Topics in big matrices, regularization, and inverse problems.

Sergey Voronin

sergey.voronin@outlook.com

2017

Contents

(1) A few applications: geotomography, imaging and compression, sparse signals, data fitting.

(2) Big matrices.

(3) Low rank decompositions and randomized algorithms.

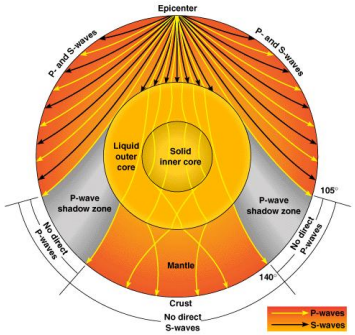
(4) Some examples from applications and HPC software.

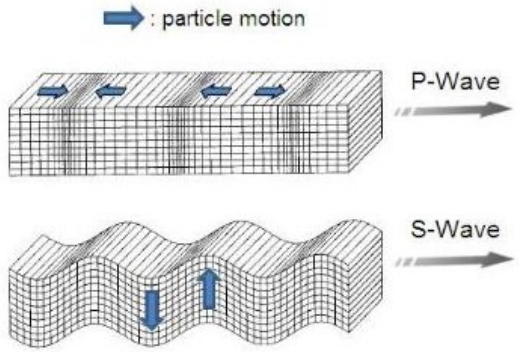
(5) More on regularization and non-smooth minimization.

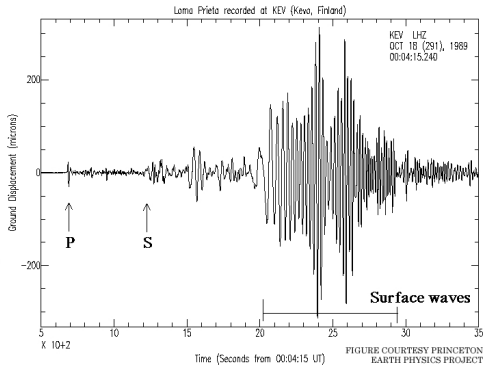
(1) Some applications of my work: geotomography, imaging and compression, sparse signals, data fitting.

Geotomography delay time inverse problem

Goal: to create a 3D map of the interior structure of the Earth using data from earthquakes. Existing spherically symmetric model $v_0(\mathbf{r})$ and delay times δT_i used to construct corrections $\delta v(\mathbf{r})$.







$$\delta T_i = T_i - T_i^0 = \int_{R_i} \frac{d\mathbf{r}}{v(\mathbf{r})} - \int_{R_i^0} \frac{d\mathbf{r}}{v_0(\mathbf{r})}$$

By Fermat's principle, travel time of ray is stationary with respect to small changes in ray path, so we can use the reference path R_i^0:

$$\delta T_i = \int_{R_i} \delta v^{-1}\left(\mathbf{r}(s)\right)\, ds \approx -\int_{R_i^0} \frac{\delta v(\mathbf{r})}{v_0^2(\mathbf{r})}\, ds = \int_{s_i(0)}^{s_i(\Delta)} -v_0^{-1}(\mathbf{r}) \frac{\delta v(\mathbf{r})}{v_0(\mathbf{r})}\, ds$$

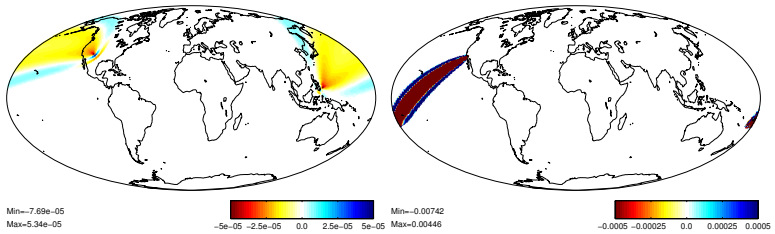
We typically replace $-v_0^{-1}(\mathbf{r})$ by a sensitivity kernel $K(\mathbf{r})$.

Linearization to $Ax = \bar{b}$ (with error) for $m \times n$ matrix A

$$(\delta T)_i = \sum_{j=1}^{N} x_j \int [K(r)]_{i,j} dr \implies \bar{b}_i = \sum_{j=1}^{N} K_{i,j} m_j \implies \bar{b} = Ax$$

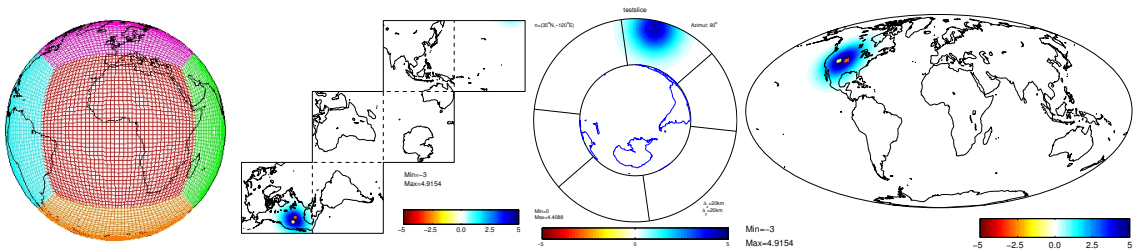
$$\implies Ax = \bar{b} \text{ with } A_{i,j} = K_{i,j},\ x_j = \frac{\delta v_j}{(v_0)_j} \text{ and } \bar{b}_i = [(\delta T)_i].$$

Matrix rows correspond to source-receiver pair kernel



Matrix columns correspond to coordinate system grid

E.g. take set of radii (levels) and at each level project sphere onto cube.



Solving $Ax \approx b$ to recover useful features.

- A has rapid decay of singular values, rhs noisy: need least squares + regularizer.
- Possibility to include correction terms.
- Need to pick up multi-scale resolution components in x.

- Least Squares

$$\bar{x} = \arg\min_x \left\{ \|Ax - b\|_2^2 \right\} \quad \Longrightarrow \quad A^T A\bar{x} = A^T b$$

$$\Longrightarrow \quad \bar{x} = V \mathrm{Diag}(\frac{1}{\sigma_i}) U^T b$$

Very large norm for small σ_i.

- Tikhonov Regularization

$$\bar{x}_\lambda = \arg\min_x \left\{ \|Ax - b\|_2^2 + \lambda \|x\|_2^2 \right\} \quad \Longrightarrow \quad (A^T A + \lambda I)\bar{x}_\lambda = A^T b$$

$$\Longrightarrow \quad \bar{x}_\lambda = V \mathrm{Diag}(\frac{\sigma_i}{\sigma_i^2 + \lambda}) U^T b$$

Tikhonov minimization filters the effects of singular vectors corresponding to small singular values.

Variance of Solutions

We define:

$$\mathrm{Cov}(x,y) = \mathbb{E}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])] \; ; \; \mathrm{var}(x) = \mathrm{Cov}(x,x) = \mathbb{E}[(x - \mathbb{E}[x])^2]$$

Assume $b = \bar{b} + e$ (the true data plus noise). Assume the two are uncorrelated and that $E[e] = 0$.

$$\begin{aligned} \mathrm{var}(e) &= \mathbb{E}[(e - \mathbb{E}[e])(e - \mathbb{E}[e])^T] = \mathbb{E}[ee^T] = \nu^2 I \\ \mathrm{var}(b) &= \mathbb{E}[(b - \mathbb{E}[b])(b - \mathbb{E}[b])^T] = \mathbb{E}[ee^T] = \nu^2 I \end{aligned}$$

For the least squares solution:

$$\| \mathrm{var}(\bar{x}) \|_2^2 = \frac{\nu^2}{\sigma_r^2}$$

For Tikhonov solution:

$$\| \mathrm{var}(\bar{x}_\lambda) \|_2^2 \leq \frac{\nu^2}{4\lambda}$$

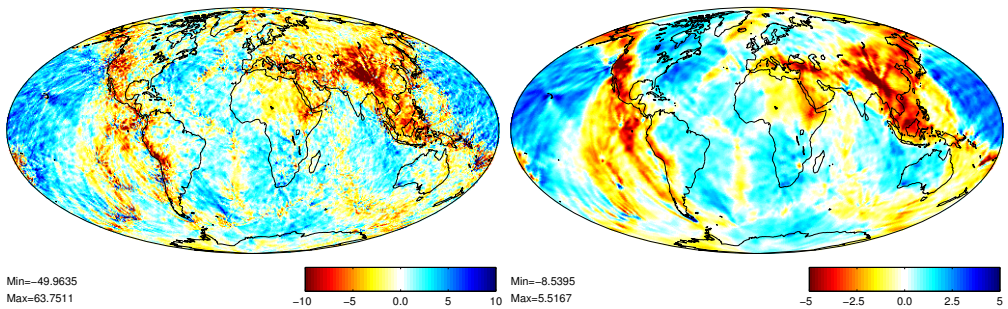
Regularized solutions are not as sensitive to noise in b and **to approximation of A.**

Tikhonov regularization with smoothing

$$\bar{x} = \arg\min_x \left\{ \|Ax - b\|_2^2 + \lambda_1 \|x\|_2^2 + \lambda_2 \|Lx\|_2^2 \right\}$$

$$\arg\min_x \left\| \begin{bmatrix} A \\ \sqrt{\lambda_1} I \\ \sqrt{\lambda_2} L \end{bmatrix} x - \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix} \right\|_2^2 \Longrightarrow \begin{bmatrix} A \\ \sqrt{\lambda_1} I \\ \sqrt{\lambda_2} L \end{bmatrix}^T \begin{bmatrix} A \\ \sqrt{\lambda_1} I \\ \sqrt{\lambda_2} L \end{bmatrix} \bar{x} = \begin{bmatrix} A \\ \sqrt{\lambda_1} I \\ \sqrt{\lambda_2} L \end{bmatrix}^T \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}$$

Smoothness controlled by Laplacian operator L



Including correction terms.

$$\{\bar{x}, \bar{v}\} = \arg\min_{x,v} \left\{ \left\| \begin{bmatrix} A & C \\ 0 & D \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2 + \lambda_1 \|x\|_2^2 + \lambda_2 \|Lx\|_2^2 \right\}$$

where C is a matrix of correction columns and $D = \epsilon I$ is a matrix of damping terms to control the size of the correction solution portion v.

Representation of model in wavelet basis (with F.J. Simons, G. Nolet, et al., 2011, 2013)

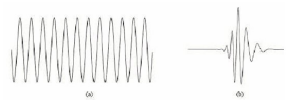
Let $w = Wx$ and $x = W^{-1}w$. Then $Ax = b \implies AW^{-1}w = b$.

$$b_i = \sum_{j=1}^{N} K_{i,j}x_j = \sum_{j=1}^{N} K_{i,j}\left(\sum_{k=1}^{N} W^{-1}_{j,k}w_j\right) = \sum_{j=1}^{N}\sum_{k=1}^{N} K_{i,j}W^{-1}_{j,k}w_j$$

For orthogonal transform W one has $\|Wx\|_2^2 = \|x\|_2^2$ and hence minimizing $\|Ax-b\|_2^2 + \lambda\|Wx\|_2^2$ is equivalent to minimizing $\|Ax-b\|_2^2 + \lambda\|x\|_2^2$. For sparse minimization, we use a sparse penalty in the wavelet domain, e.g.:

$$\bar{w} = \arg\min_{w}\left\{\|AW^{-1}w - b\|_2^2 + 2\tau\|w\|_1\right\} \quad ; \quad \bar{x} = W^{-1}\bar{w}$$

Compactly supported wavelets

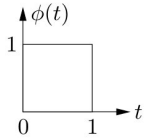


$$\psi_{k,n}(t) = a^{-k/2}\psi(a^{-k}t - nb)$$

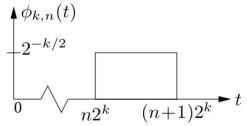
$$x(t) = \sum_{k\in\mathbb{Z}}\sum_{n\in\mathbb{Z}}\langle x, \psi_{k,n}\rangle\psi_{k,n}(t)$$

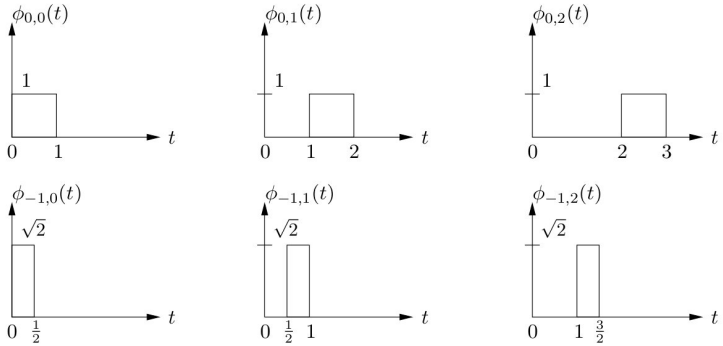
Translations and Scalings of the Haar Wavelet

$$\phi(t) = \begin{cases} 1 & 0 \le t < 1 \\ 0 & \text{else} \end{cases}.$$

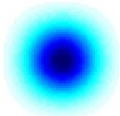


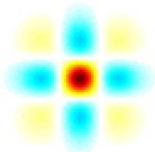
$$\begin{aligned} \phi_{k,n}(t) &= 2^{-k/2}\phi(2^{-k}t - n), \quad k, n \in \mathbb{Z}, \\ &= 2^{-k/2}\phi\big(\tfrac{1}{2^k}(t - n2^k)\big), \end{aligned}$$





Different choices of wavelets





Sparse and wavelet based regularization

- Different constraints on the solution are possible. E.g. sparsity (few nonzeros) with respect to a suitable basis.

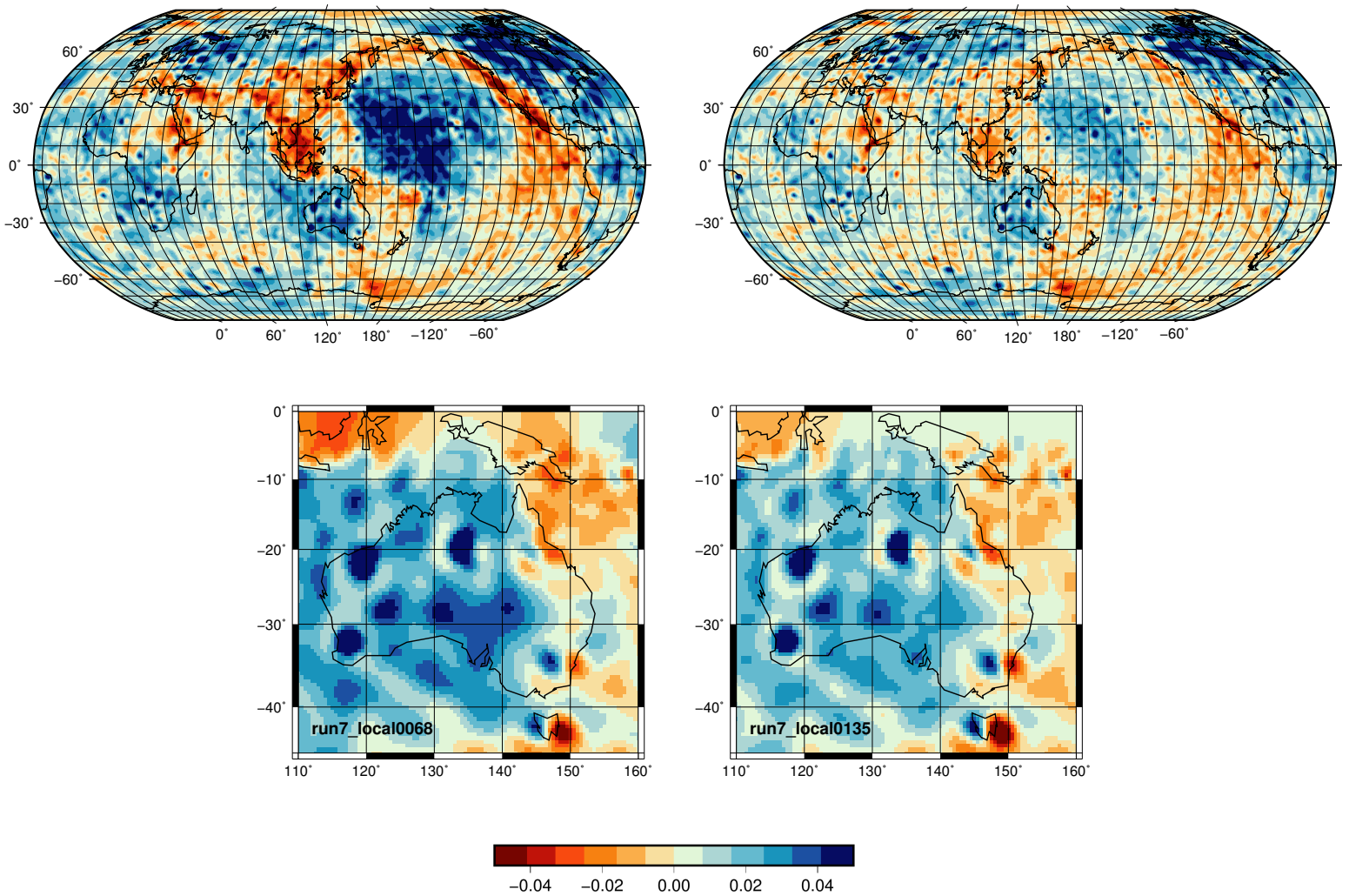
$$\begin{aligned}
\bar{x}_1 &= \arg\min_x \left\{ \|Ax - b\|_2^2 + \tau \|x\|_2^2 + \lambda \|Lx\|_2^2 \right\} \\
\bar{x}_2 &= \arg\min_x \left\{ \|Ax - b\|_2^2 + \tau \|x\|_1 \right\} \\
\bar{w}_3 &= \arg\min_w \left\{ \|AW^{-1}w - b\|_2^2 + \tau \|w\|_1 \right\} \; ; \; \bar{x}_3 = W^{-1} w_3
\end{aligned}$$

- These opt problems can be solved via CG and iterative thresholding:

$$\begin{aligned}
&\left(A^*A + \tau I + \lambda L^*L\right) \bar{x}_1 = A^*b \\
&\bar{w}^{n+1} = \mathbb{S}_{\frac{\tau}{2}} \left(\bar{w}^n + A^*(b - A\bar{w}^n)\right)
\end{aligned}$$

- Need be able to apply methods with very large matrices (several TB).

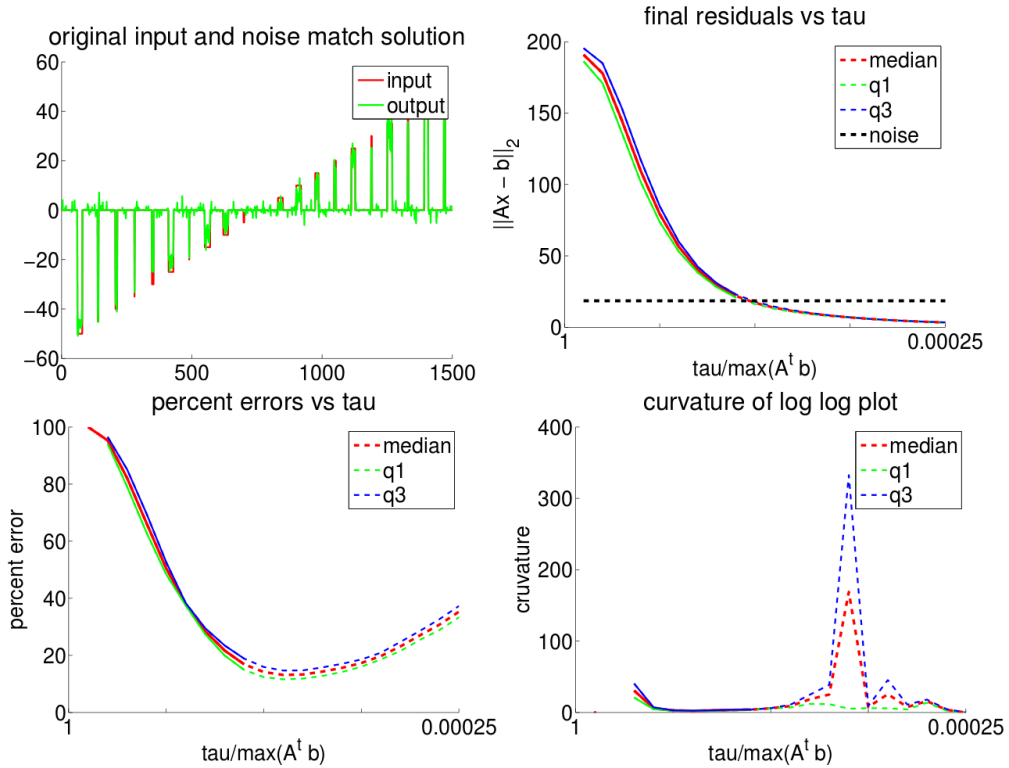
Sample global and local reconstructions (68 KM, 135 KM)



These models consist of 3 *million variables; required significant amounts of large cluster computer use.*

Regularization parameter estimation

Need to find *optimal* regularization parameter τ to use.



$$\bar{\epsilon} = \log \|x_\tau\| \quad \text{and} \quad \bar{\rho} = \log \|Ax_\tau - b\|.$$

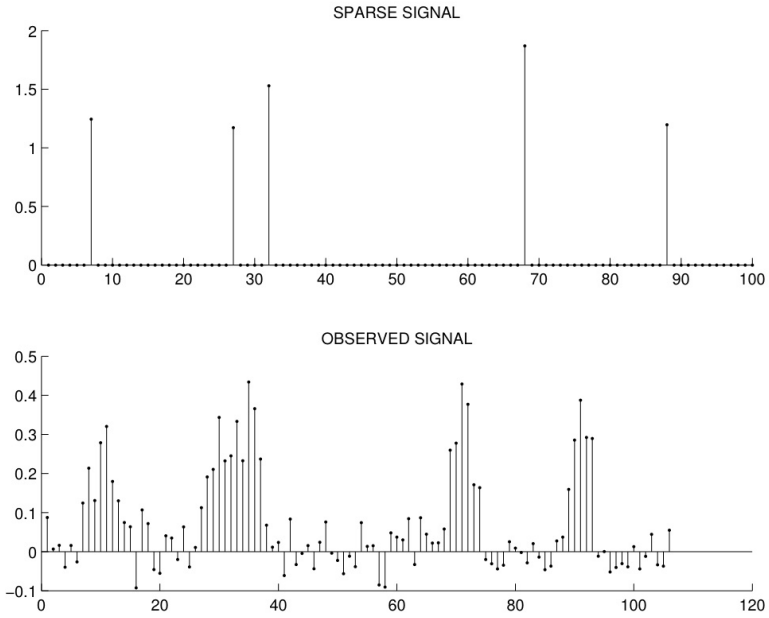
$$\bar{c_\tau} = 2\frac{\bar{\rho}'\bar{\epsilon}'' - \bar{\rho}''\bar{\epsilon}'}{((\bar{\rho}')^2 + (\bar{\epsilon}')^2)^{\frac{3}{2}}},$$

$\Rightarrow$ Requires several runs with possibly very large matrix A.

Sparse signals

- Can one identify the *support* of a sparse signal from a limited number of measurements?

E.g. intermittent bird singing in the forest, sound blurred by strong wind. (One wants to find the times when a particular noise is made) $\Rightarrow Ax = b$.



- Can do frequency filtering with FFT but this will not tell you where in time a particular frequency occured.
- Typically, can phrase this as a linear problem $Ax = b$ s.t x is sparse and would like to identify the *support* of x.

Denoising and deblurring of images

Denoising with Wavelet thresholding:

$$\tilde{w} = \mathbb{T}[Wx] \Rightarrow \tilde{x} = W^{-1}\tilde{w}$$

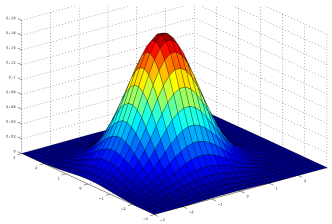
Deblurring with FFT and filter:

$$\begin{aligned} g(x,y) &= f(x,y) \star h(x,y) + n(x,y) \\ G(k_x,k_y) &= F(k_x,k_y)H(k_x,k_y) + N(k_x,k_y) \\ \tilde{F}(k_x,k_y) &= Y(k_x,k_y)G(k_x,k_y) \end{aligned}$$

For noisy and blurred images, iterative de-convolution can be used.

Known blurring source

When the blurring source is known (or can be estimated) a linear optimization problem can be setup. Images can be blurred via convolution:


$(\star)$

$+$ (noise) $=$


Blurring can be represented in terms of a linear operator:

```
bm = zeros(npr,npr);
for j=1:npr
    ej = zeros(npr,1);
    ej(j) = 1;
    bmj = conv2(reshape(ej,m,n),gaussian2d(7,2.5),'same');
    bm(:,j) = bmj(:);
end
```

Deblurring done in terms of a least squares problem:

$$Bx = y + n \rightarrow \bar{x} = \arg\min_{x} \left\{ \|Bx - y\|^2 + \Phi(x) \right\}$$

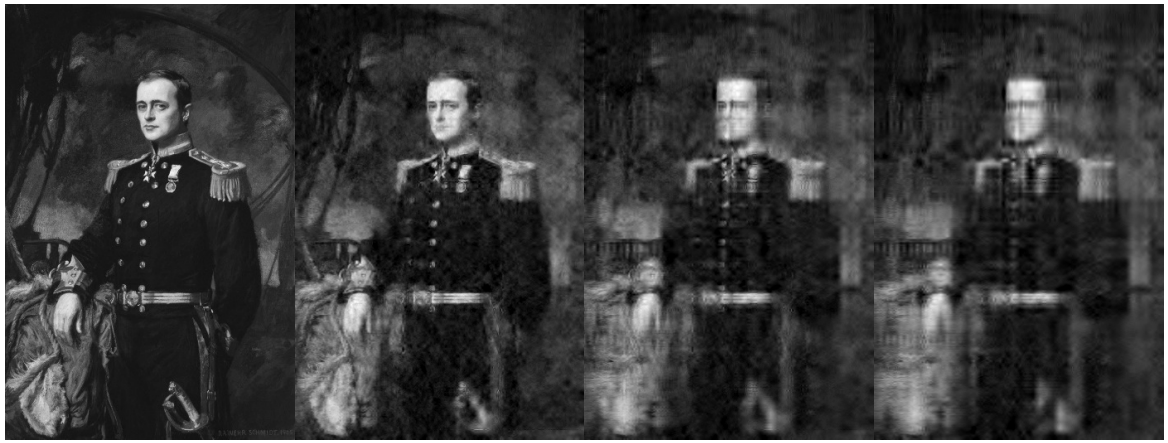
- What to pick for $\Phi(x)$? How to estimate B when the blur source is unknown?

High ratio image and video compression

- Take image matrix X (or well compressible portion).
- Apply 2D wavelet transform (CDF 97) to get $w = WX$.
- Apply thresholding to get $\tilde{w} = \mathbb{T}(w)$.
- Further compress $\tilde{w}$. Use $W^{-1}(\cdot)$ to reconstruct.

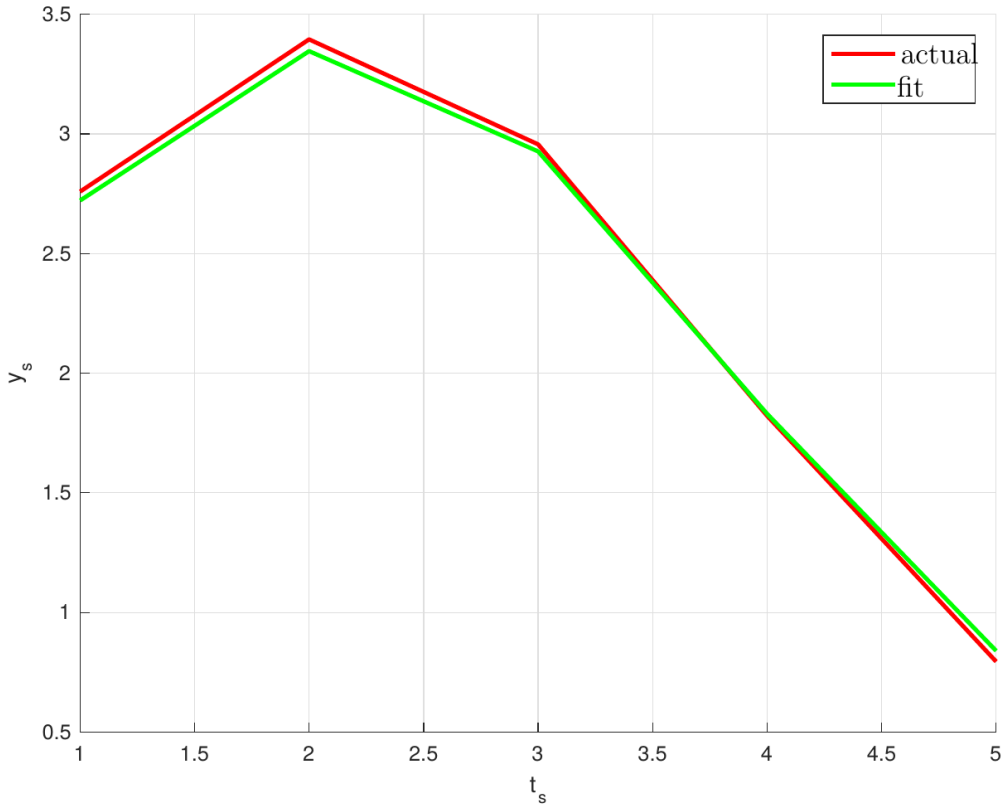
How to compress $\tilde{w}$ further?

Original and compression ratios $2, 4, 8$ after pure wavelet compression.



Nonlinear model fitting

Fitting set of points $(t_1, y_1), \ldots, (t_m, y_m)$ with a nonlinear model. E.g. $F(x,t) = x_1 \exp\left(-\frac{(t-x_2)^2}{2x_3^2}\right) + x_4$. Penalty: $g(x) = \frac{1}{2}\|r(x)\|^2$.



Newton and Gauss Newton

Fitting set of points $(t_1, y_1), \ldots, (t_m, y_m)$ to a nonlinear model. Let $g(x) = \frac{1}{2}\|r(x)\|^2$ with $r_i(x) = y_i - F(x, t_i)$.
Model fitting min problem: $\bar{x} = \arg\min_x g(x)$.
Setting $\nabla g(x) = 0$, yields with Newton's method:

$$\bar{x}^{n+1} = \bar{x}^n - \left[\nabla^2 g(\bar{x}^n)\right]^{-1} \nabla g(\bar{x}^n)$$

Expanding the gradient and Hessian of g yields:

$$\nabla g(x) = \sum_{i=1}^{m} r_i(x) \nabla r_i(x) = J^T r(x) \text{ where } J = J[r(x)]$$

$$\nabla^2 g(x) = \sum_{i=1}^{m} \nabla r_i(x) \nabla r_i(x)^T + \sum_{i}^{m} r_i(x) \nabla^2 r_i(x) = J^T J + T(x) \approx J^T J.$$

$T(x) = \sum_{i=1}^{m} r_i(x) \nabla^2 r_i(x)$ and $J[r(x)]_{(i,:)} = \nabla r_i(x)^T = -\nabla F(x, t_i)^T$.

Gauss-Newton Method: $\bar{x}^{n+1} = \bar{x}^n - \left[J_n^T J_n\right]^{-1} J_n^T r_n$ is easy but not stable!

Improvements of Gauss-Newton method

- Introduction of step size ("α_n")

$$\bar{x}^{n+1} = \bar{x}^n - \alpha_n \left[J_n^T J_n \right]^{-1} J_n^T r_n$$

$$\alpha_n = \arg\min_{\alpha} g(\bar{x}^n - \alpha s^n) \quad \text{with} \quad J_n^T J_n s^n = J_n^T r_n.$$

- Regularization (Levenberg-Marquard method): $J_n^T J_n y = J_n^T r_n$ replaced by ℓ_2 norm penalty: $(J_n^T J_n + \lambda I)\tilde{y} = J_n^T r_n$. (Prof. Mikesell's group looking into this).
- Other types of regularization here? Methods for parameter estimation of λ? Previous tools apply to this non-linear problem.

(2) Matrix compression. The matrices in applications can be very big!

The big matrix A

In Geotomography application, A is of size 2968933×3637248 ($\approx 2-3$ TB in sparse format). It's too big. We divide into 20 blocks:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{20} \end{bmatrix}$$

Each block is between $50,000$ and $500,000$ rows. Can do block matrix operations:

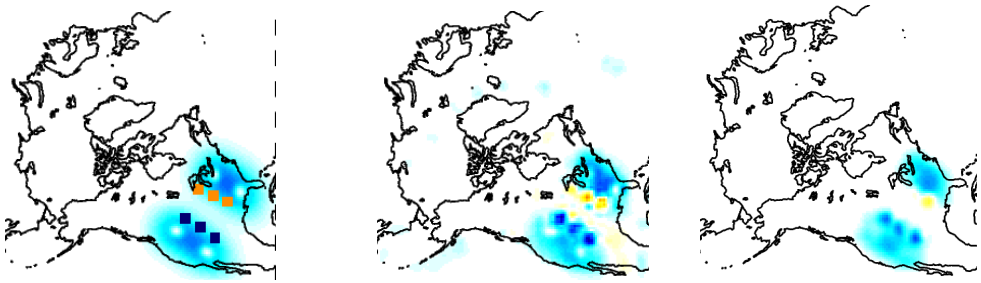
$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix} \Longrightarrow Ax = \begin{bmatrix} A_1 x \\ A_2 x \\ \vdots \\ A_p x \end{bmatrix} \quad ; \quad A^T y = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix} = \sum_{j=1}^{p} A_j^T y_j$$

Idea for compressing blocks of A

The blocks of A in original form are too big. Want to compress them in a simple way. We appeal to Wavelet image compression:

$$x \approx W^{-1}\left(Thr(Wx)\right)$$

The original image is approximately equal to the inverse transform of the thresholded forward transform of the image.



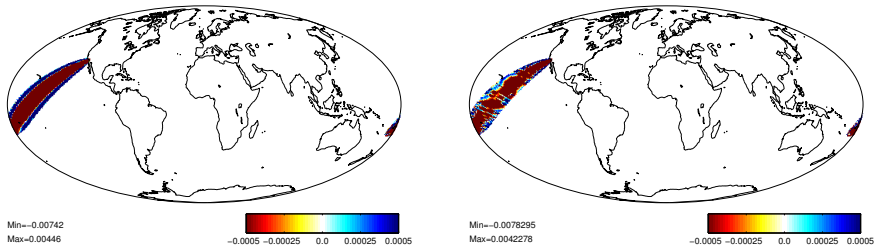
Original; retain 15% of largest (by absolute value) wavelet coefficients; retain 3%. By thresholding, we mean the hard thresholding function:

$$H_\alpha(x) = \{x \text{ if } |x| > \alpha \quad \text{and} \quad 0 \text{ if } |x| \leq \alpha\}$$

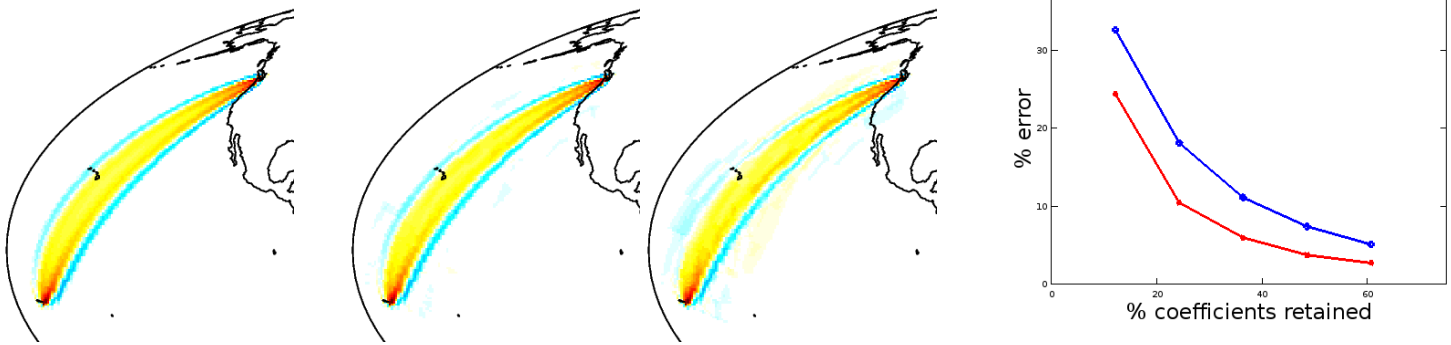
To see if this works for the matrix, we apply it to some rows of A.

Wavelet compression for kernels

Left: x ; Right: $W^{-1}\left(Thr(Wx)\right)$ retain 5% largest coefficients



Another kernel, from 25% and 5% retained.



We have to retain about 25%.

Applying wavelet compression to matrix vector ops [with D. Mikesell, G. Nolet, 2015]

$$A = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix} \rightarrow M = \begin{bmatrix} Thr(Wr_1^T)^T \\ Thr(Wr_2^T)^T \\ \vdots \\ Thr(Wr_m^T)^T \end{bmatrix} = Thr(AW^T) \approx AW^T$$

We can then approximate the operations Ax and $A^T y$ with the matrix M.

$$Mx \approx AW^T x \quad \text{and} \quad M^T y \approx (AW^T)^T y = WA^T y$$

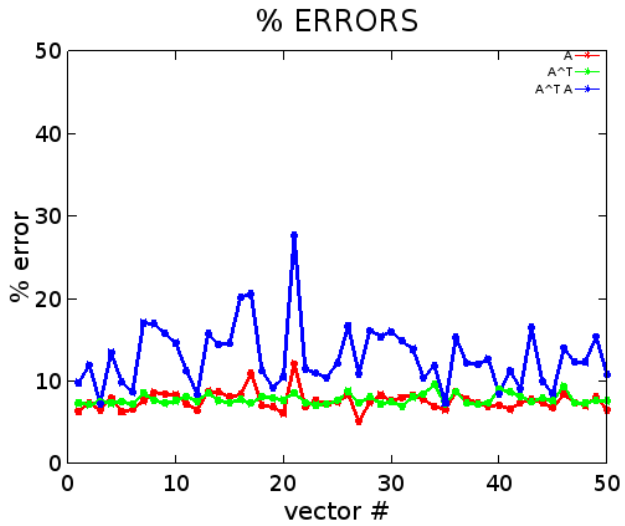
we obtain the approximation formulas:

$$\boxed{Ax \approx MW^{-T} x \quad \text{and} \quad A^T y \approx W^{-1} M^T y}$$

To test how well this works, we need to write code to form A, then form M and to be able to apply the wavelet transforms.

Result for one-matrix case

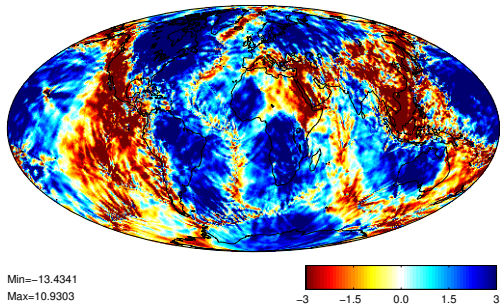
Sizes: A_1 (438674×3637248) is 115 GB; M_1 (438674×3637248) is 35 GB
$A_1 x$ versus $M_1 W^{-T} x$; $A_1^T y$ versus $W^{-1} M_1^T y$; and $A_1^T A_1 x$ versus $W^{-1} M_1^T M_1 W^{-T} x$ for 50 random vectors x and y.

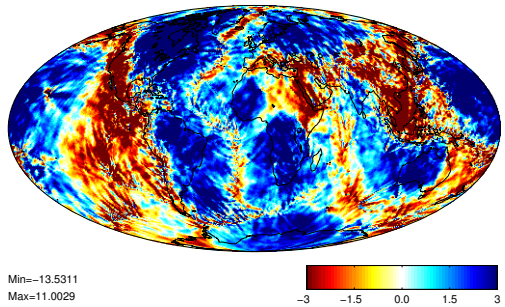


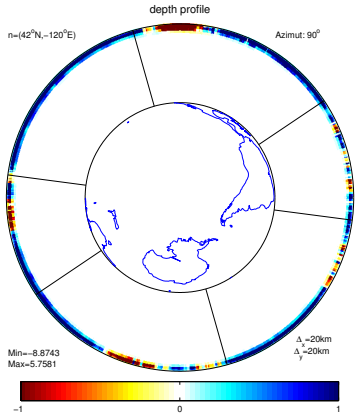
Errors in $A_1 x$ and $A_1^T y$ about 8%. Errors in $A_1^T A_1 x$ about 15%. Now we compare the solutions to:

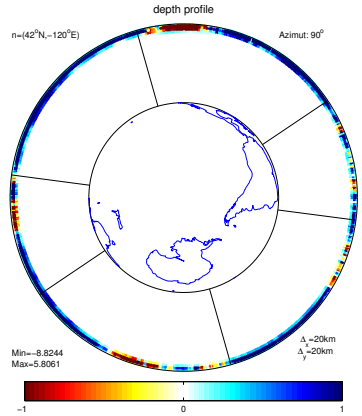
$$(A_1^T A_1 + \lambda I) x_1 = A_1^T b \quad \text{and} \quad (W^{-1} M_1^T M_1 W^{-T} + \lambda I) x_2 = W^{-1} M_1^T b$$

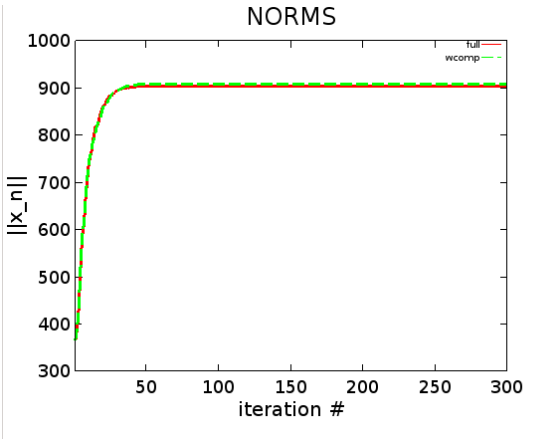
What really matters is the error in the operation $A_1^T A_1 x$.

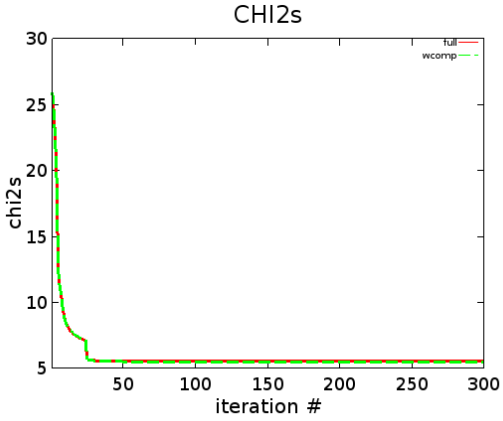

Min=−13.4341
Max=10.9303
−3
−1.5
0.0
1.5
3


Min=−13.5311
Max=11.0029
−3
−1.5
0.0
1.5
3


depth profile
n=(42°N,−120°E)
Azimut: 90°
Min=−8.8743
Max=5.7581
Δx=20km
Δy=20km
−1
0
1

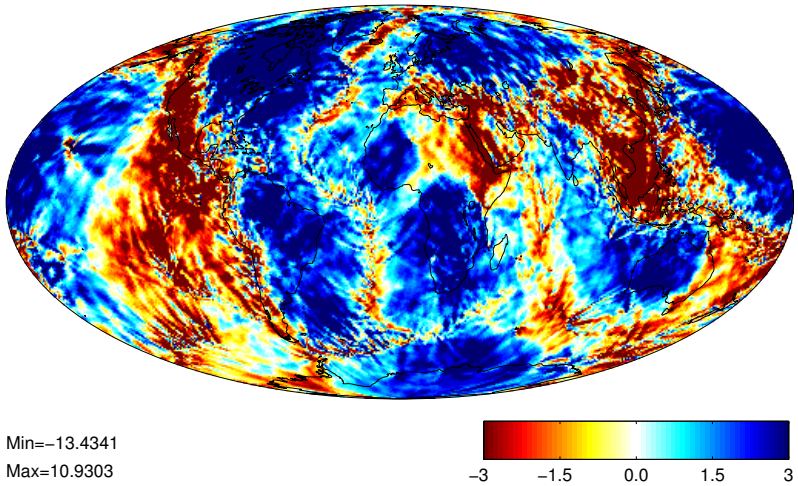

depth profile
n=(42°N,−120°E)
Azimut: 90°
Min=−8.8244
Max=5.8061
Δx=20km
Δy=20km
−1
0
1

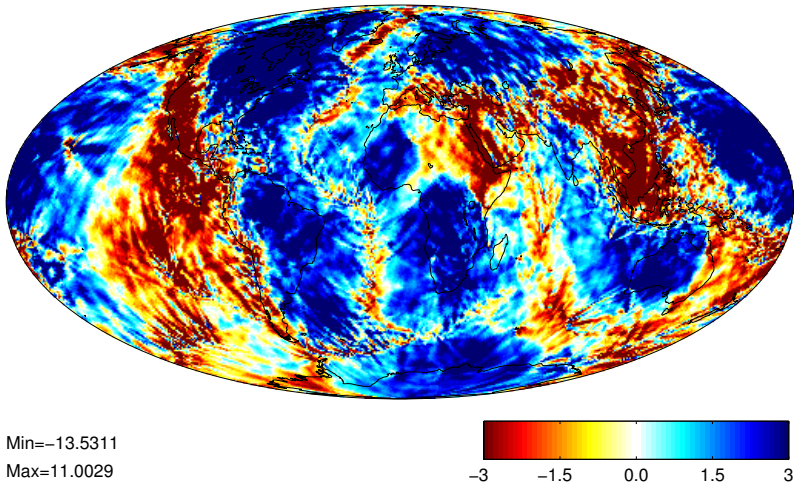

NORMS
||x_n||
iteration #
full
wcomp
1000
900
800
700
600
500
400
300
50
100
150
200
250
300


CHI2s
chi2s
iteration #
full
wcomp
30
25
20
15
10
5
50
100
150
200
250
300

Compression works well but only for individual blocks

Error at one depth about 5%; error at all depths about 12% (similar to $A_1^T A_1 x$ error) ; consequence of Tikhonov regularization





Using the whole system of 3 million rows

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{20} \end{bmatrix} \Longrightarrow Ax = \begin{bmatrix} A_1 x \\ A_2 x \\ \vdots \\ A_{20} x \end{bmatrix} \quad ; \quad A^T y = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{20} \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{20} \end{bmatrix} = \sum_{j=1}^{20} A_j^T y_j$$

approximated via:

$$M = \begin{bmatrix} Thr(A_1 W^T) \\ Thr(A_2 W^T) \\ \vdots \\ Thr(A_{20} W^T) \end{bmatrix} \Longrightarrow Ax \approx \begin{bmatrix} M_1 W^{-T} x \\ M_2 W^{-T} x \\ \vdots \\ M_{20} W^{-T} x \end{bmatrix} \quad ; \quad A^T y \approx \sum_{j=1}^{20} W_j^{-1} M_j^T y_j$$

For this, we first form the 20 submatrices $M_1, \ldots M_{20}$ by transforming and thresholding the rows of $A_1, \ldots, A_{20}$ (a lot of i/o).

Not enough compression for a $\approx$ 2 TB matrix

Want to take advantage of rapid singular value decay of A.

(3) Low rank decompositions and randomized algorithms for efficiently computing them.

Low Rank SVD

$$A = \begin{bmatrix} u_1 \dots u_r \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r \end{bmatrix} \begin{bmatrix} v_1^* \\ \vdots \\ {v_r}^* \end{bmatrix} = U \Sigma V^*$$

$$\approx U_k \Sigma_k V_k^* = \begin{bmatrix} u_1 \dots u_k \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_k \end{bmatrix} \begin{bmatrix} v_1^* \\ \vdots \\ {v_k}^* \end{bmatrix}$$

with $k << r \leq \min(m, n)$.

- A is sparse $m \times n$
- U_k is dense $m \times k$
- Σ_k is diagonal $k \times k$
- V_k is dense $n \times k$
- $\Longrightarrow$ if k is much smaller than $\min(m, n)$ one gets very substantial savings.

Randomized algorithm $\mathcal{O}(mnk)$, [Halko, Martinsson, Tropp]

- A is $m \times n$
- U_k is $m \times k$, $U_k^* U_k = I$; V_k is $n \times k$, $V_k^* V_k = I$; Σ_k is $k \times k$.

Sample range of A with $k+p$ lin. indep. vectors, so that $QQ^*A \approx A$.

- Draw an $n \times (k+p)$ Gaussian random matrix Ω.
  `Omega = randn(n,k+p)`
- Form the $m \times (k+p)$ sample matrix $Y = A\Omega$.
  `Y = A * Omega` ; $\mathsf{ran}Y \approx \mathsf{ran}A$
- Form an $m \times (k+p)$ orthonormal matrix Q such that $Y = QR$.
  `[Q, R] = qr(Y)` ; $\mathsf{ran}Q \approx \mathsf{ran}A$
- Form the $(k+p) \times n$ matrix Q^*A.
  `B = Q' * A`
- Compute the SVD of the smaller $(k+p) \times n$ matrix B: $B = \hat{U}\Sigma V^*$.
  `[Uhat, Sigma, V] = svd(B)`
- Form the matrix $U = Q\hat{U}$.
  `U = Q * Uhat` ; $QQ^*A \approx A$
- $U_k = U(:, 1:k), \Sigma_k = \Sigma(1:k, 1:k), V_k = V(:, 1:k)$.

Low rank SVD via SVD of $(k+p) \times (k+p)$ matrix

SVD of $B = Q^* A$ which is $(k+p) \times n$ is expensive, but we can work with $BB^* = Q^* A A^* Q$ which is $(k+p) \times (k+p)$.

$$B = U \Sigma V^* = \sum_{i=1}^{k} \sigma_i u_i v_i^* \quad \text{and} \quad B v_i = \sigma_i u_i$$

From which it follows that we can extract the eigenvectors u_i from BB^*:

$$BB^* = \left(\sum_{i=1}^{k} \sigma_i u_i v_i^* \right) \left(\sum_{j=1}^{k} \sigma_j u_j v_j^* \right)^* = \sum_{i,j=1}^{k} \sigma_i \sigma_j u_i v_i^* v_j u_j^* = \sum_{i=1}^{k} \sigma_i^2 u_i u_i^*$$

For the right eigenvectors v_i we can use:

$$\begin{aligned} B^* U = V \Sigma U^* U = V \Sigma \quad &\Longrightarrow \quad B^* U \Sigma^{-1} = V \\ &\Longrightarrow \quad v_i = V e_i = (B^* U \Sigma^{-1}) e_i = \frac{1}{\sigma_i} B^* u_i \end{aligned}$$

Power sampling scheme

$$(AA^*)^q A = U \Sigma^{(2q+1)} V^*$$

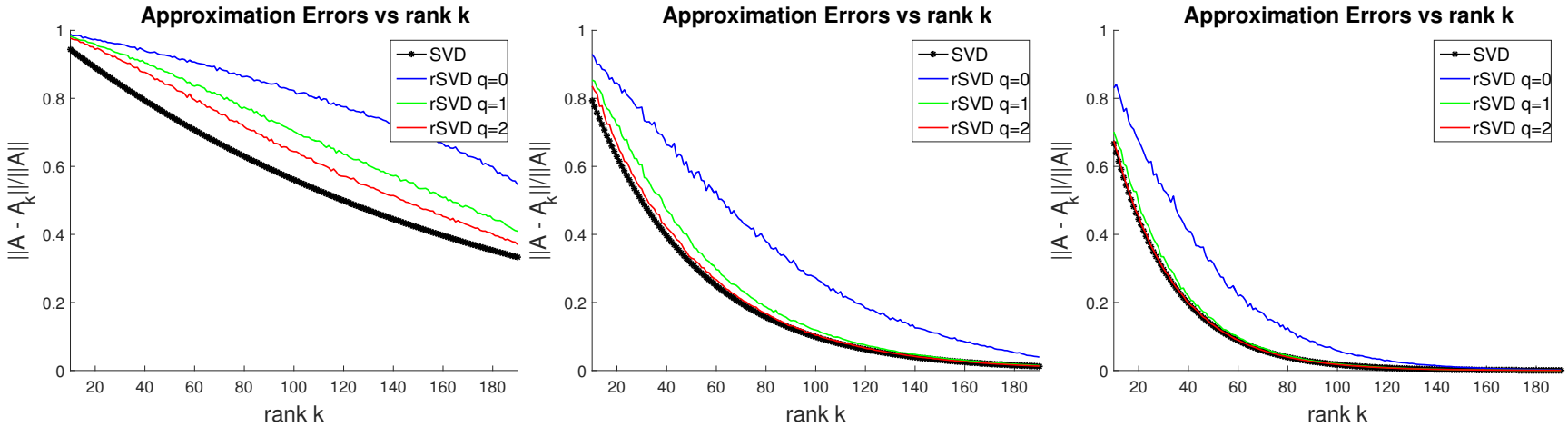
Eckart-Young Thm ; Power Sampling $(AA^*)^q A\Omega, q \geq 1$.

A an $m \times n$ matrix. For $1 \leq k \leq \min(m,n)$, the truncated SVD A_k gives:

$$\|A - A_k\|_2 = \sigma_{k+1} \quad ; \quad \|A - A_k\|_F = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2} \tag{1}$$

$$\mathbb{E}\|A - U_k \Sigma_k V_k^*\|_F = \left(1 + \frac{k}{p-1}\right)^{\frac{1}{2}} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)$$

$$\mathbb{E}\|A - U_k \Sigma_k V_k^*\|_2 = \left(1 + \sqrt{\frac{k}{p-1}}\right) \sigma_{k+1} + \left(\frac{e\sqrt{k+p}}{p}\right) \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)$$



When $Y = A\Omega$ captures the range of A, then $QQ^*A = A$.

(1) $\mathcal{R}(A) \subseteq \mathcal{R}(Q)$ (the range of A is a subset of the range of Q)

(2) $A = QQ^*A$

As k approaches the rank of A, the approximation QQ^*A approaches A.

$$A = U \begin{array}{c} \begin{matrix} k & n-k \end{matrix} \\ \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \end{array} \begin{array}{c} \begin{matrix} k \\ n-k \end{matrix} \\ \begin{bmatrix} V_1^* \\ V_2^* \end{bmatrix} \end{array} \begin{matrix} \\ n \end{matrix}.$$

Let $\Omega_1 = V_1^*\Omega$ and $\Omega_2 = V_2^*\Omega$; set $Y = A\Omega$ and $Q = orth(Y)$. Then [HMT]:

$$\|(I - QQ^*)A\| \leq \|\Sigma_2\|^2 + \|\Sigma_2\Omega_2\Omega_1^\dagger\|^2. \tag{2}$$

Everything works if we can construct Q s.t. $\|QQ^*A - A\| < \epsilon$:

$$\boxed{B = Q^*A, \quad \|A - QB\| < \epsilon,} \tag{3}$$

We can form different approximate low rank factorizations from B:

$$B = \tilde{U}DV^* \implies A \approx (Q\tilde{U})DV^*$$

$$BP = \tilde{Q}R \implies AP \approx (Q\tilde{Q})R$$

Constructing Q given $\epsilon > 0$ [Martinsson, V., 2015]

(1) $\mathbf{Q}_0 = [\,]$; $\mathbf{B}_0 = [\,]$; $\mathbf{A}_0 = \mathbf{A}$; $j = 0$;
(2) **while** $\|\mathbf{A}^{(j)}\| > \varepsilon$
(3) $\quad j = j + 1$
(4) $\quad$ Pick a unit vector $\mathbf{q}_j \in \mathsf{ran}(\mathbf{A}^{(j-1)})$.
(5) $\quad \mathbf{b}_j = \mathbf{q}_j^* \mathbf{A}^{(j-1)}$
(6) $\quad \mathbf{Q}_j = [\mathbf{Q}_{j-1}\ \mathbf{q}_j]$
(7) $\quad \mathbf{B}_j = \begin{bmatrix} \mathbf{B}_{j-1} \\ \mathbf{b}_j \end{bmatrix}$
(8) $\quad \mathbf{A}^{(j)} = \mathbf{A}^{(j-1)} - \mathbf{q}_j \mathbf{b}_j$
(9) **end while**
(10) $k = j$.

After first iteration, $A^{(1)} = A^{(0)} - q_1 b_1 = A - q_1 q_1^* A = (I - Q_1 Q_1^*)A$. Next, $q_2 \in \mathsf{ran}(A^{(1)}) = \mathsf{ran}((I - q_1 q_1^*)A) \in \mathsf{ran}(I - q_1 q_1^*)$ so $q_2^* q_1 = 0$ and $A^{(2)} = A^{(1)} - q_2 b_2 = A - q_1 q_1^* A - q_2 q_2^* A = (I - Q_2 Q_2^*)A$. At the end of iteration j, we have:

$$A^{(j)} = (I - Q_j Q_j^*)A \quad \text{and} \quad B_j = Q_j^* A \tag{4}$$

When $\|A^{(j)}\| < \epsilon$, we have Q_j s.t. $\|Q_j Q_j^* A - A\| < \epsilon$.

A block algorithm (add many vectors at a time)

(1) **for** $i = 1, 2, 3, \ldots$
(2) $\quad \mathbf{\Omega}_i = \texttt{randn}(n, b)$.
(3) $\quad \mathbf{Q}_i = \texttt{orth}(\mathbf{A}\mathbf{\Omega}_i)$.
(4) $\quad$ **for** $j = 1 : P$
(5) $\quad\quad \mathbf{Q}_i = \texttt{orth}(\mathbf{A}^*\mathbf{Q}_i)$.
(6) $\quad\quad \mathbf{Q}_i = \texttt{orth}(\mathbf{A}\mathbf{Q}_i)$.
(7) $\quad$ **end for**
(8) $\quad \mathbf{Q}_i = \texttt{orth}(\mathbf{Q}_i - \sum_{j=1}^{i-1} \mathbf{Q}_j\mathbf{Q}_j^*\mathbf{Q}_i)$
(9) $\quad \mathbf{B}_i = \mathbf{Q}_i^*\mathbf{A}$
(10) $\quad \mathbf{A} = \mathbf{A} - \mathbf{Q}_i\mathbf{B}_i$
(11) $\quad$ **if** $\|\mathbf{A}\| < \varepsilon$ **then stop**
(12) **end while**
(13) Set $\mathbf{Q} = [\mathbf{Q}_1 \ \cdots \ \mathbf{Q}_i]$ and $\mathbf{B} = [\mathbf{B}_1^* \ \cdots \ \mathbf{B}_i^*]^*$.

At the end of iteration j, we have as before:

$$A^{(j)} = (I - Q_jQ_j^*)A \quad \text{and} \quad B_j = Q_j^*A \tag{5}$$

When $\|A^{(j)}\| < \epsilon$, we have Q s.t. $\|QQ^*A - A\| < \epsilon$. Algorithm increases matrix multi cost but *decreases* QR factorization cost.

Ideas for very large matrices

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} \approx \begin{bmatrix} Q_1 B_1 \\ Q_2 B_2 \\ Q_3 B_3 \\ Q_4 B_4 \end{bmatrix} = \begin{bmatrix} Q_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & Q_3 & 0 \\ 0 & 0 & 0 & Q_4 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix}$$

We then perform QB factorizations on the blocks of the B matrix:

$$M^{(1)} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \approx Q_{12} B_{12} \quad ; \quad M^{(2)} = \begin{bmatrix} B_3 \\ B_4 \end{bmatrix} \approx Q_{34} B_{34}$$

Finally, we perform a QB factorization on:

$$M^{(3)} = \begin{bmatrix} B_{12} \\ B_{34} \end{bmatrix} \approx Q_{1234} B_{1234}$$

It follows that:

$$\begin{aligned} A &\approx \begin{bmatrix} Q_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & Q_3 & 0 \\ 0 & 0 & 0 & Q_4 \end{bmatrix} \begin{bmatrix} Q_{12} & 0 \\ 0 & Q_{34} \end{bmatrix} \begin{bmatrix} B_{12} \\ B_{34} \end{bmatrix} \approx \begin{bmatrix} Q_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & Q_3 & 0 \\ 0 & 0 & 0 & Q_4 \end{bmatrix} \begin{bmatrix} Q_{12} & 0 \\ 0 & Q_{34} \end{bmatrix} Q_{1234} B_{1234} \\ &= Q^{(3)} Q^{(2)} Q^{(1)} B^{(1)} = QB \end{aligned}$$

Rank k pivoted QR factorization.

$$\underset{m\times n}{A}\quad\underset{n\times n}{P}\quad=\quad\underset{m\times r}{Q}\quad\underset{r\times n}{S},\tag{6}$$

where P is a permutation matrix, Q has orthonormal columns, and S is upper triangular and $AP = A(:, J_c)$. We can stop after the first k iterations of the algorithm, obtaining:

$$A(:,J_c) \quad = \quad \overset{m}{} \begin{bmatrix} \overset{k}{Q_1} & \overset{r-k}{Q_2} \end{bmatrix} \times \begin{matrix} \overset{k}{} \\ \underset{r-k}{} \end{matrix} \begin{bmatrix} \overset{n}{S_1} \\ S_2 \end{bmatrix} = Q_1S_1 + Q_2S_2. \tag{7}$$

$$S_1 = \overset{k}{} \begin{bmatrix} \overset{k}{S_{11}} & \overset{n-k}{S_{12}} \end{bmatrix} \quad \text{and} \quad S_2 = \overset{k}{} \begin{bmatrix} \overset{k}{0} & \overset{n-k}{S_{22}} \end{bmatrix}, \tag{8}$$

$$(\text{i.e., } S = \begin{matrix} \overset{k}{} \\ \underset{r-k}{} \end{matrix} \begin{bmatrix} \overset{k}{S_{11}} & \overset{n-k}{S_{12}} \\ 0 & S_{22} \end{bmatrix}, \ ) \tag{9}$$

$$\begin{aligned} A(:,J_c) \quad &= \quad Q_1 \begin{bmatrix} S_{11} & S_{12} \end{bmatrix} + Q_2 \begin{bmatrix} 0 & S_{22} \end{bmatrix} \\ &= \quad \overset{m}{} \begin{bmatrix} \overset{k}{Q_1S_{11}} & \overset{n-k}{Q_1S_{12} + Q_2S_{22}} \end{bmatrix}. \end{aligned}$$

Rank k ID and tsID factorizations.

$$C := A(:, J_c(1:k)) = Q_1 S_{11}.$$

$$Q_1 S_1 = \begin{bmatrix} Q_1 S_{11} & Q_1 S_{12} \end{bmatrix} = Q_1 S_{11} [I_k \quad S_{11}^{-1} S_{12}] = C \, [I_k \quad T_l],$$

where T_l is the solution to the matrix equation $S_{11} T_l = S_{12}$ which can be solved for T_l a column at a time.

$$A \approx C V^*, \quad \text{where} \quad V^* = \begin{bmatrix} I_k & T_l \end{bmatrix} P^*. \tag{10}$$

The **one sided ID of (rank k)** is the approximate factorization:

$$\underset{m \times n}{A} \approx \underset{m \times k}{A(:, J_c(1:k))} \; \underset{k \times n}{V^*}, \tag{11}$$

where we use a *partial column skeleton* $C = A(:, J_c(1:k))$ of a subset of the columns of A and V is a well-conditioned matrix.

The **two sided ID of (rank k)**

$$\underset{m \times n}{A} \approx \underset{m \times k}{W} \; \underset{k \times k}{A(J_r(1:k), J_c(1:k))} \; \underset{k \times n}{V^*}, \tag{12}$$

is obtained via two successive one sided rank k ID computations.

Column norm preservation

Lemma

Let $\tilde{\Omega} \in \mathbb{R}^{l \times m}$ *be a matrix with GIID entries. Then for any* $a \in \mathbb{R}^m$ *we have that* $E\left[\frac{\|\tilde{\Omega}a\|^2}{\|a\|^2}\right] = l$ *and* $Var\left[\frac{\|\tilde{\Omega}a\|^2}{\|a\|^2}\right] = 2l$.

$\Rightarrow$ Suppose A is $m \times n$ and we draw an $l \times m$ GIID matrix $\tilde{\Omega}$. Suppose we then form the $l \times n$ matrix $Z = \tilde{\Omega}A$. Then, $E\left[\frac{\|Z(:,j)\|^2}{\|A(:,j)\|^2}\right] = l$.

Randomized algorithm $\mathcal{O}(mnk)$, [Voronin, Martinsson, 2015]

Input : $\mathbf{A} \in \mathbb{R}^{m \times n}$, a rank parameter $k < \min(m, n)$, and an oversampling parameter p.
Output: A column index set J and a matrix $\mathbf{V} \in \mathbb{R}^{n \times k}$ (such that $\mathbf{A} \approx \mathbf{A}(:, J(1:k))\mathbf{V}^*$).

1 Construct a random matrix $\mathbf{\Omega} \in \mathbb{R}^{(k+p) \times m}$ with i.i.d. Gaussian entries;
2 Construct the sample matrix $\mathbf{Y} = \mathbf{\Omega A}$;
3 Perform full pivoted QR factorization on $\mathbf{Y}$ to get: $\mathbf{YP} = \mathbf{QS}$;
4 Remove p columns of $\mathbf{Q}$ and p rows of $\mathbf{S}$ to construct $\mathbf{Q}_1$ and $\mathbf{S}_1$;
5 Define the ordered index set J via $\mathbf{I}(:, J) = \mathbf{P}$;
6 Partition $\mathbf{S}_1$: $\mathbf{S}_{11} = \mathbf{S}_1(:, 1:k)$, $\mathbf{S}_{12} = \mathbf{S}_1(:, k+1:n)$;
7 $\mathbf{V} = \mathbf{P}\begin{bmatrix}\mathbf{I}_k & \mathbf{S}_{11}^{-1}\mathbf{S}_{12}\end{bmatrix}^*$;

Rank k CUR factorization.

The two sided ID allows us to construct the popular **Column/Row skeleton CUR (rank k)** decomposition:

$$\underset{m\times n}{A} \approx \underset{m\times k}{C}\ \underset{k\times k}{U}\ \underset{k\times n}{R}, \tag{13}$$

Suppose we compute a two sided rank k ID factorization forming the $k \times k$ column/row skeleton $A(J_r(1:k), J_c(1:k))$. Set:

$$C = A(:, J_c(1:k)) \quad \text{and} \quad R = A(J_r(1:k), :)$$

We then set this to equal the factors C and R in CUR:

$$CUR \ = \ A(:, J_c(1:k))UA(J_r(1:k), :) \approx A(:, J_c(1:k))V^* \tag{14}$$

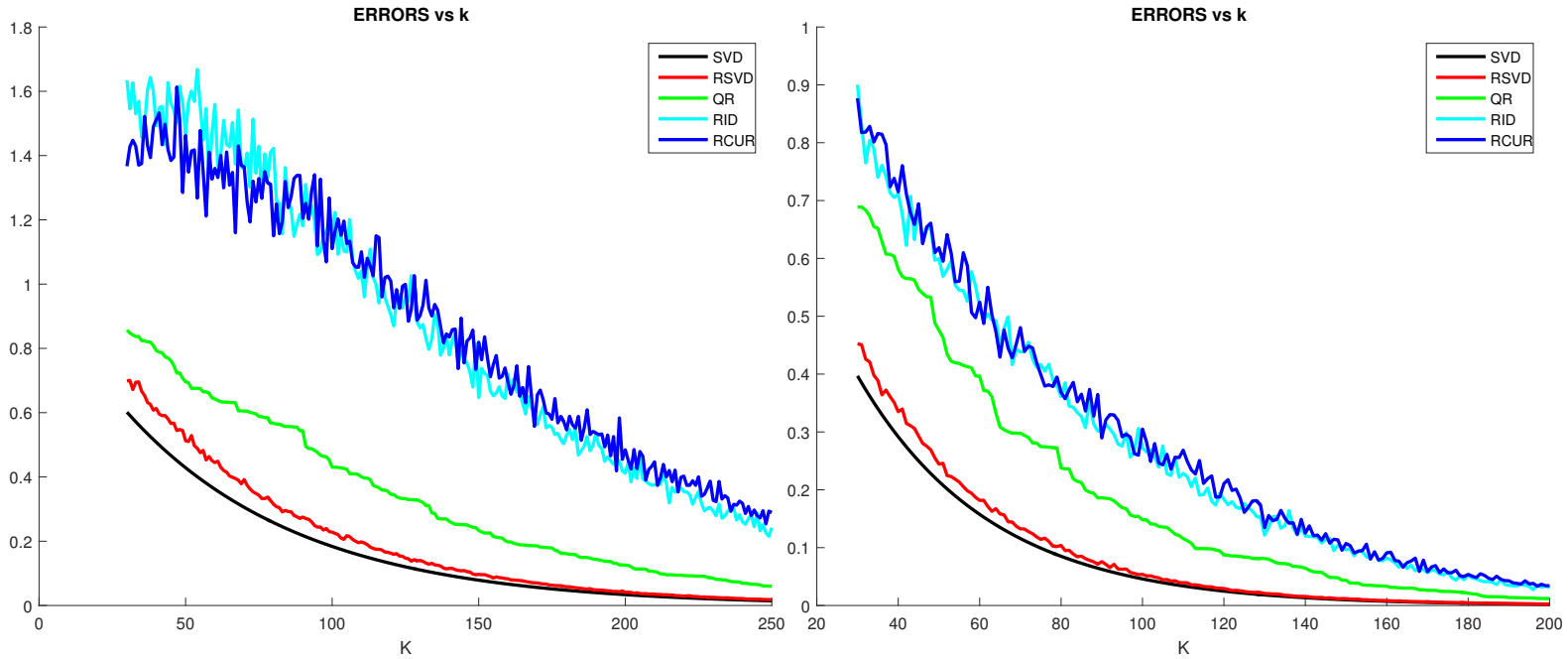
where we take U to satisfy the system:

$$UR = V^*, \tag{15}$$

Interestingly, this scheme (non-randomized) often has better error than the partial QR.

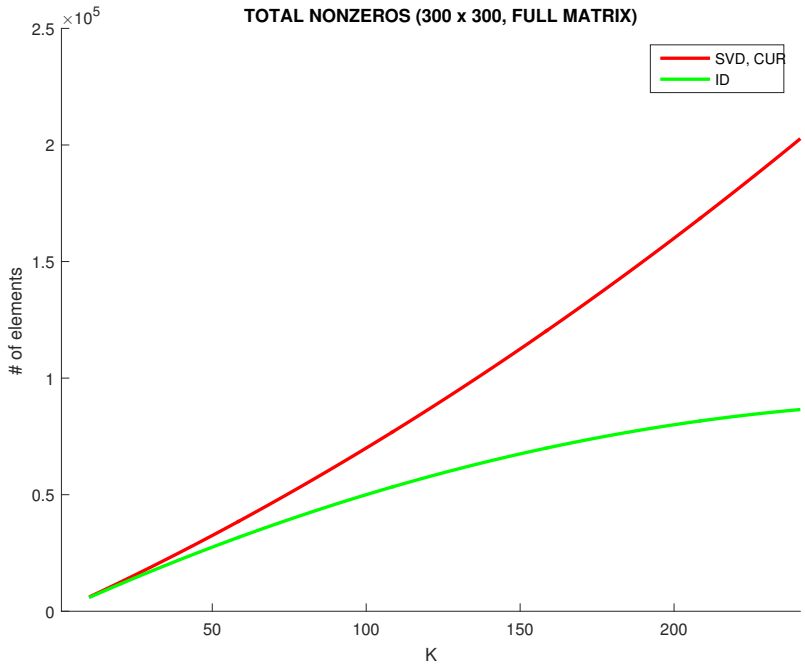
Compare rank k SVD, QR, rand CUR errors

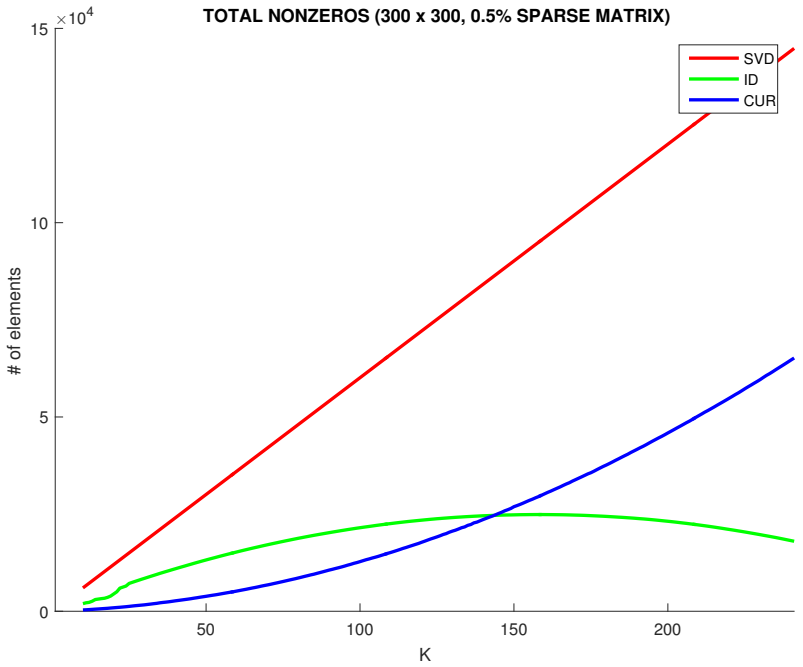
300×300 matrix, $p = 10$, $q = 2$.



- Errors can be far from optimal.
- For sparse A, ID and CUR require less storage than SVD.

Storage sizes for full and sparse matrices





(4) Application to inverse problems (model order reduction).

Low rank SVD $U_k S_k V_k^* \approx A$ for model reduction

Classic Tikhonov problem: $\tilde{x} = \arg\min_x \left\{ \|Ax - b\|_2^2 + \tau \|x\|_2^2 + \lambda \|Lx\|_2^2 \right\}$.

$$\Rightarrow \left(A^* A + \tau I + \lambda L^* L\right) \tilde{x} = A^* b$$

Multiply on the left by V_k^* and choose solution of the form $\tilde{x} = V_k \tilde{y}$:

$$\begin{aligned} & V_k^* \left(A^* A + \tau I + \lambda L^* L\right) V_k \tilde{y} &&= V_k^* A^* b \\ \Longleftrightarrow \quad & \left(\Sigma_k^2 + \tau I + \lambda V_k^* L^* L V_k\right) \tilde{y} &&= \Sigma_k U_k^* b \\ & && (\because V_k^* A^* A V_k = V_k^* A_k^* A_k V_k = \Sigma_k^2). \end{aligned}$$

- Notice that $\tilde{x}$ has **n** variables and $\tilde{y}$ has **k** variables, with $k \ll n$.
- When $\lambda = 0$, equivalent to replacing A by $U_k \Sigma_k V_k^*$ above:

$$\left(V_k \Sigma_k^2 V_k^* + \tau I\right) \tilde{x} = V_k \Sigma_k U_k^* b$$

- Why set $\tilde{x} = V_k \tilde{y}$? Since we want $A\tilde{x} \approx U_k S_k V_k^* \tilde{x} \approx b$:

$$\Rightarrow U_k \Sigma_k V_k^* \tilde{x} \approx b \Rightarrow U_k \Sigma_k v \approx b \Rightarrow \text{set } x = V_k v \Rightarrow U_k \Sigma_k (V_k^* V_k) v \approx b$$

Low rank QB decomposition $Q_k B_k \approx A$ for model reduction

Classic ℓ_1-min problem: $\bar{x} = \arg\min_x \left\{ \|Ax - b\|_2^2 + \tau \|x\|_1 \right\}$.

$$x^{n+1} = \mathbb{S}_{\frac{\tau}{2}} \left(x^n - A^*(Ax^n - b) \right)$$

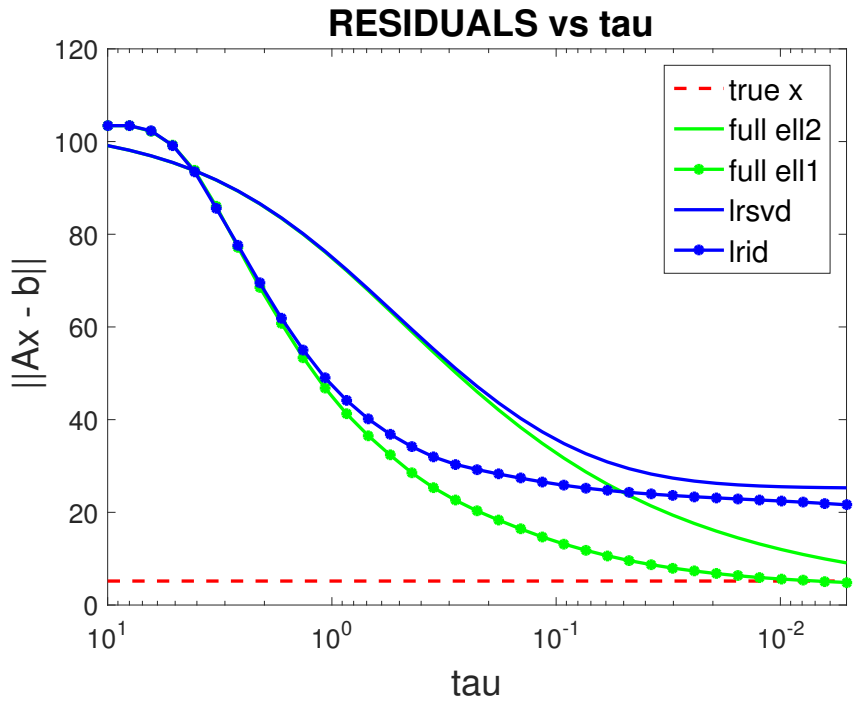
$$\underset{m \times n}{A} \approx \underset{m \times k}{Q_k} \underset{k \times n}{B_k}, \quad \Rightarrow Q_k B_k x \approx b \Rightarrow Q_k y \approx b$$

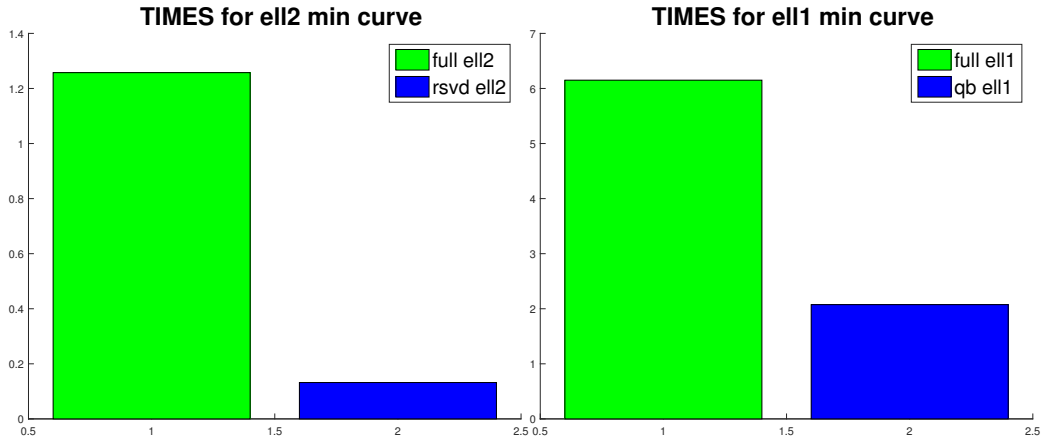
- Set $\boxed{y = Q_k^* b}$ (x has **n** variables, y has **k** variables).
- Solve $y = B_k x$ via ℓ_1 minimization (cheaper since B_k is $k \times n$).

$$\boxed{x^{n+1} = \mathbb{S}_{\frac{\tau}{2}} \left(x^n - B_k^*(B_k x^n - y) \right)}$$

- Improve solution using a few iterations with A and b in place of B_k and y.

Construction of approximate L curves (1000×1500, $k = 300$)





Application to large scale problems [Voronin et al., 2015]

Break A into blocks, compress each block, get low rank SVD of each block, then get overall low rank SVD of compressed version:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{bmatrix} \rightarrow M = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_N \end{bmatrix} = \begin{bmatrix} \mathbb{T}(A_1 W_1^T) \\ \mathbb{T}(A_2 W_2^T) \\ \vdots \\ \mathbb{T}(A_N W_N^T) \end{bmatrix} \Rightarrow A \approx \begin{bmatrix} U_{k_1} \Sigma_{k_1} V_{k_1}^T \\ U_{k_2} \Sigma_{k_2} V_{k_2}^T \\ \vdots \\ U_{k_N} \Sigma_{k_N} V_{k_N}^T \end{bmatrix} \approx U_k \Sigma_k V_k^T$$

- Form Q_r, B_r of each block using matrix-vector operations with M_r.
- Form $(k+p) \times (k+p)$ matrix $B_r B_r^T$ column by column ($B_r B_r^T = Q_r^T A_r A_r^T Q_r$)

$$B_r B_r^T e_j = Q_r^T A_r A_r^T Q_r e_j \approx Q_r^T M_r W_r^{-T} W_r^{-1} M_r^T Q e_j$$

- Compute eigendecomposition of $B_r B_r^T$ to get $\tilde{U}$ and D.
- Compute $\Sigma_{i,i} = \sqrt{D_{i,i}}$
- Compute $U = Q\tilde{U}$ (since $QQ^T A = QB \approx A$)
- Compute $v_j = \frac{1}{\sigma_j} A^T U e_j \approx \frac{1}{\sigma_j} W^{-1} M^T U e_j$ as columns of V.

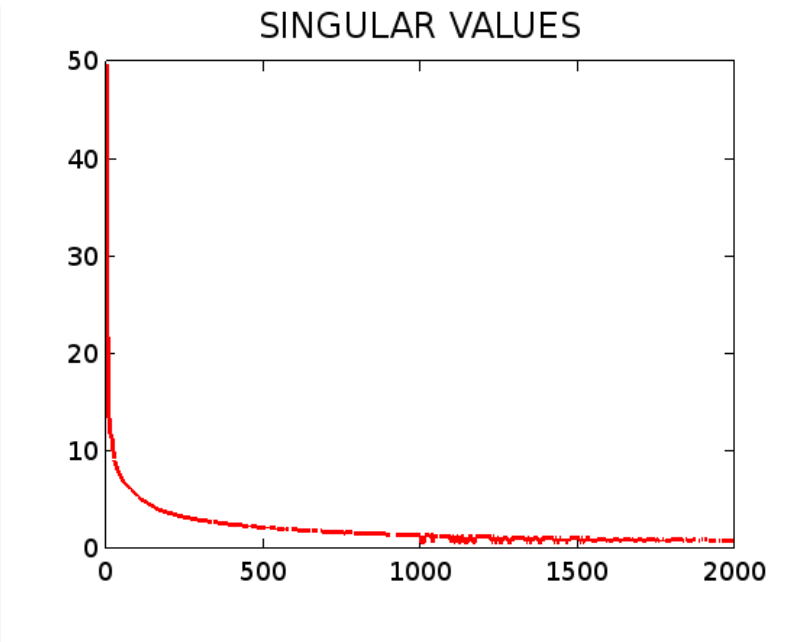
Sizes with $k = 2000$ for small and large matrix

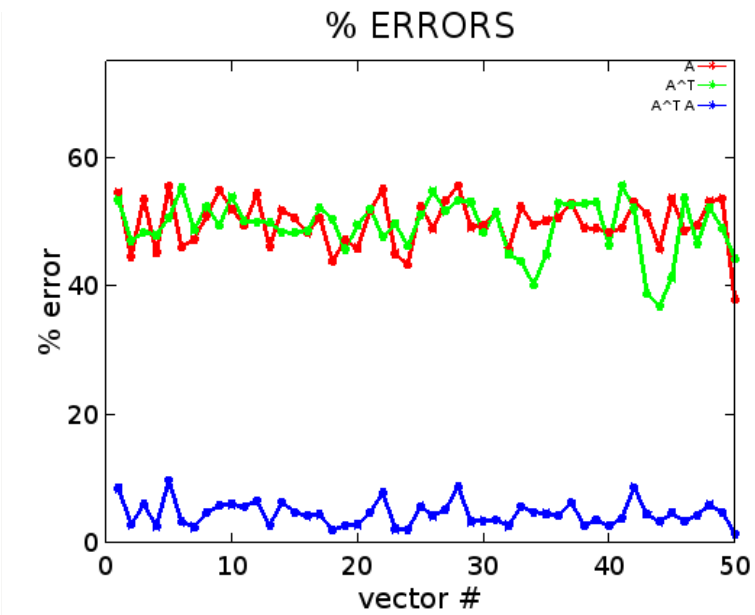
- A_1, dimensions (438674×3637248), size is 115 GB
- M_1, dimensions (438674×3637248), size is 35 GB
- U_{1_k}, Σ_{1_k}, V_{1_k}, dimensions $(438674 \times 2000), (2000 \times 2000), (3637248 \times 2000)$, sizes are 7 GB, 30 MB, 55 GB ($\approx$ 62 GB total)
- A, dimensions (2968933×3637248), size is 3.2 TB (approximate, never computed)
- M, dimensions (2968933×3637248), size is 1 TB
- U_k, Σ_k, V_k, dimensions $(2968933 \times 2000), (2000 \times 2000), (3637248 \times 2000)$, sizes are 45 GB, 30 MB, 55 GB ($\approx$ 100 GB total)

For small matrix (115 GB), SVD offers no compression (wavelet compression better).
For large matrix (3.5 TB), SVD offers $30x$ compression.
Results in double precision; factor of 2 savings possible.

Approximate Matrix-Vector Operations (smaller matrix)

$A_1 x \approx U_{k_1} \Sigma_{k_1} V_{k_1}^T x$, $A_1^T y \approx V_{k_1} \Sigma_{k_1} U_{k_1}^T$, $A_1^T A_1 \approx V_{k_1} \Sigma_{k_1}^2 V_{k_1}^T$.

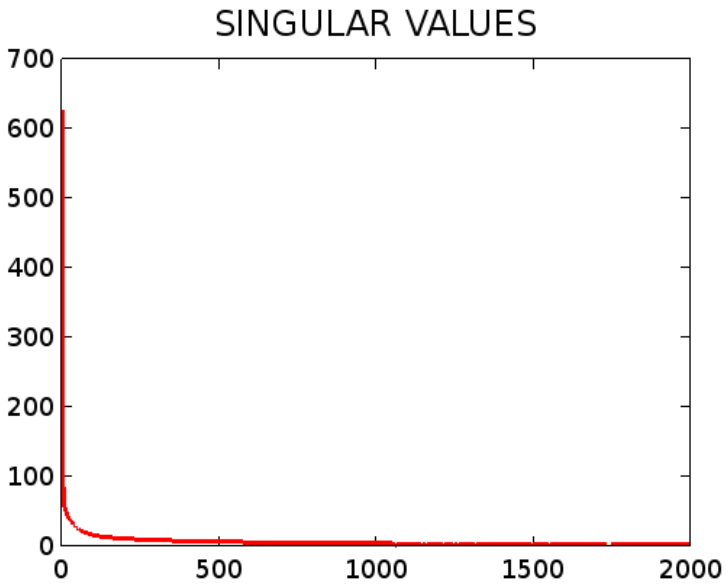


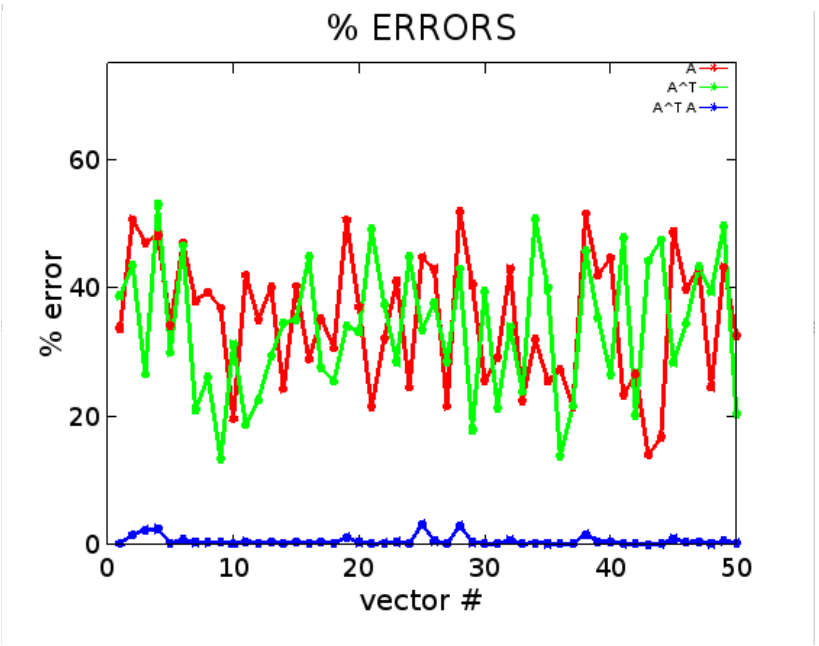


Smaller matrix not so badly conditioned. Errors in $A_1 x$ and $A_1^T y$ approximations are very high. Errors in $A_1^T A_1 x$ approximation are acceptable.

Approximate Matrix-Vector Operations (big matrix)

$Ax \approx U_k \Sigma_k V_k^T x$, $A^T y \approx V_k \Sigma_k U_k^T$, $A^T A \approx V_k \Sigma_k^2 V_k^T$.





Larger matrix is worser conditioned. Errors in Ax and $A^T y$ approximations are high. Errors in $A^T Ax$ approximation are very small.

Observations

- $A^T A$ is well approximated via low rank SVD.
- A is not well approximated unless a high enough rank k is used.

Approximate Regularization Schemes [Voronin 2015]

$$(A^T A + \lambda_1 I + \lambda_2 L^T L)\bar{x} = A^T b$$

Get the schemes:

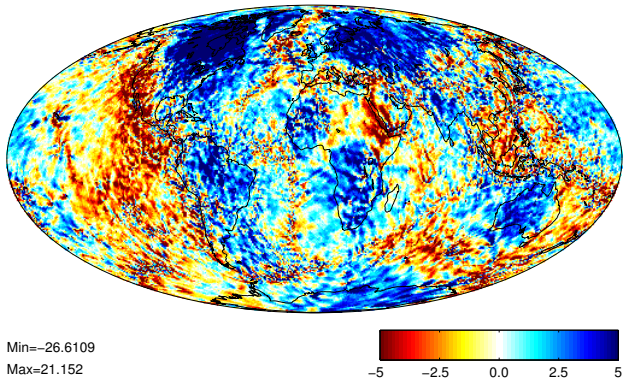
$$(V_k \Sigma_k^2 V_k^T + \lambda_1 I + \lambda_2 L^T L)\tilde{x_1} = V_k \Sigma_k U_k^T b$$
$$(V_k \Sigma_k^2 V_k^T + \lambda_1 I + \lambda_2 L^T L)\hat{x_1} = A^T b$$

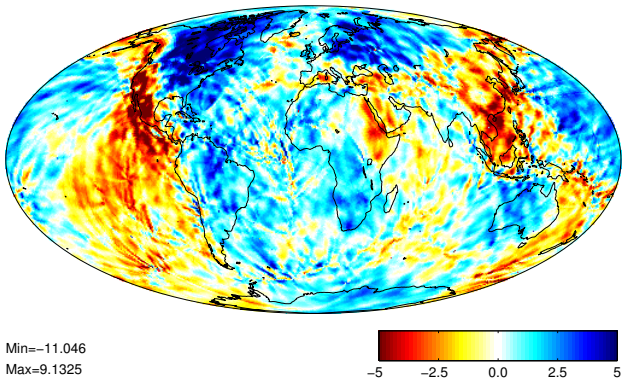
Can get explicit error bounds when $\lambda_1 = \lambda$ and $\lambda_2 = 0$:

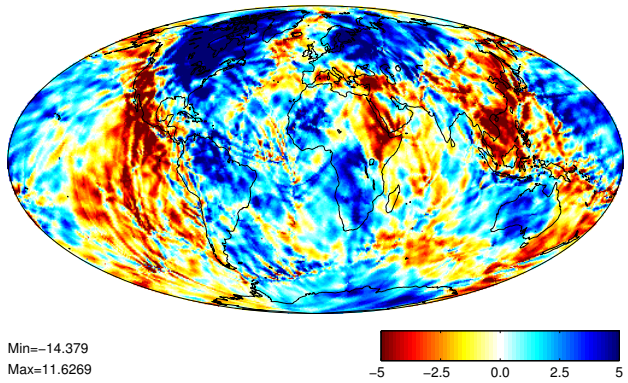
$$\|\bar{x} - \tilde{x_1}\|_2 \leq \frac{\sigma_{k+1}}{\lambda + \sigma_{k+1}^2} \|b\|_2$$

$$\|\bar{x} - \hat{x_1}\|_2 \leq \frac{\sigma_{k+1}^3}{\lambda\left(\lambda + \sigma_{k+1}^2\right)} \|b\|_2$$

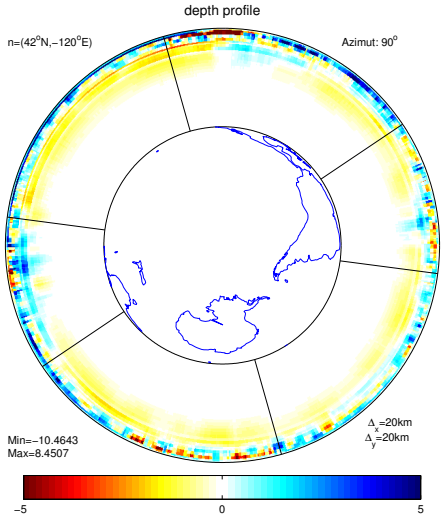
Low rank SVD solutions for A

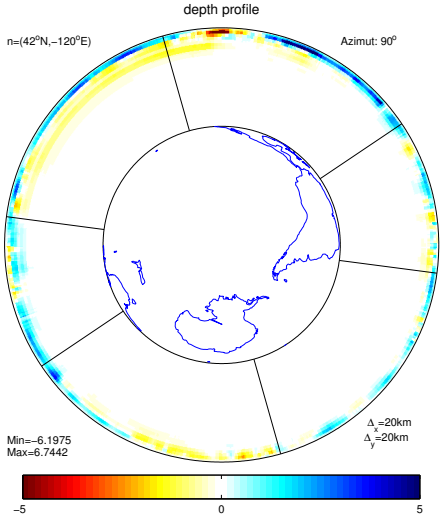


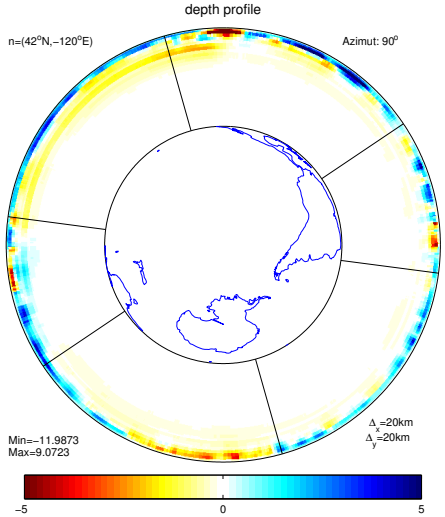




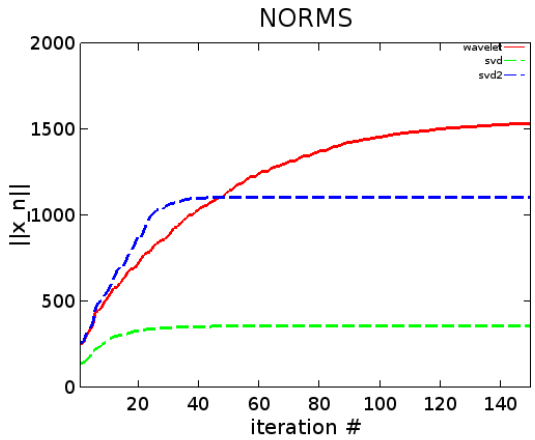
Low rank SVD solutions for A

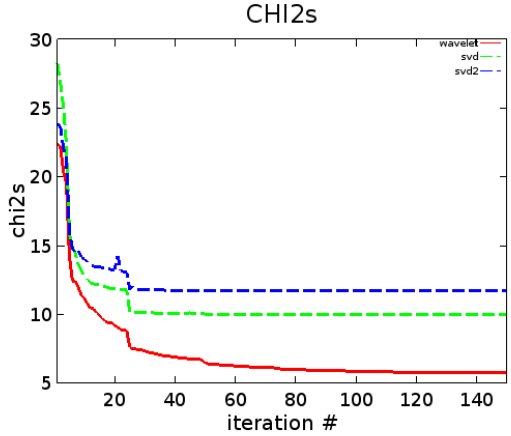


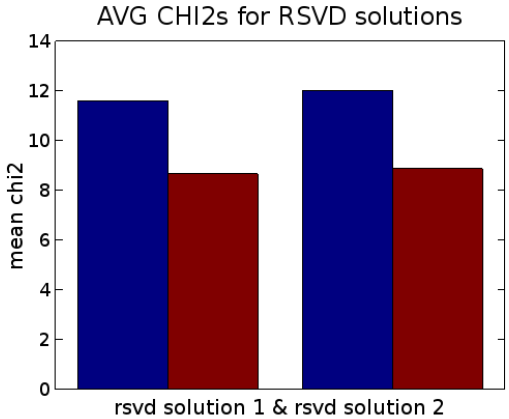




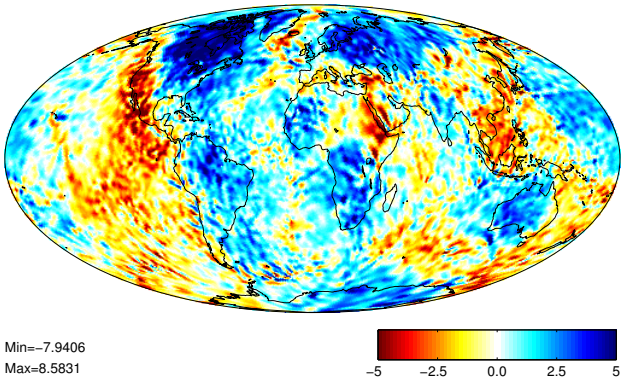
Low rank SVD solutions for A

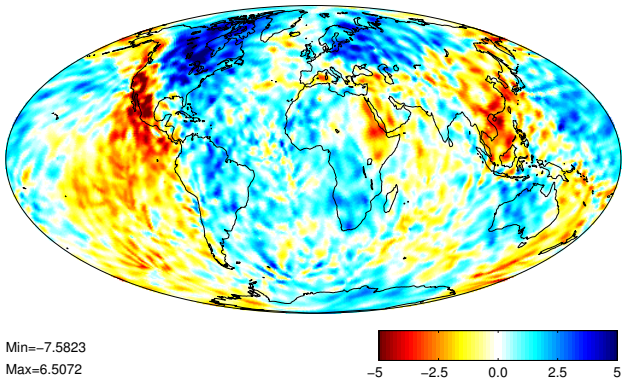


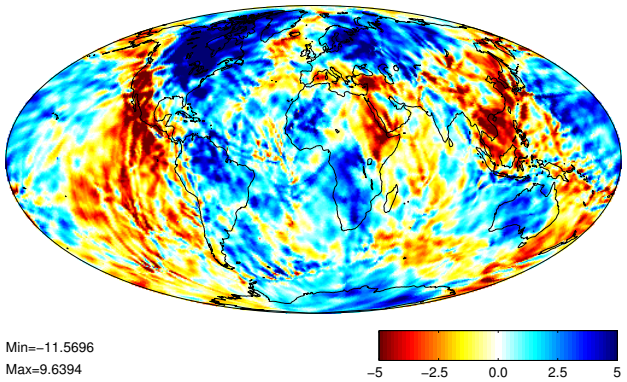




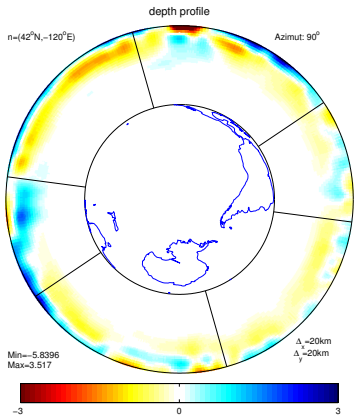
Low rank SVD solutions for A with Laplacian

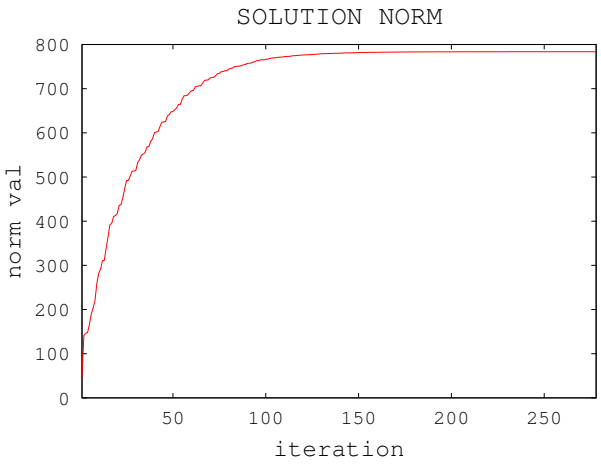


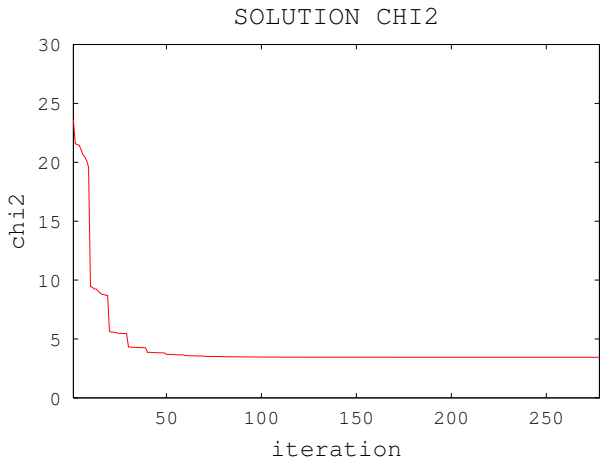


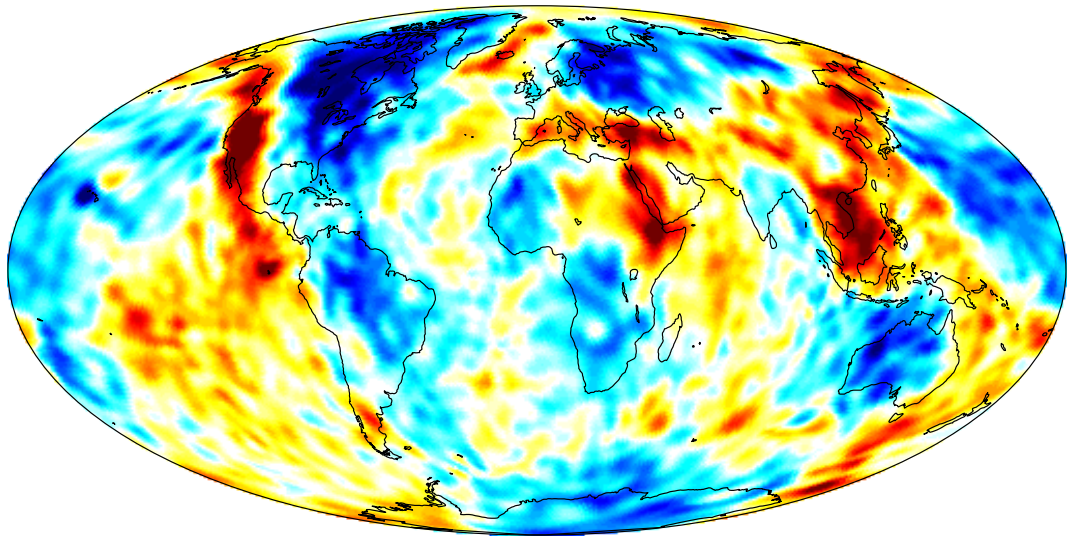


Solution with more Laplacian smoothing and correction terms



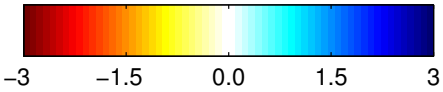




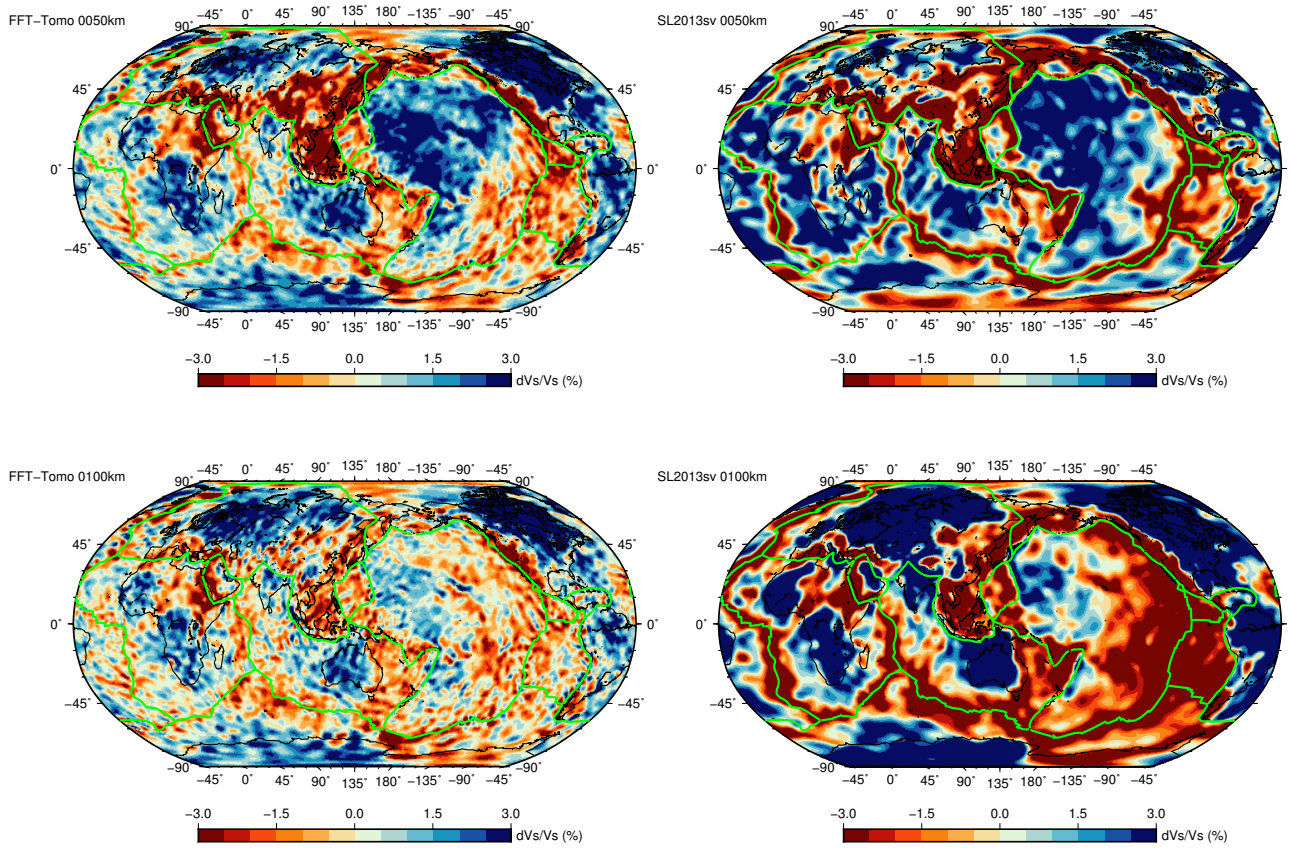


Min=−4.8722

Max=3.7114



Comparison with (Schaeffer and Lebedev, 2013) at 50, 100 km depth



We recover similar features close to surface. Data sets for matrices are different so direct comparison is not possible.

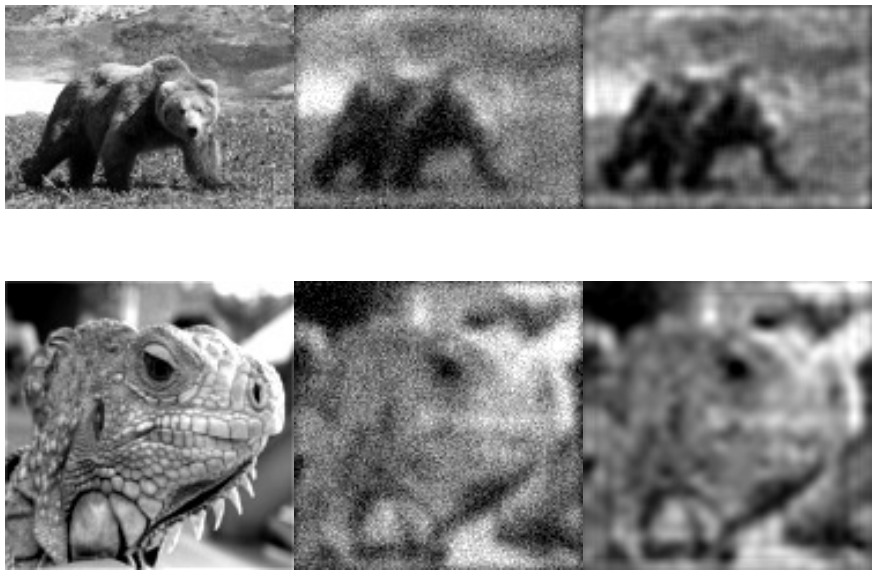
Even more compression [with G. Nolet and D. Mikesell, 2014]

Use compressed system with automatic clustering:

$$\begin{bmatrix} U_{k_1}^T A_1 \\ U_{k_2}^T A_2 \\ \vdots \\ U_{k_N}^T A_N \end{bmatrix} x = \begin{bmatrix} U_{k_1}^T b_1 \\ U_{k_2}^T b_2 \\ \vdots \\ U_{k_p}^T b_N \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \Sigma_{k_1} V_{k_1}^T \\ \Sigma_{k_2} V_{k_2}^T \\ \vdots \\ \Sigma_{k_N} V_{k_N}^T \end{bmatrix} x = \begin{bmatrix} U_{k_1}^T b_1 \\ U_{k_2}^T b_2 \\ \vdots \\ U_{k_N}^T b_N \end{bmatrix}$$

By clustering matrix rows into blocks with overlapping nonzero patterns, very high compression ratios are possible. For 3 million rows, we used about $20,000$ blocks. Need to calculate SVD components of individual blocks only, which does not present computational challenge.

Applications to image deblurring and denoising

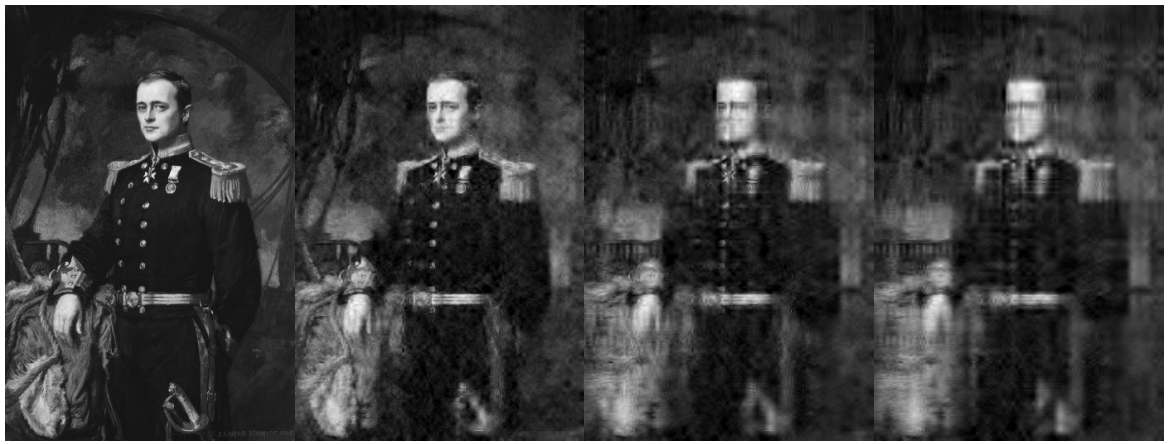


Apply Wiener filter, then apply low rank inverse Toeplitz matrices:

Using sparsity preserving CUR for image compression

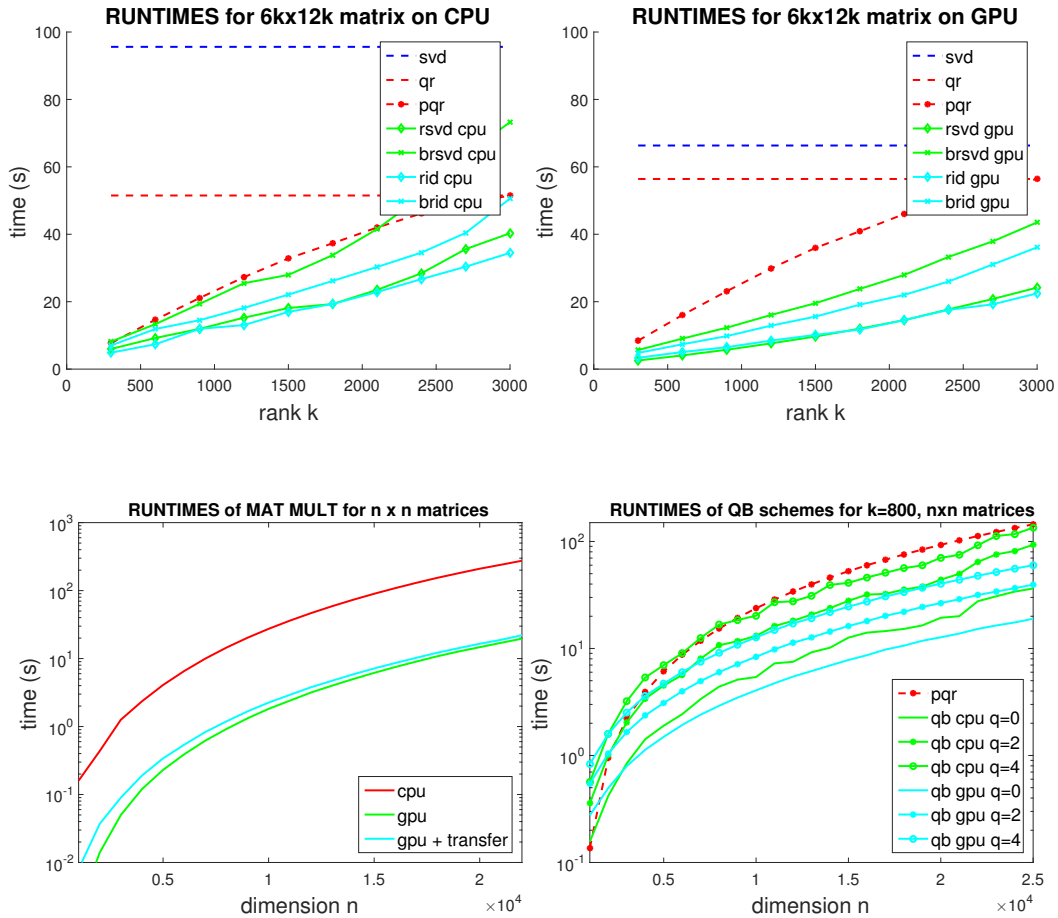
- Take image matrix X (or well compressible portion).
- Apply 2D wavelet transform (CDF 97) to get $w = WX$.
- Apply thresholding to get $\tilde{w} = \mathbb{T}(w)$.
- Apply low rank CUR to compress $\tilde{w}$: $\tilde{w} \approx CUR$.
- To reconstruct: $\tilde{X} = W^{-1}(CUR)$.

Original and compression ratios $2, 4, 8$ after pure wavelet compression.



RSVDPACK

- Open source routines, for multi-core and GPU architectures, available at `https://github.com/sergeyvoronin`.
- Can efficiently construct low rank QB, SVD, ID ($A \approx A(:, J_c(1:k))V^*$), and CUR ($A \approx CUR$) factorizations in $\mathcal{O}(mnk)$ time.

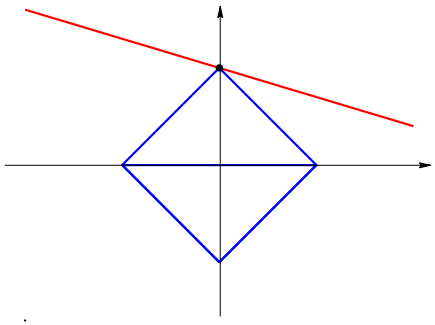
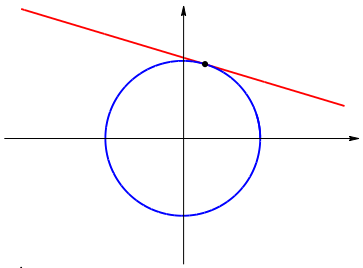
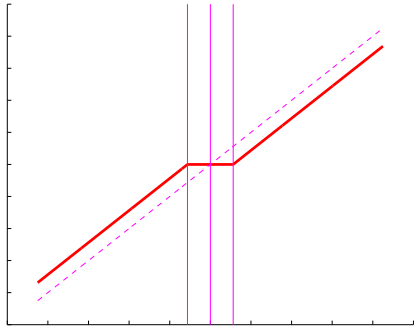


(5) Iterative regularization techniques: new algorithms useful for recovering sparse and multi-scale solutions.

Regularization with sparse penalties

- $f(x,p) = |x|^p$ for $p < 1$ is not convex.
- $f(x,1) = |x|$ is convex.
- Interesting to consider $||x||_1 = \sum_{k=1}^{N} |x_k|$

$$\min\{|x| + |y| : a_1 x + b_1 y = c_1\} \; ; \; \min\{x^2 + y^2 : a_2 x + b_2 y = c_2\}$$

Want to minimize: $||Ax - b||_2^2 + 2\tau||x||_1$ but $||x||_1$ is not smooth.

Soft thresholding

$$(\mathbb{S}_\tau(x))_k = \operatorname{sgn}(x_k)\max(0, |x_k| - \tau)$$

$$\mathbb{S}_\tau(b) = \arg\min_x \left\{||x - b||_2^2 + 2\tau||x||_1\right\}$$

Majorization-Minimization Approach for ISTA

Introduce two parameter G such that:

$$G(x,y) \geq F(x) \quad \forall\, x,y \quad \text{and} \quad G(x,x) = F(x)$$

Then set $x^{n+1} = \arg\min_x G(x, x^n)$.

$$G(x^{n+1}, x^{n+1}) = F(x^{n+1}) \leq G(x^{n+1}, x^n) \leq G(x^n, x^n) = F(x^n)$$

Scale A (and b) so that $\|A\|_2 < 1$ and set:

$$\begin{aligned} G(x, x^n) &= \|Ax - b\|_2^2 + \|x - x^n\|_2^2 - \|A(x - x^n)\|_2^2 + 2\tau\|x\|_1 \\ &= \|x - (x^n + A^T b - A^T A x^n)\|_2^2 + 2\tau\|x\|_1 + K \end{aligned}$$

Since $\mathbb{S}_\tau(c) = \arg\min_x \|x - c\|_2^2 + 2\tau\|x\|_1$ we get the scheme:

$$x^{n+1} = \arg\min_x G(x, x^n) = \mathbb{S}_\tau(x^n + A^T b - A^T A x^n)$$

This is known as the Iterative Soft Thresholding Algorithm (ISTA) but it is slow.

$$F(x^n) - F(\bar{x}) \leq \frac{C_1 \|x^0 - \bar{x}\|_2^2}{n}$$

Fast-ISTA (Nesterov; Beck and Teboulle)

$$y^0 = x^0 \quad , \quad t_1 = 1 \quad , \quad x^{n+1} = \mathbb{S}_\tau(y^n - A^T(Ay^n - b))$$
$$t_{n+1} = \frac{1 + \sqrt{1 + 4t_n^2}}{2}$$
$$y^{n+1} = x^n + \frac{t_n - 1}{t_{n+1}}(x^n - x^{n-1})$$

Much Faster Convergence

$$F(x^n) - F(\bar{x}) \leq \frac{C_2 \|x^0 - \bar{x}\|_2^2}{(n+1)^2}$$

FISTA speedup trick above can be applied to different algorithms.

Other Approaches

- Coordinate Descent: Update one coordinate at a time.
- Dual Space: work with dual of $||x||_1$.
- Smooth Approximation: replace $||x||_1$ by something smooth.

Dual Space

Definition

The dual of a norm $||\cdot||$ on $\mathbb{R}^N$ is defined for any $y \in \mathbb{R}^N$ as,

$$||y||_* = \max_{x \in \mathbb{R}^n, x \neq 0} \frac{\langle y, x \rangle}{||x||}.$$

Lemma

The dual of $||x||_1 = \sum_{k=1}^{N} |x_k|$ *is* $||y||_\infty = \max_i [|y_i|]$

Dual space algorithm procedure:

$$(1) \quad \left[\min_x f(x) \text{ s.t. } Ax = b\right] \rightarrow L(x, y) = f(x) + y^T(b - Ax)$$

$$(2) \quad g(y) = \min_x L(x, y) \rightarrow \bar{y} = \arg\max_y g(y)$$

$$(3) \quad \bar{x} = \arg\min_x L(x, \bar{y})$$

$\min ||x||_1$ s.t. $Ax = b$ (DALM [Yang, Zhang. 2011])

$$L(x, y) = ||x||_1 + y^T(b - Ax)$$

Dual problem:

$$\left[\max_y b^T y \text{ s.t. } ||A^T y||_\infty \leq 1\right] \rightarrow \left[\min_y -b^T y \text{ s.t } z = A^T y, ||z||_\infty \leq 1\right]$$

Augmented Lagrangian:

$$\min_{x,y,z} L_\mu(y, z, x) := -b^T y - x^T(z - A^T y) + \frac{\mu}{2}\|z - A^T y\|_2^2 \quad \text{s.t.} \quad \|z\|_\infty \leq 1.$$

We **can differentiate** L with respect to each variable:

$$\begin{aligned}
\nabla_x L_\mu(x, y, z) &= A^T y - z \\
\nabla_y L_\mu(x, y, z) &= -b + Ax + \mu A(A^T y - z) \\
\nabla_z L_\mu(x, y, z) &= -x + \mu(z - A^T y).
\end{aligned}$$

We alternate with the different updates and increase μ.

Sparse Reconstruction

- Start with sparse x. Set $y = Ax + \nu$ where A is a sample matrix, ν is noise.
- Solve $\bar{x} = \arg\min_w ||Ax - y||_2^2 + \Phi(x)$ and compare $\bar{x}$ to original x.
- Use continuation strategy for τ, starting close to $||A^T y||_\infty$ and decreasing until $||A\bar{x} - y||_2 \approx ||\nu||_2$.

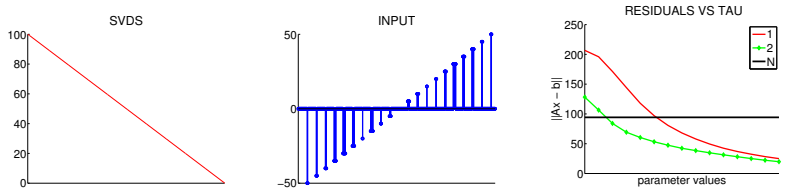
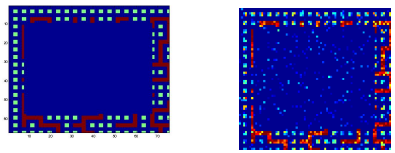


Image Reconstruction

- Start with vectorized image x.
- Obtain samples with sensing matrix A: $y = Ax + \nu$ where ν is some Gaussian noise.
- Recover x via $\bar{x} = \arg\min_x ||Ax - y||_2^2 + \Phi(x)$.



Wavelet Image Denoising

- Start with vectorized image x sparse under some transform W.
- Obtain $y = x + \nu$ where ν is some Gaussian noise.
- Denoise by solving $\bar{w} = \arg\min_w ||W^{-1}w - y||_2^2 + \Phi(w)$. Set $\bar{x} = W^{-1}\bar{w}$.

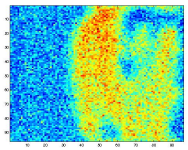
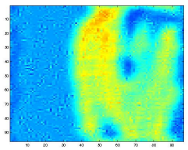
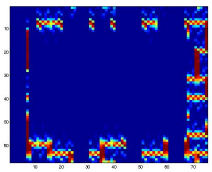
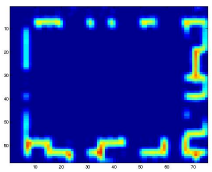
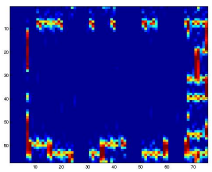
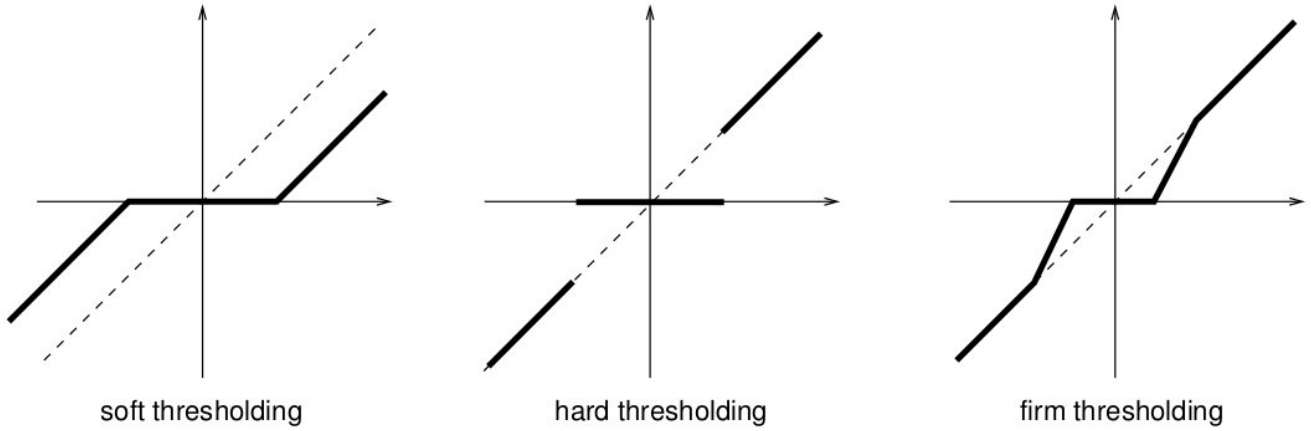



Image Deblurring

- Start with vectorized image x sparse under some transform W.
- Obtain $y = AHx + \nu$ where A is a sample matrix, H is a blur matrix and ν is noise.
- Solve $\bar{w} = \arg\min_w ||AHW^{-1}w - y||_2^2 + \Phi(w)$ and compare $\bar{x} = W^{-1}\bar{w}$ to the original x.

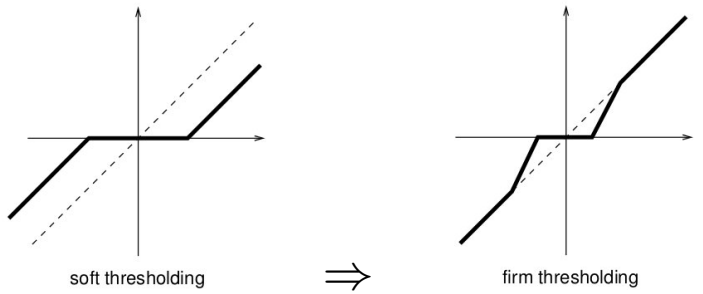
Variable Thresholding Function [with H. Woerdeman, 2012]



$$\mathbb{V}_{\rho,\tau}(a) = \begin{cases} a - (2\rho - \tau), & a \geq \tau; \\ 2(a - \rho), & \rho < a < \tau; \\ 0, & -\rho \leq a \leq \rho; \\ 2(a + \rho), & -\tau < a < -\rho\ ; \\ a + (2\rho - \tau), & a \leq -\tau\ . \end{cases}$$

Notice that when $\rho = \tau$ above, $\mathbb{V}_{\tau,\tau}(a) = \mathbb{S}_{\tau}(a)$. Also, when $\rho = \frac{\tau}{2}$, the large entries are not penalized.

We proceed from soft to firm thresholding (convex to non-convex optimization):



Iterative Variable Thresholding Algorithm

$$x^{n+1} = \mathbb{V}_{\rho_n,\tau}(x^n + A^T b - A^T A x^n)$$

with $\rho_0 = \tau$ and $\rho_n \to \frac{\tau}{2}$ as $n \to \infty$.

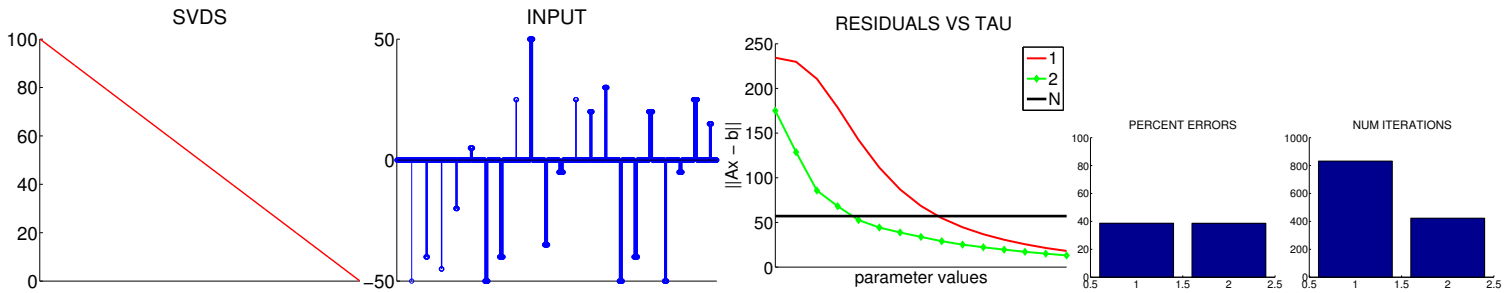
FIVTA

$$y^0 = x^0,\ x^n = \mathbb{V}_{\rho^n,\tau}(y^n + A^T(b - Ay^n)),$$

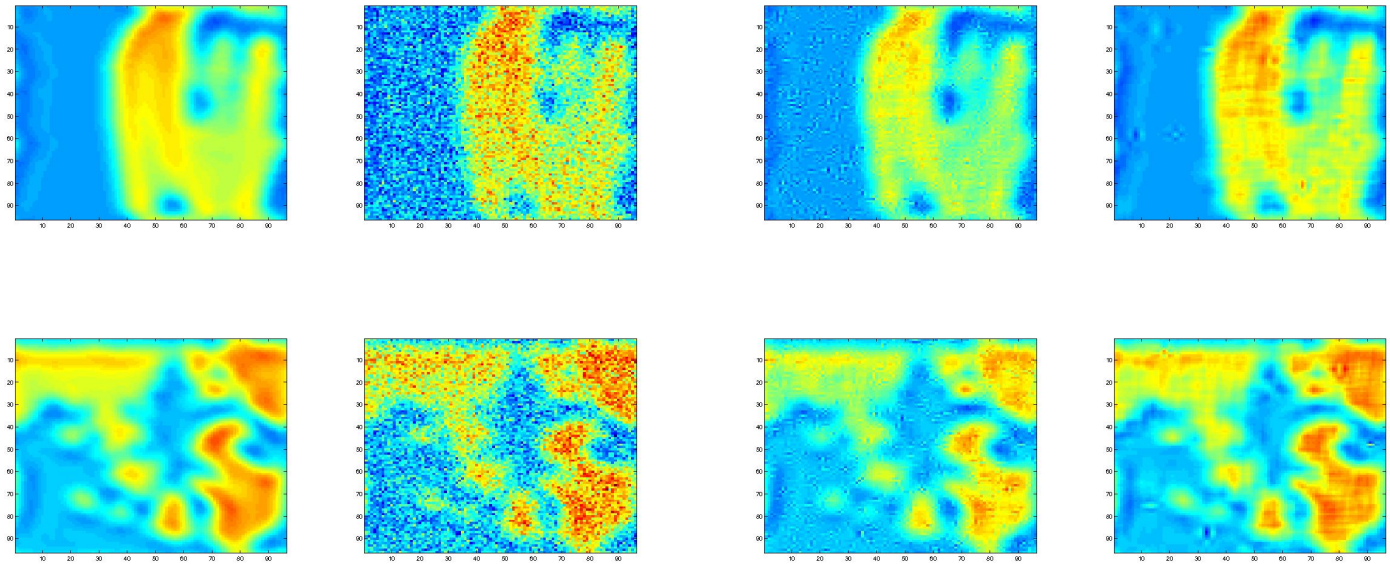
$$y^{n+1} = x^n + \frac{t_n - 1}{t_{n+1}}(x^n - x^{n-1})$$

Numerical Advantages of IVTA/FIVTA schemes

Faster Convergence:

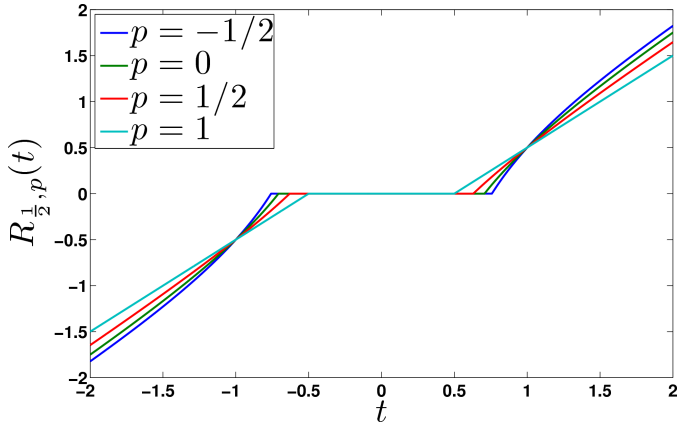


Better wavelet image denoising:

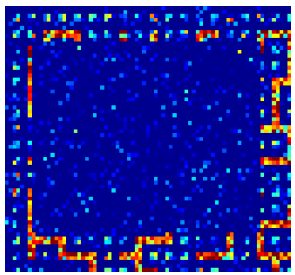
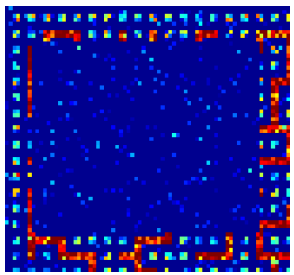
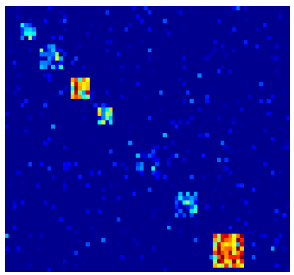
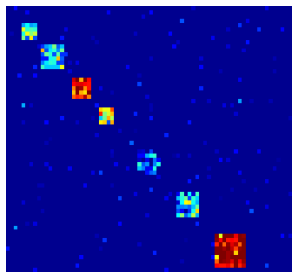


p-thresholding with R. Chartrand (2013)

$$
\begin{aligned}
(\mathbb{S}_\tau(x))_k &= \operatorname{sgn}(x_k)\max(0, |x_k| - \tau) \\
(\mathbb{R}_{\tau,p}(x))_k &= \operatorname{sgn}(x_k)\max(0, |x_k| - \tau|x_k|^{p-1})
\end{aligned}
$$



$$x^{n+1} = \mathbb{R}_{\tau,p}(x^n + A^T b - A^T A x^n)$$

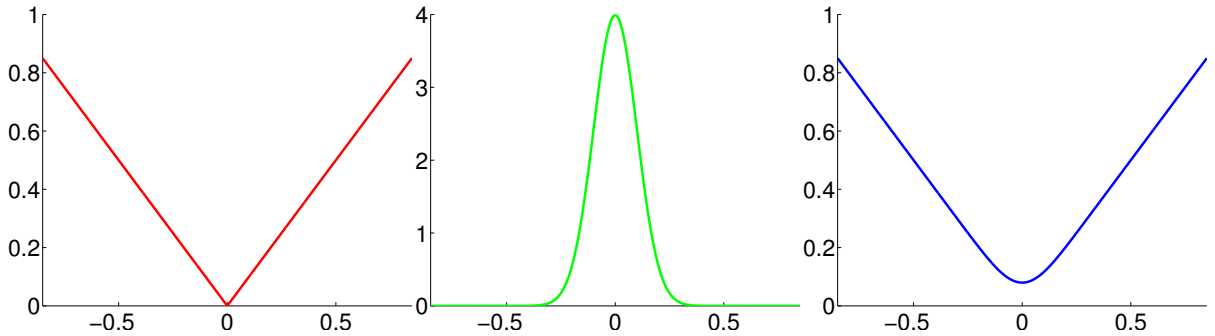





Convolution smoothing (with G. Ozkaya and D. Yoshida)

Approximate mollifier; smooth out absolute value $|t|$ with bump function:

$$f(t) = \frac{1}{2\pi\sigma^2} e^{\frac{-t^2}{2\sigma^2}} \ , \ g(t) = |t| \ , \ \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$$

$$|t| \approx (f \star g)(t) = \int_{-\infty}^{\infty} f(s) g(s-t)\, \mathrm{d}s = t\,\mathrm{erf}\left(\frac{t}{\sqrt{2}\sigma}\right) + \sqrt{\frac{2}{\pi}}\sigma e^{\frac{-t^2}{2\sigma^2}}$$



$$\begin{aligned} F_1(x) &= \|Ax-b\|_2^2 + 2\tau\|x\|_1 \\ &\approx H_1(x) = \|Ax-b\|_2^2 + 2\tau \sum_{k=1}^{N} \left(x_k\, \mathrm{erf}\left(\frac{x_k}{\sqrt{2}\sigma}\right) + \sqrt{\frac{2}{\pi}}\sigma e^{\frac{-x_k^2}{2\sigma^2}} \right) \end{aligned}$$

$$\nabla_x \left[H_1(x)\right] = 2A^T(Ax-b) + 2\tau \left\{ \mathrm{erf}(\frac{x_k}{\sqrt{2}\sigma}) \right\}_{k=1,\ldots,N}$$

$$\nabla_x^2 \left[H_1(x)\right] = 2A^T A + \frac{4\sqrt{2}\tau}{\sigma\sqrt{\pi}} Diag\left(\exp\left(-\frac{x_k^2}{2\sigma^2} \right) - \frac{x_k^2}{2\sigma^2} \exp\left(-\frac{x_k^2}{2\sigma^2} \right) \right)$$

Can generalize to non-convex min ($p < 1$).

$$\begin{aligned} F_p(x) &= \|Ax-b\|_2^2 + 2\tau \left(\sum_{k=1}^n |x_k|^p \right)^{\frac{1}{p}} \\ &\approx H_p(x) = \|Ax-b\|_2^2 + 2\tau \left(\sum_{k=1}^n \phi_\sigma^p(x_k) \right)^{\frac{1}{p}} \end{aligned}$$

$$\nabla H_{p,\sigma}(x) = 2A^T(Ax-b) + 2\tau p \left(\sum_{k=1}^n \phi_\sigma(x_k)^p \right)^{(1-p)/p} \left\{ \phi_\sigma(x_j)^{p-1} \,\mathrm{erf}\left(\frac{x_j}{\sqrt{2}\sigma} \right) \right\}_{j=1}^n,$$

and the Hessian is given by:

$$\nabla^2 H_{p,\sigma}(x) = 2A^T A + 2\tau \left(v(x)v(x)^T + \mathrm{Diag}\big(w(x)\big) \right),$$

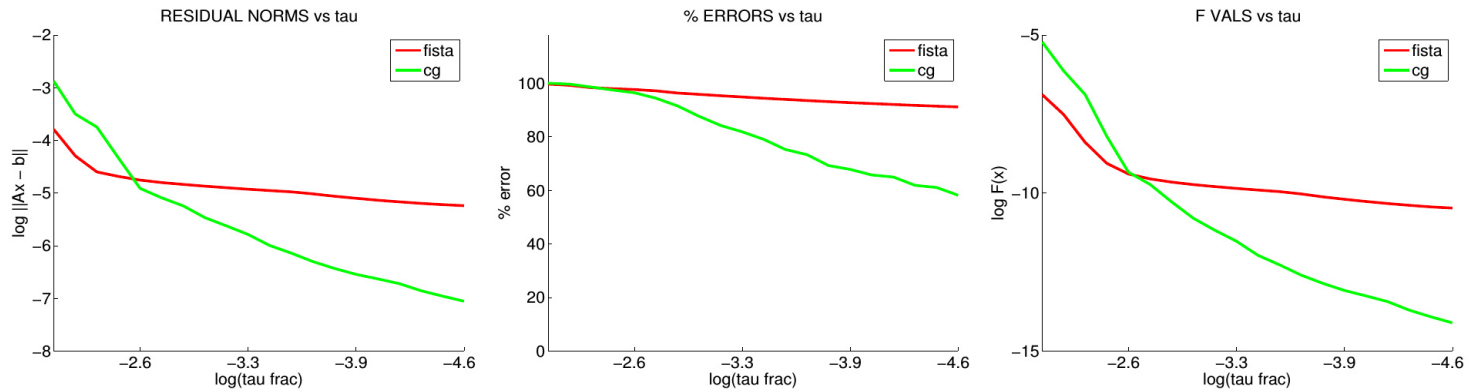
where $v(x)$ and $w(x)$ depend on $\mathrm{erf}(x)$.

Algorithm Nonlinear Conjugate Gradient Scheme

Input : An $m \times n$ matrix A, an initial guess $n \times 1$ vector x^0, a parameter $\tau < \|A^T b\|_\infty$, a parameter $p \in (0, 1]$, a parameter $\sigma_0 > 0$, a parameter $0 < \alpha < 1$, the maximum number of iterations M, and a routine to evaluate the gradient $\nabla H_{p,\sigma}(x)$ (and possibly the Hessian $\nabla^2 H_{p,\sigma}(x)$ depending on choice of line search method).

Output: A vector $\bar{x}$, close to either the global or local minimum of $F_p(x)$, depending on choice of p.

$s^0 = -\nabla H_{p,\sigma_0}(x^0)$;
for $k = 0, 1, \ldots, M$ **do**
  use line search to find $\mu > 0$;
  $x^{n+1} = \texttt{Threshold}(x^n + \mu s^n, \tau)$;
  $\beta^{n+1} = \max\left\{ \frac{\nabla H_{p,\sigma_n}(x^{n+1})^T (\nabla H_{p,\sigma_n}(x^{n+1}) - \nabla H_{p,\sigma_n}(x^n))}{\nabla H_{p,\sigma_n}(x^n)^T \nabla H_{p,\sigma_n}(x^n)}, 0 \right\}$;
  $s^{n+1} = -\nabla H_{p,\sigma_n}(x^{n+1}) + \beta^{n+1} s^n$;
  $\sigma_{n+1} = \alpha \sigma_n$;
end
$\bar{x} = x^{n+1}$;



Re-weighted least squares (with I. Daubechies, 2012, 2016)

Use a weighted two norm: $||x||_{2,w} = \sum_{k=1}^{N} w_k x_k^2$. Weight based on x^n:

$$||x||_1 = \sum_{k=1}^{N} |x_k| = \sum_{k=1}^{N} \frac{x_k^2}{|x_k|} \approx \sum_{k=1}^{N} \frac{x_k^2}{\sqrt{(x_k^n)^2 + (\epsilon_n)^2}} = \sum_{k=1}^{N} w_k^n x_k^2$$

Iteratively Reweighted Least Squares MM Algorithm

$$\frac{\partial}{\partial x_k} \left(||Ax - b||_2^2 - ||A(x - x^n)||_2^2 + ||x - x^n||_2^2 + 2\tau \sum_{l=1}^{N} w_l^n x_l^2 \right) = 0$$

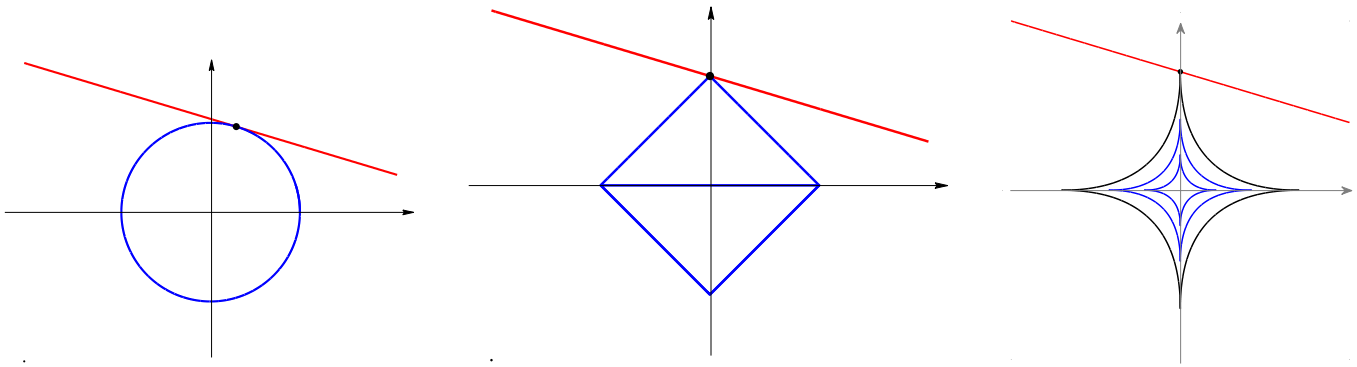
$$\Rightarrow -2(A^T b)_k + 2(A^T A x^n)_k + 2x_k - 2x_k^n + 4\tau w_k^n x_k = 0.$$

$$\begin{aligned} x_k^{n+1} &= (\arg\min_x G(x, x^n, w^n, \epsilon_n))_k \\ \Longrightarrow \; x_k^{n+1} &= \frac{1}{1 + 2\tau w_k^n} \left(x_k^n + (A^T b)_k - (A^T A x^n)_k \right) \end{aligned}$$

Generalization of the Objective Functional

Instead of $||Ax - b||_2^2 + 2\tau||x||_1$, we generalize to:

$$||Ax - b||_2^2 + 2\sum_{k=1}^{N} \lambda_k |x_k|^{q_k} \quad \text{for} \quad 1 \leq q_k \leq 2$$



$$\min(|x|^q + |y|^q)^{\frac{1}{q}} \text{ for } q = 2, q = 1, q = 0.5.$$

$$||x||_1 = \sum_{k=1}^{N} |x_k| = \sum_{k=1}^{N} \frac{x_k^2}{|x_k|} \approx \sum_{k=1}^{N} \frac{x_k^2}{\sqrt{(x_k^n)^2 + (\epsilon_n)^2}}$$

$$|x_k|^{q_k} \approx \frac{x_k^2}{((x_k^n)^2 + (\epsilon_n)^2)^{\frac{2-q_k}{2}}}$$

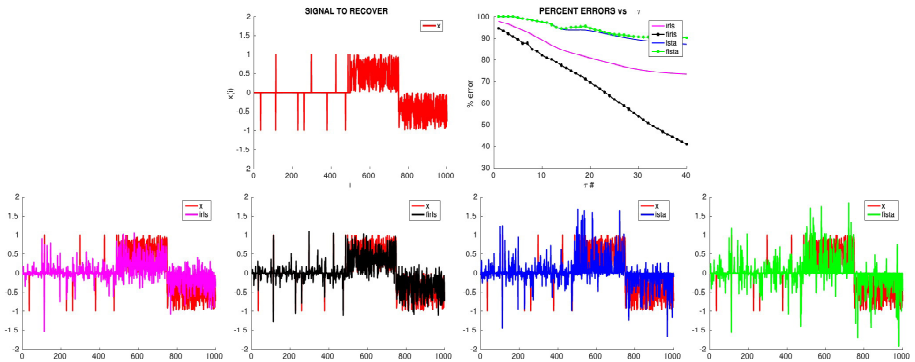
Simple derivation of IRLS scheme

Using the Majorization-Minimization setup:

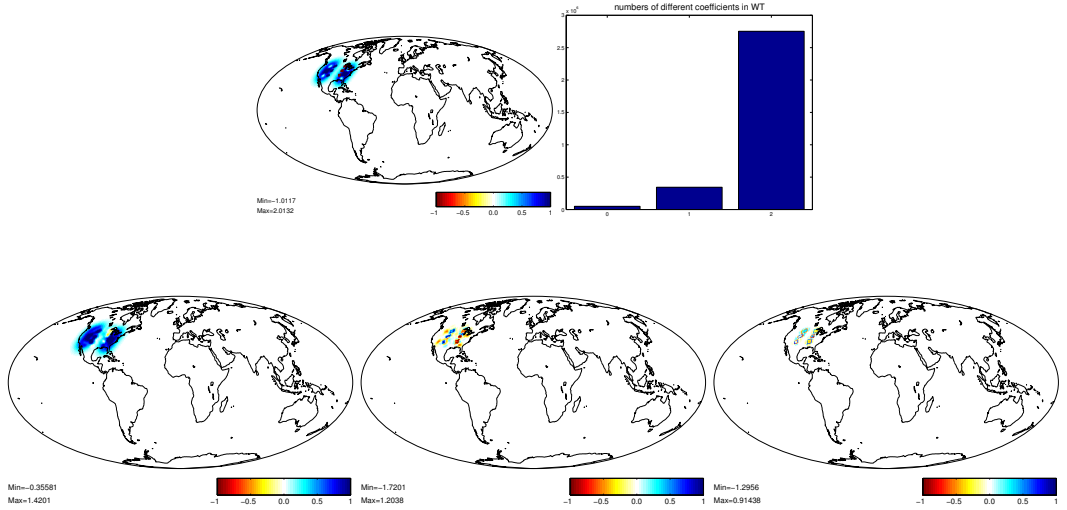
$$
\begin{aligned}
x_k^{n+1} &= \left(\arg\min_x G(x, x^n, w^n, \epsilon_n) \right)_k \\
&= \frac{1}{1 + q_k \lambda_k w_k^n} \left(x_k^n + (A^T b)_k - (A^T A x^n)_k \right) \\
w_k^n &= \frac{1}{((x_k^n)^2 + (\epsilon_n)^2)^{\frac{2-q_k}{2}}} \quad ; \quad \epsilon_n = \min\left(\epsilon_{n-1}, \left(\|x^n - x^{n-1}\|_2 + \alpha_n \right)^{\frac{1}{2}} \right)
\end{aligned}
$$

for the generalized functional

$$
\arg\min_x \left\{ \|Ax - b\|_2^2 + 2 \sum_{k=1}^{N} \lambda_k |x_k|^{q_k} \right\} \quad \text{for} \quad 1 \le q_k \le 2
$$



Feature extraction for Geophysical models



Expand model in a wavelet basis

Let $w = Wx$ and $x = (m_j) = W^{-1}w$. Then $Ax = b \implies AW^{-1}w = b$.

$$b_i = \sum_{j=1}^{N} K_{i,j} m_j = \sum_{j=1}^{N} K_{i,j} \left(\sum_{k=1}^{N} W_{j,k}^{-1} w_j \right) = \sum_{j=1}^{N} \sum_{k=1}^{N} K_{i,j} W_{j,k}^{-1} w_j$$

Hence, for the discretized minimization problem, we can solve:

$$\bar{w} = \arg\min_{w} \left\{ \|AW^{-1}w - b\|_2^2 + \sum_{k=1}^{N} \theta(w_k, \lambda_k) \right\} \quad ; \quad \bar{x} = W^{-1}\bar{w}$$

YAMPA Support Detection (with A. Lodhi and W. Bajwa, 2016)

CoSaMP type of method with **matrix dependent threshold**. Threshold derived based on assuming high probability of correct support in first iteration.

Algorithm 1: Yet Another Matching Pursuit Algorithm (YAMPA)

Require: Measurements $\mathbf{b}$ and measurement matrix $\mathbf{A}$.

Ensure: Initial estimate $\hat{\mathbf{x}}_0 \leftarrow \mathbf{0}$, initial global support $\mathbf{T}_{gl} \leftarrow \emptyset$, and iteration number $s \leftarrow 0$.

1: **while** stopping criteria is satisfied **do**
2: $s \leftarrow s + 1$
3: Obtain s-th residual signal $\mathbf{r}_s \leftarrow \mathbf{b} - \mathbf{A}\hat{\mathbf{u}}_{s-1}$
4: Form (absolute-valued) proxy vector $\mathbf{q} \leftarrow |\mathbf{A}^H \mathbf{r}_s|$
5: Find local support set $\mathbf{T}_s \leftarrow \{i |\, q_i > \lambda_s\}$
6: Merge supports: $\mathbf{T}_{gl} \leftarrow \mathbf{T}_{gl} \cup \mathbf{T}_s$
7: Find new estimate $\hat{\mathbf{x}}_{s_{\mathbf{T}_{gl}}} \leftarrow \mathbf{A}^{\dagger}_{\mathbf{T}_{gl}} \mathbf{b}$
8: **end while**

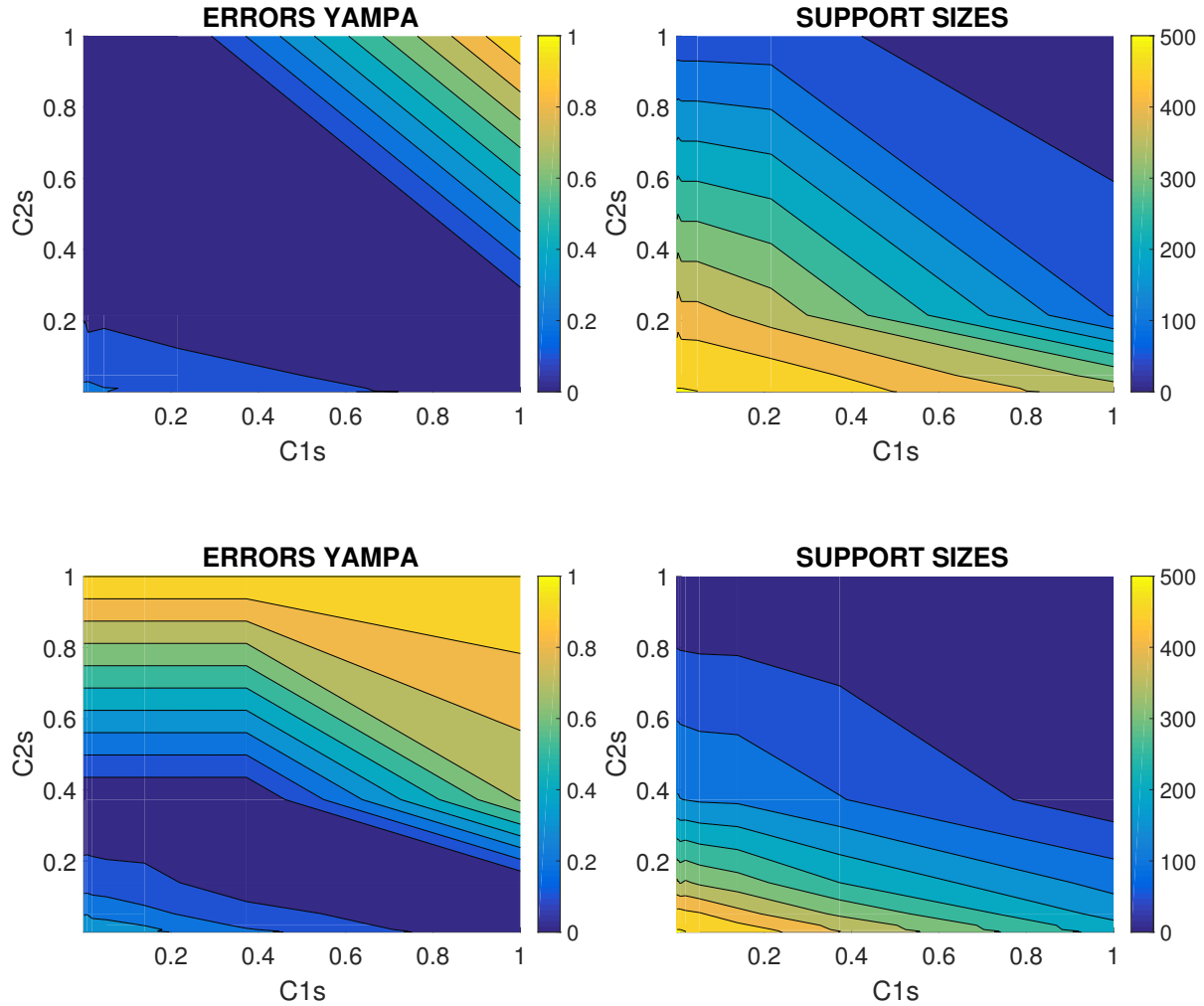
Output: Final estimate $\hat{\mathbf{x}}_s$.

$$\text{Worst-Case Coherence:} \quad \mu(\mathbf{A}) = \max_{i,j;i\neq j} \left|\langle \mathbf{a}_i, \mathbf{a}_j \rangle\right|, \text{ and}$$

$$\text{Average Coherence:} \quad \nu(\mathbf{A}) = \frac{1}{n-1} \max_{i} \left| \sum_{j:j\neq i} \langle \mathbf{a}_i, \mathbf{a}_j \rangle \right|, \tag{16}$$

$$\text{Threshold:} \quad \lambda_s = c_1 \mu \|\mathbf{r}_s\|_2 + c_2 \nu \sqrt{\hat{k}} \|\mathbf{r}_s\|_2, \tag{17}$$

Reconstruction errors for 1000×1000 Gaussian random matrices with rapid singular value decay and 0 and 300 approximately correlated columns.



Thanks!

Regularization, wavelets, big matrices. Low rank matrix approximations can be computed efficiently using existing software and applied to various inverse problems. Various open problems.

- S. Voronin and P.G. Martinsson. RSVDPACK: An implementation of randomized algorithms for computing the singular value, interpolative, and CUR decompositions of matrices on multicore and GPU architectures, 2016.
- S. Voronin and P.G. Martinsson. Efficient algorithms for CUR and Interpolative Matrix Decomposition, 2016.
- S. Voronin and I. Daubechies. An iteratively reweighted least squares algorithm for regularization with sparsity constraints, 2016.
- P.G. Martinsson and S. Voronin. A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices, 2015.
- S. Voronin, D. Mikesell, and G. Nolet. Compression Approaches for the Regularized Solutions of Linear System from Large-Scale Inverse Problems, 2015.
- S. Voronin, D. Mikesell, I. Slezak, and G. Nolet. Solving large tomographic linear systems: size reduction and error estimation, 2014.