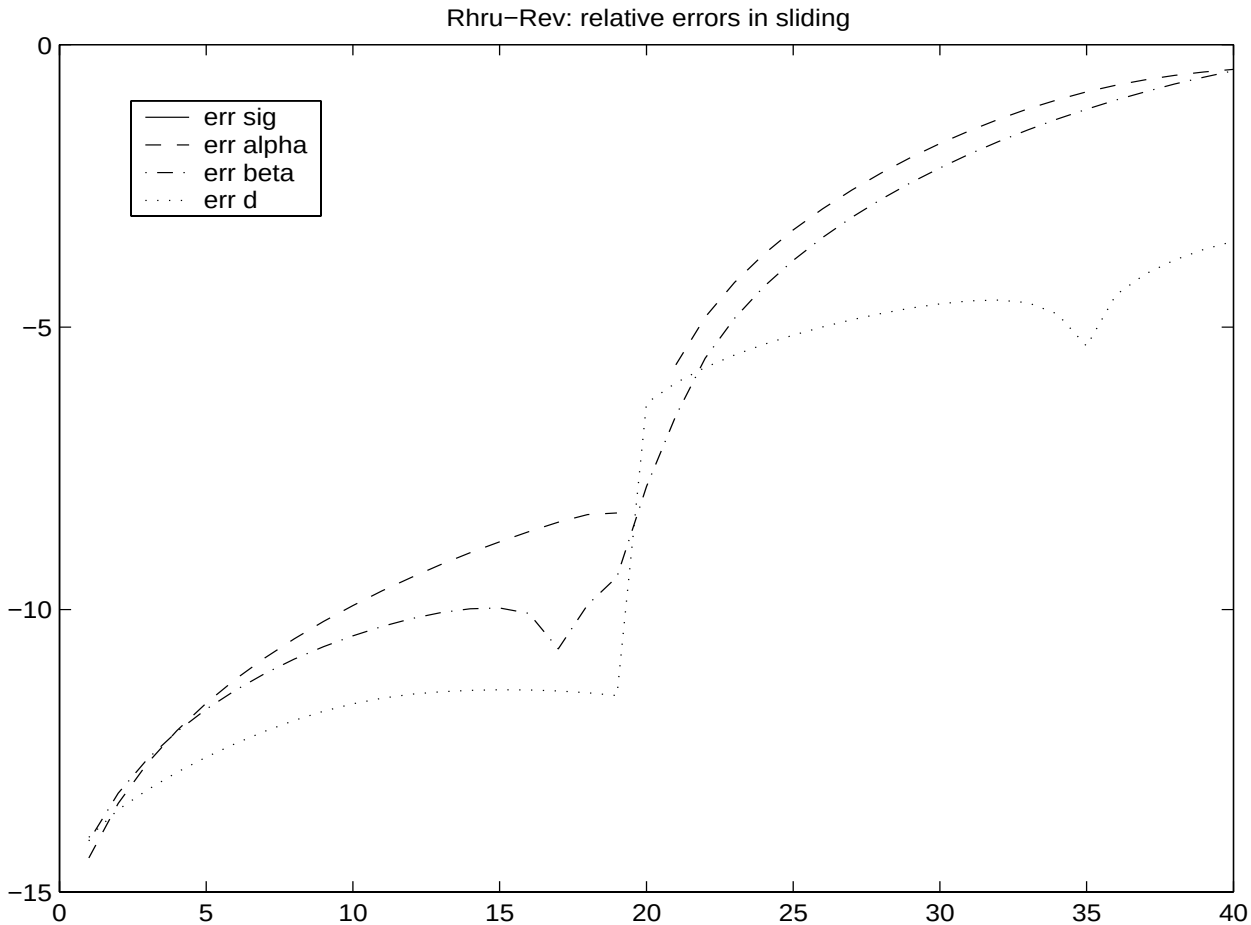
- Assume we know the data for $j = 1, \ldots, N$
- Let $Y_k = \{t_j, w_j, y_j\}_{j=k}^{k+M-1}$, $k = 1, \ldots, N - M + 1$
- A least squares fit of dim $n \leq M$ is computed for Y_1
- For $k = 2, \ldots, N - M + 1$ we add $\{t_{k+M}, w_{k+M}, y_{k+M}\}$ and then remove $\{t_{k-1}, w_{k-1}, y_{k-1}\}$
- We compare to the solution computed directly (using RHR) for Y_k

Updating – Downdating with RHRud



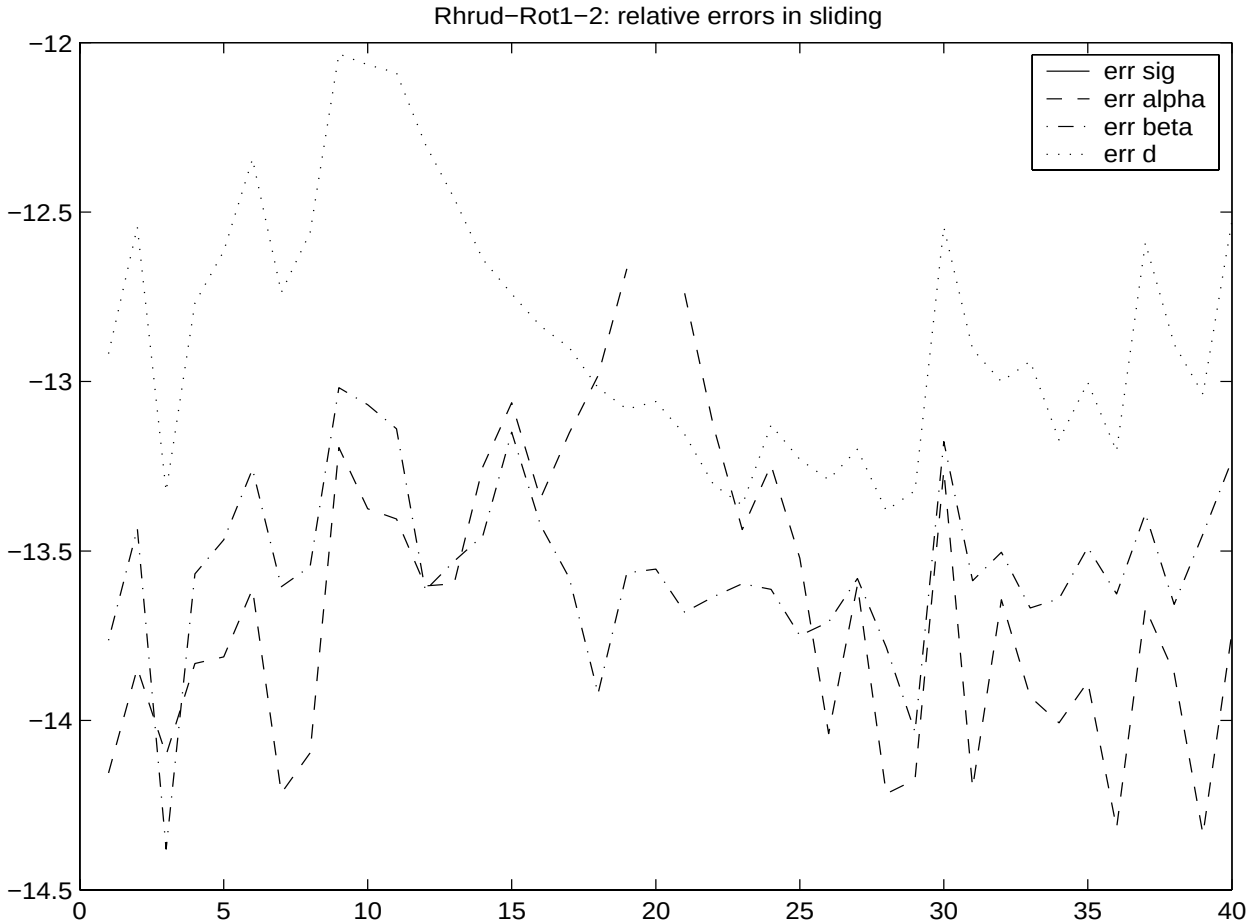
Example 1: $\log_{10}$ of relative errors when sliding the window, $N = 50, M = 10, n = 5$

Updating with rot. – Downdating with REV



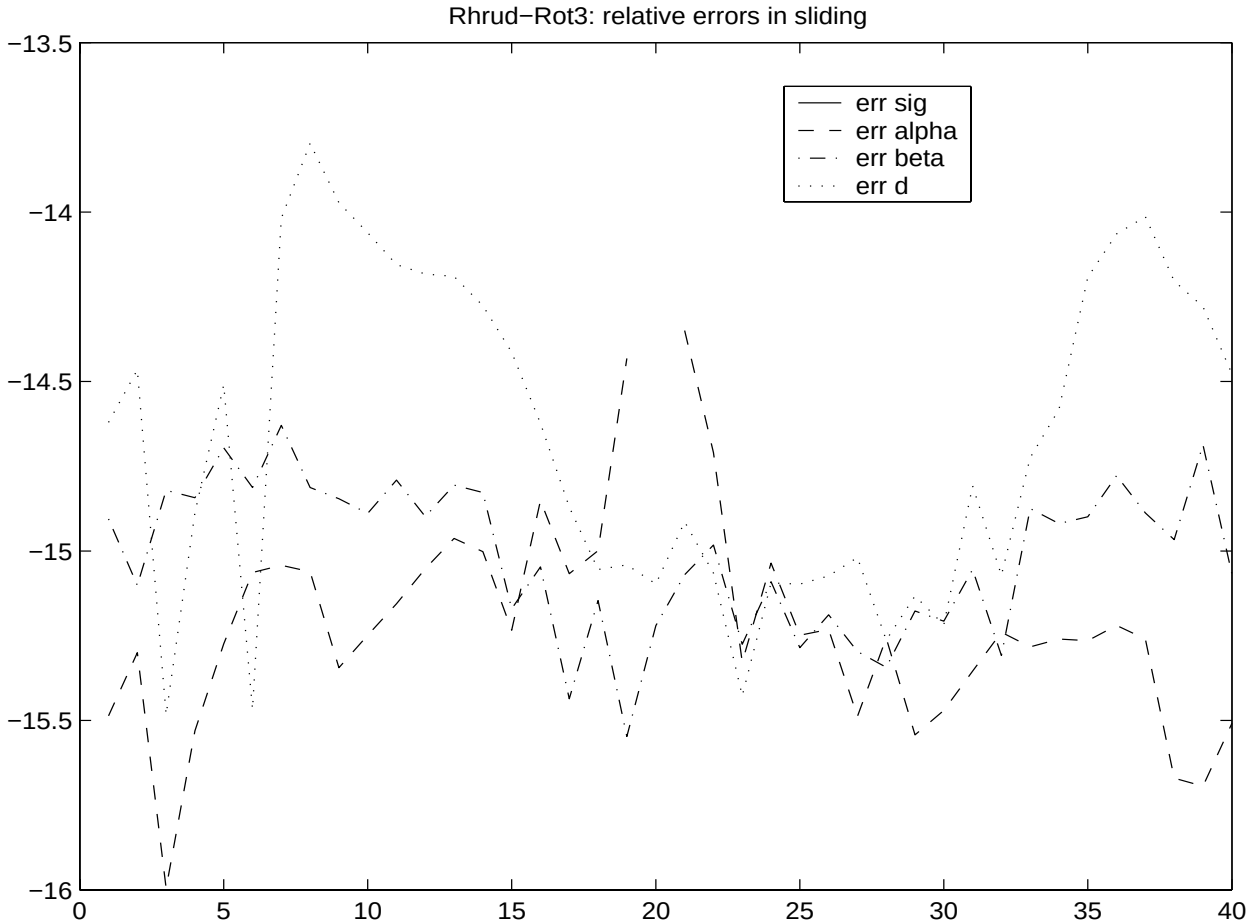
Example 1: $\log_{10}$ of relative errors when sliding the window, $N = 50, M = 10, n = 5$

Updating with rot. – Downdating with eigenvectors



Example 1: $\log_{10}$ of relative errors when sliding the window, $N = 50, M = 10, n = 5$

Updating with rot. – Downdating with MR^3



Example 1: $\log_{10}$ of relative errors when sliding the window, $N = 50, M = 10, n = 5$

Conclusions

- The updating methods using rotations work well

Conclusions

- The updating methods using rotations work well
- To downdate, one has to be careful when computing the eigenvector

Conclusions

- The updating methods using rotations work well
- To downdate, one has to be careful when computing the eigenvector
- Using these modifications, large data sets can be handled