Problems of Varying Size

Small-scale problems:

- "anything goes,"
- no problem to use SVD (recommended).

Medium-size problems:

- cannot ignore computing time,
- other factorizations, sparse matrix aspects, etc.

Large-scale problems:

- storage and computing time set the limitations,
- factorizations are not possible in general,
- if possible, use matrix structure (Toeplitz, Kronecker, . . .),
- otherwise must use iterative methods!

Advantages of Iterative Methods

Iterative methods produce a sequence $x^{[0]} \to x^{[1]} \to x^{[2]} \to \cdots$ of iterates that (hopefully) converge to the desired solution, solely through the use of matrix-vector multiplications.

- The matrix A is never altered, only "touched" via matrix-vector multiplications $A\,x$ and $A^T y$.
- The matrix A is not explicitly required – we only need a "black box" that computes the action of A or the underlying operator.
- Atomic operations of iterative methods (mat-vec product, saxpy, norm) suited for high-performance computing.
- Often produce a natural sequence of regularized solutions; stop when the solution is "satisfactory" (parameter choice).

Two Types of Iterative Methods

1. Iterative solution of a regularized problem, such as Tikhonov

$$\left(A^T A + \lambda^2 L^T L\right) x = A^T b \ .$$

   Challenge: to construct a good preconditioner!

2. Iterate on the un-regularized system, e.g., on

$$A\,x = b \quad \text{or} \quad A^T A\,x = A^T b$$

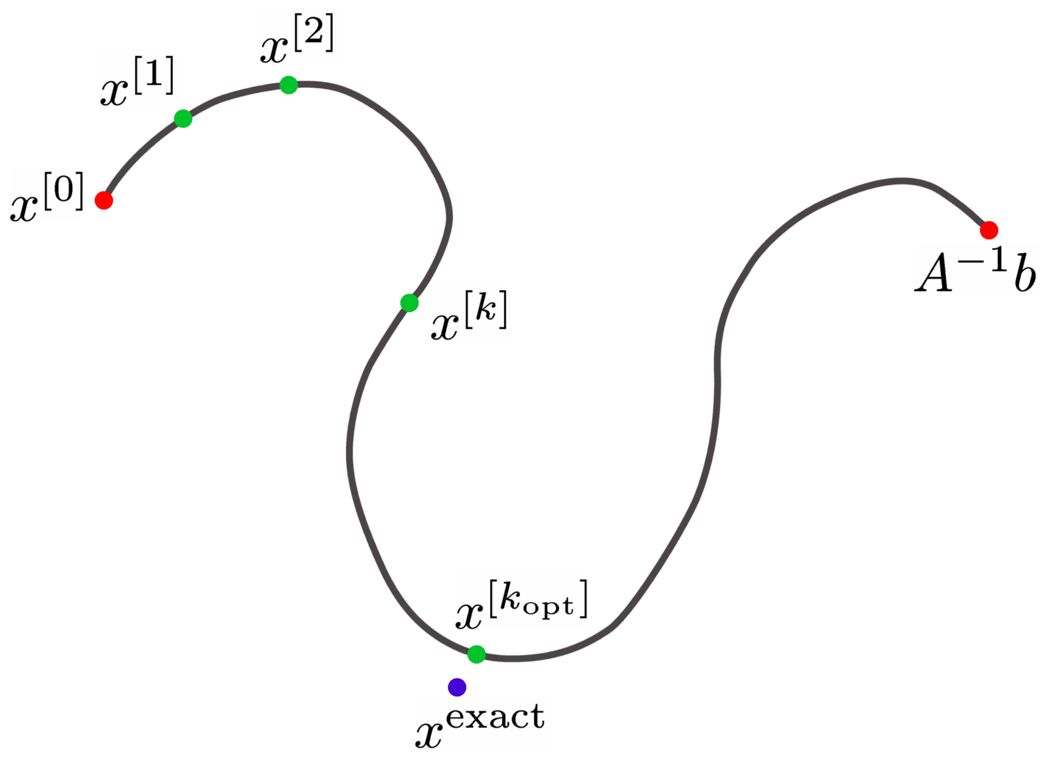
   and use the iteration number as the regularization parameter.

The latter approach relies on *semi-convergence:*

- initial convergence towards x^{exact},
- followed by (slow) convergence to x_{naive}.

Must stop at the end of the first stage!

Illustration of Semi-Convergence



Landweber Iteration

A classical stationary iterative method:

$$x^{[k+1]} = x^{[k]} + \omega\, A^T (b - A\, x^{[k]}) \ , \qquad k = 0, 1, 2, \ldots$$

where $0 < \omega < 2\, \|A^T A\|_2^{-1} = 2\, \sigma_1^{-2}$.

Where does this come from? Consider the function

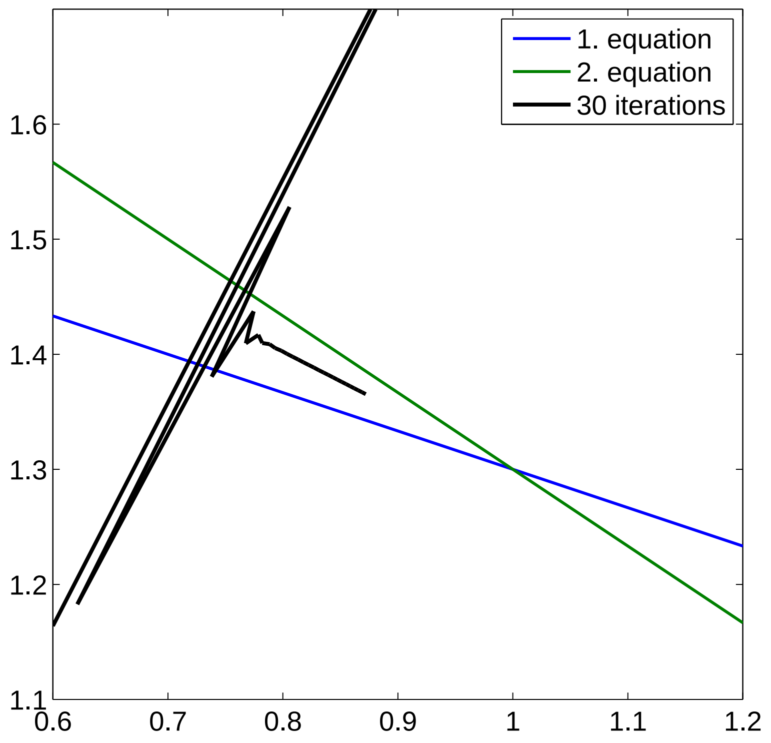
$$\phi(x) = \tfrac{1}{2} \|b - A\, x\|_2^2$$

associated with the least squares problem $\min_x \phi(x)$. It is straightforward (but perhaps a bit tedious) to show that the gradient of ϕ is

$$\nabla \phi(x) = -A^T (b - A\, x).$$

Thus, each step in Landweber's method is a step in the direction of steepest descent. See next slide for an example of iterations.

The Geometry of Landweber Iterations



SVD Analysis of Landweber's Method

SVD analysis shows that the filter factors (see next page) are:

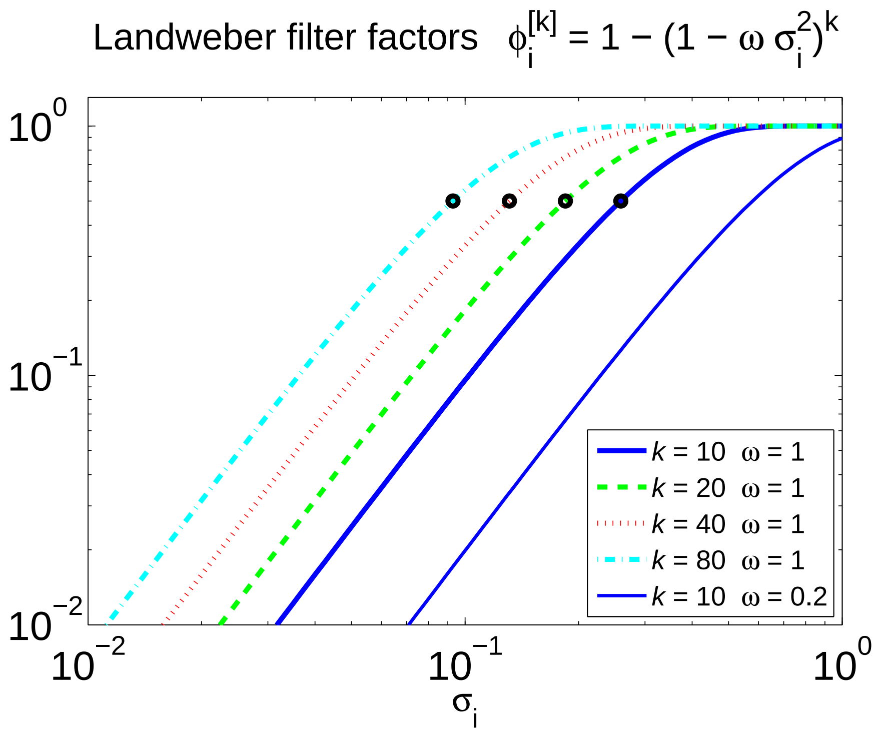
$$\phi_i^{[k]} = 1 - (1 - \omega\,\sigma_i^2)^k .$$

Let $\sigma_{\text{break}}^{[k]}$ denote the value of σ_i for which $f_i^{[k]} = 0.5$. Then

$$\frac{\sigma_{\text{break}}^{[k]}}{\sigma_{\text{break}}^{[2k]}} = \sqrt{1 + (\tfrac{1}{2})^{\frac{1}{2k}}} \to \sqrt{2} \quad \text{for} \quad k \to \infty .$$

Hence, as k increases, the breakpoint tends to be reduced by a factor $\sqrt{2} \approx 1.4$ each time the number of iterations k is doubled.

Landweber Filter Factors



Cimmino Iteration

Cimmino's method is a variant of Landweber's method, with a diagonal scaling:

$$x^{[k+1]} = x^{[k]} + \omega\, A^T D\, (b - A\, x^{[k]}), \qquad k = 1, 2, \ldots$$

in which $D = \mathrm{diag}(d_i)$ is a diagonal matrix whose elements are defined in terms of the rows $a_i^T = A(i,:)$ of A as

$$d_i = \begin{cases} \dfrac{1}{m}\, \dfrac{1}{\|a_i\|_2^2}, & a_i \neq 0 \\ 0, & a_i = 0. \end{cases}$$

Landweber and Cimmino belong to a class of iterative methods called Simultaneous Iterative Reconstruction Techniques (SIRT).

. . . and the prize for best acronym goes to "ART"

Kaczmarz's method = algebraic reconstruction technique (ART).

Let $a_i^T = A(i,:) = i$th row of A, and $b_i = i$th component b.

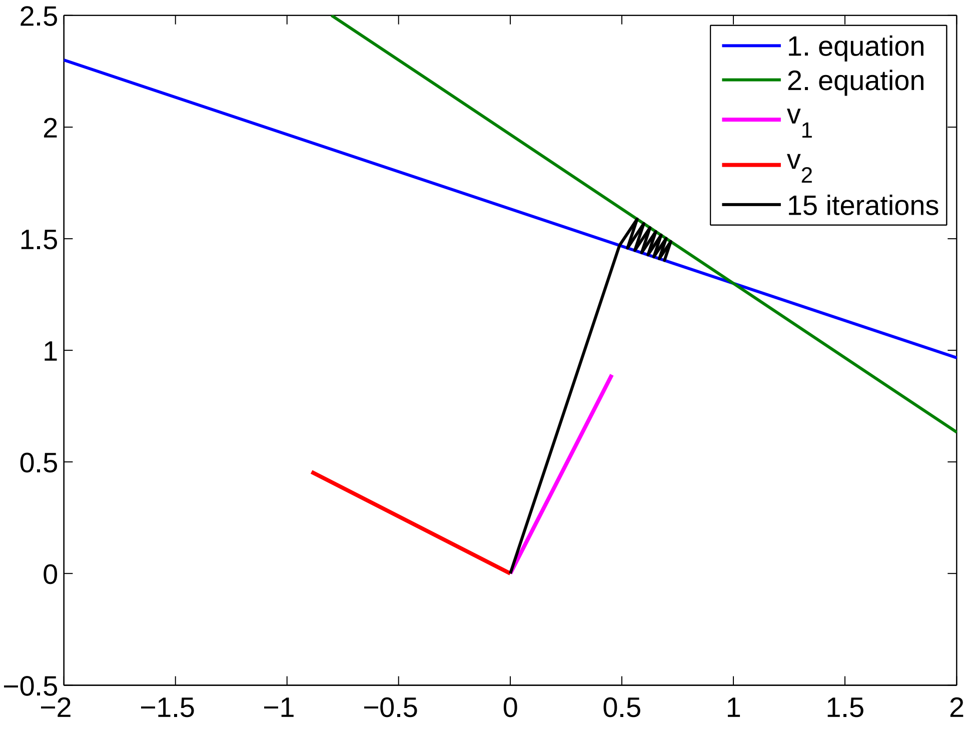
Each iteration of ART involves the following "sweep" over all rows:

```
$x^{[k^{(0)}]} = x^{[k]}$
for $i = 1, \ldots, m$
    $x^{[k^{(i)}]} = x^{[k^{(i-1)}]} + \dfrac{b_i - a_i^T x^{[k^{(i-1)}]}}{\|a_i\|_2^2}\, a_i$
end
$x^{[k+1]} = x^{[k^{(m)}]}$
```

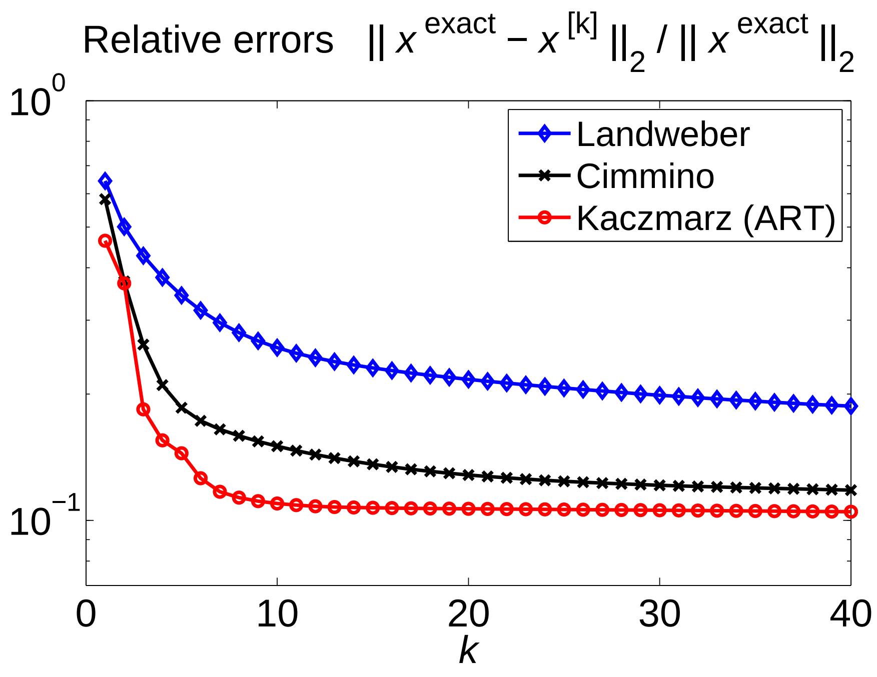
This method is not "simultaneous" because each row must be processed sequentially.

In general: fast initial convergence, then slow. See next slides.

The Geometry of ART Iterations



Slow Convergence of SIRT and ART Methods



The test problem is `shaw`.

Projection Methods

As an important step towards the faster *Krylov subspace methods*, we consider projection methods.

Assume the columns of $W_k = (w_1, \ldots, w_k) \in \mathbb{R}^{n\times k}$ form a "good basis" for an approximate regularized solution, obtained by solving

$$\min_x \|A\,x - b\|_2 \qquad \text{s.t.} \qquad x \in \mathcal{W}_k = \text{span}\{w_1, \ldots, w_k\}.$$

This solution takes the form

$$x^{(k)} = W_k\, y^{(k)}, \qquad y^{(k)} = \text{argmin}_y \|(A\,W_k)\,y - b\|_2,$$

and we refer to the least squares problem $\|(A\,W_k)\,y - b\|_2$ as the *projected problem*, because it is obtained by projecting the original problem onto the k-dimensional subspace span$(w_1, \ldots, w_k)$.

If $W_k = V_k$ then we obtain the TSVD method, and $x^{(k)} = x_k$

But we want to work with computationally simpler basis vectors.

Computations with DCT Basis

Note that

$$\widehat{A}_k = A\,W_k = (W_k^T A^T)^T = \left[(W^T A^T)^T\right]_{:,1:k}.$$

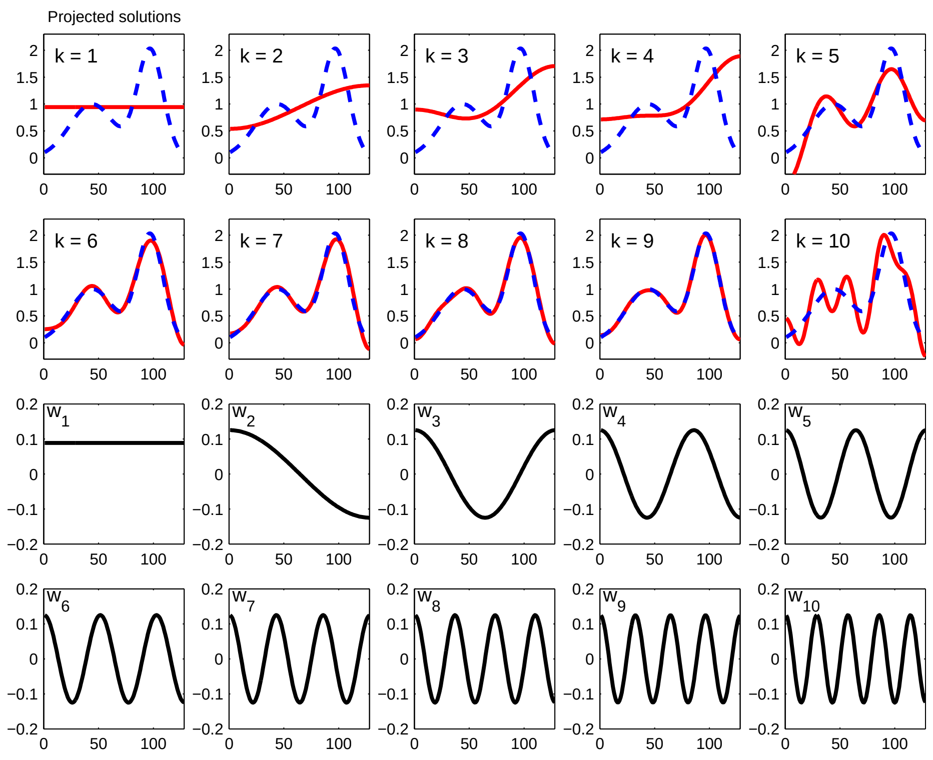
In the case of the discrete cosine basis, multiplication with W^T is equivalent to a DCT. The algorithm takes the form:

```
Akhat = dct(A')';
Akhat = Akhat(:,1:k);
y = Akhat\b;
xk = idct([y;zeros(n-k,1)]);
```

Next page:

- Top: solutions $x^{(k)}$ for $k = 1, \ldots, 10$.
- Bottom: cosine basis w_i, $i = 1, \ldots, 10$.

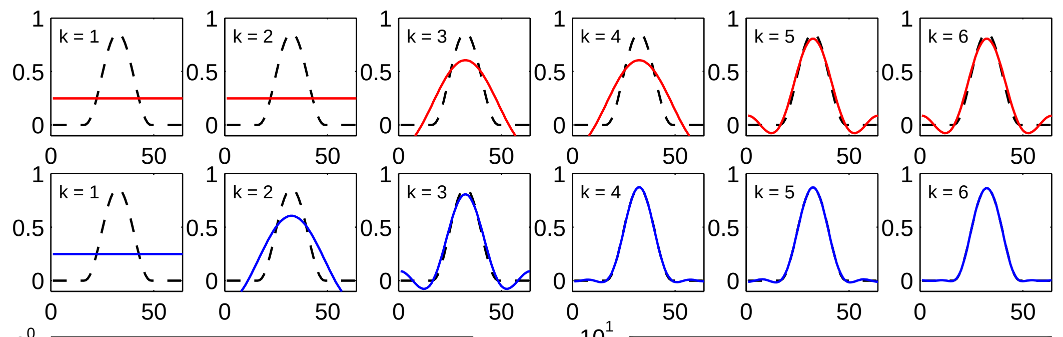
Example Using Discrete Cosine Basis (`shaw`)

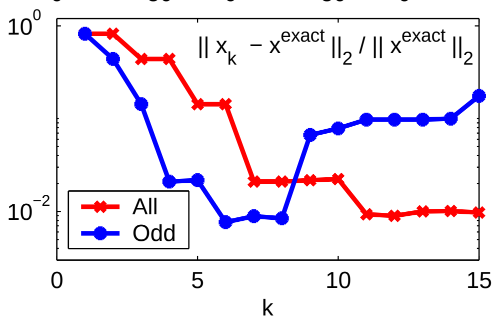


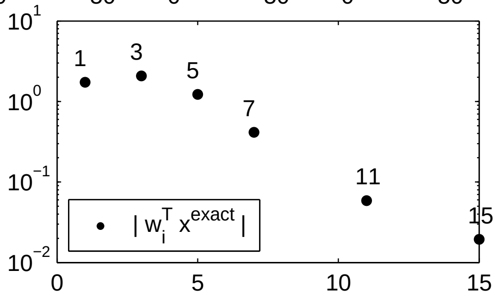
Symmetric Solution and DCT (`phillips`)

Red: Using all the DCT basis vectors $w_1, w_2, w_3, w_4, w_5, w_6, \ldots$

Blue: Using only the odd-numbered DCT vectors $w_1, w_3, w_5, \ldots$







The Krylov Subspace

The DCT basis is sometimes a good basis – but not always.

The **Krylov subspace**, defined as

$$\mathcal{K}_k \equiv \text{span}\{A^T b, A^T A\, A^T b, (A^T A)^2 A^T b, \ldots, (A^T A)^{k-1} A^T b\},$$

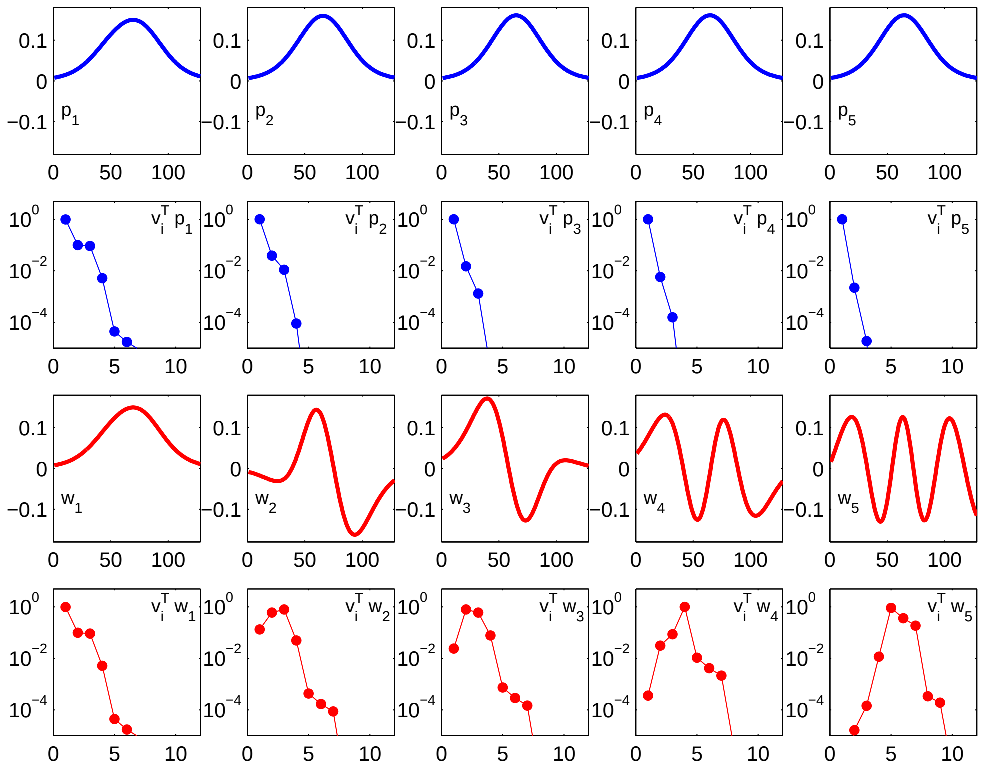
always *adapts* itself to the problem at hand! But the "naive" basis,

$$p_i = (A^T A)^{i-1} A^T b \,/\, \|(A^T A)^{i-1} A^T b\|_2, \qquad i = 1, 2, \ldots$$

are NOT useful: $p_i \to v_1$ as $i \to \infty$. Use modified Gram-Schmidt:

$$\begin{array}{lll}
w_1 \leftarrow A^T b; & w_1 \leftarrow w_1/\|w_1\|_2 & \\
w_2 \leftarrow A^T A\, w_1; & w_2 \leftarrow w_2 - w_1^T w_2\, w_1; & w_2 \leftarrow w_2/\|w_2\|_2 \\
w_3 \leftarrow A^T A\, w_2; & w_3 \leftarrow w_3 - w_1^T w_3\, w_1; & \\
 & w_3 \leftarrow w_3 - w_2^T w_3\, w_2; & w_3 \leftarrow w_3/\|w_3\|_2
\end{array}$$

Comparison of basis vectors p_i (blue) and w_i (red).



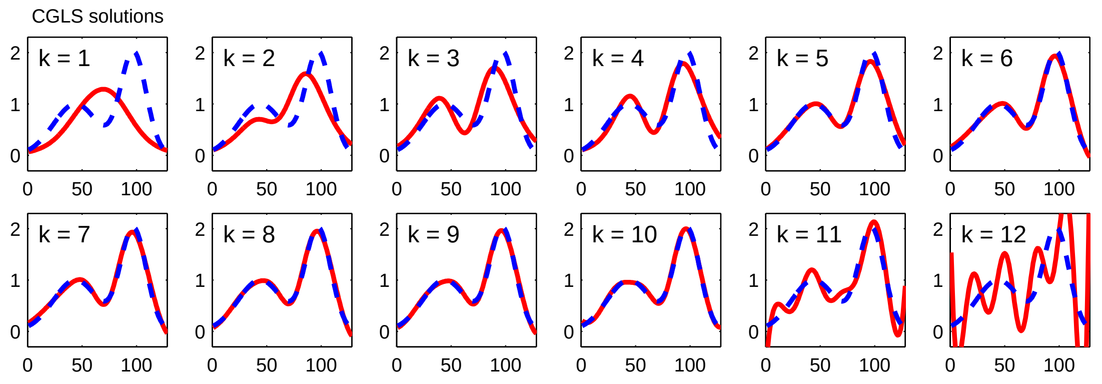
Can We Compute $x^{(k)}$ Without Storing W_k?

Yes – the CGLS algorithm computes iterates given by

$$x^{(k)} = \operatorname{argmin}_x \|A\,x - b\|_2 \qquad \text{s.t.} \qquad x \in \mathcal{K}_k .$$

The algorithm eventually converges to the least squares solution.

But since $\mathcal{K}_k$ is a good subspace for approximate regularized solutions, CGLS exhibits semi-convergence.

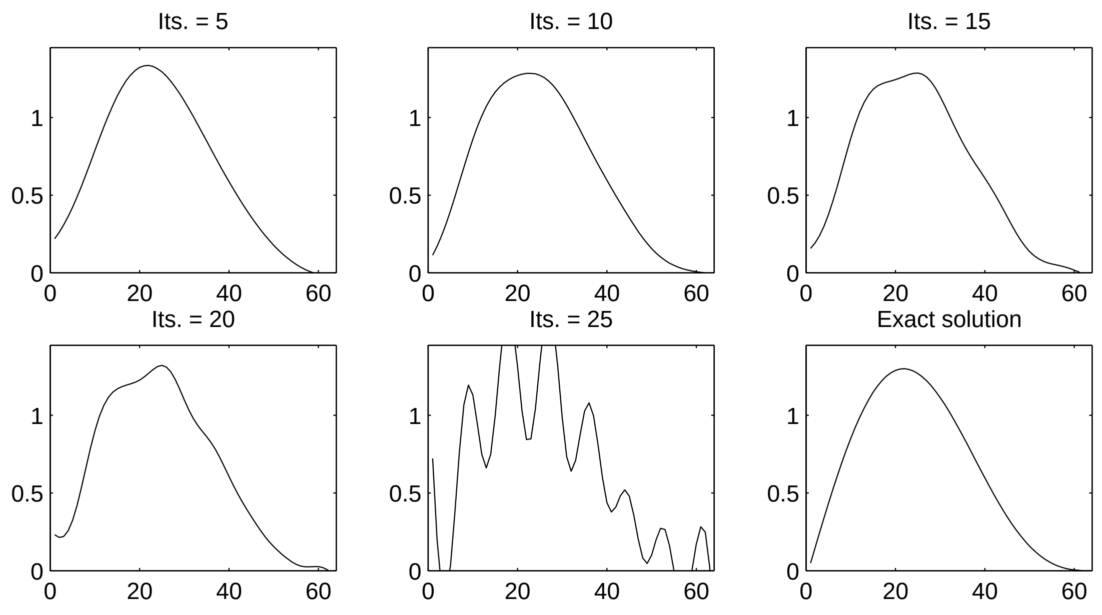


CGLS = Conjugate Gradients for Least Squares

The CGLS algorithm takes the following form:

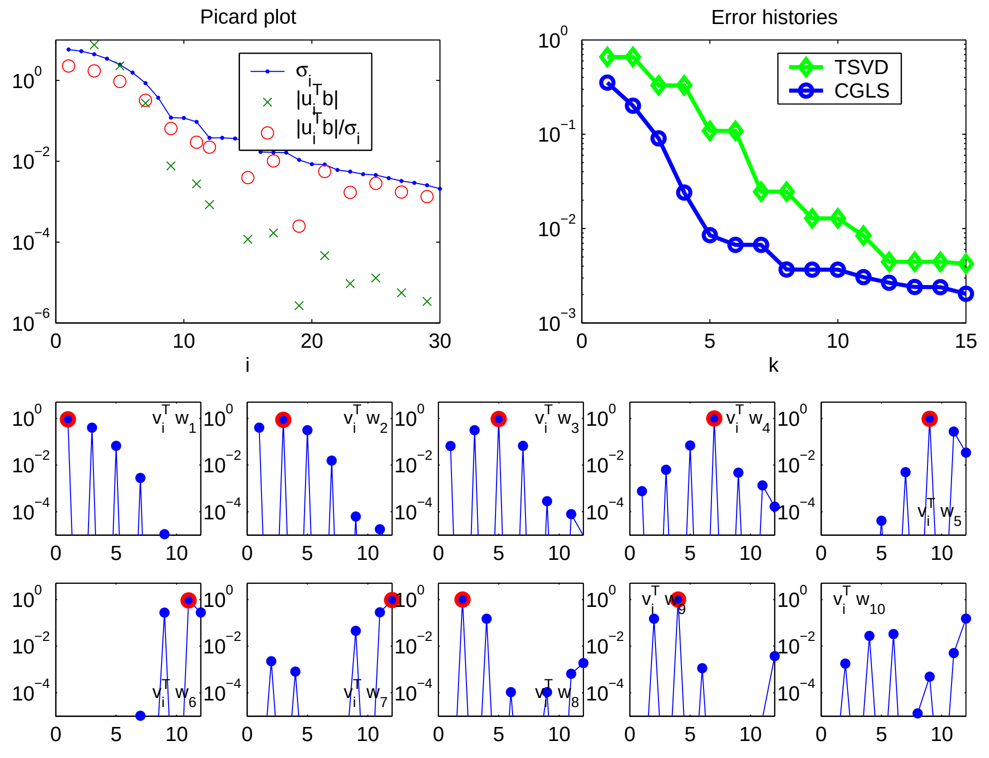
```
$x^{(0)}$ = starting vector (e.g., zero)
$r^{(0)} = b - A\,x^{(0)}$
$d^{(0)} = A^T r^{(0)}$
for $k = 1, 2, \ldots$
    $\bar{\alpha}_k = \|A^T r^{(k-1)}\|_2^2 / \|A\,d^{(k-1)}\|_2^2$
    $x^{(k)} = x^{(k-1)} + \bar{\alpha}_k\, d^{(k-1)}$
    $r^{(k)} = r^{(k-1)} - \bar{\alpha}_k\, A\, d^{(k-1)}$
    $\bar{\beta}_k = \|A^T r^{(k)}\|_2^2 / \|A^T r^{(k-1)}\|_2^2$
    $d^{(k)} = A^T r^{(k)} + \bar{\beta}_k\, d^{(k-1)}$
end
```

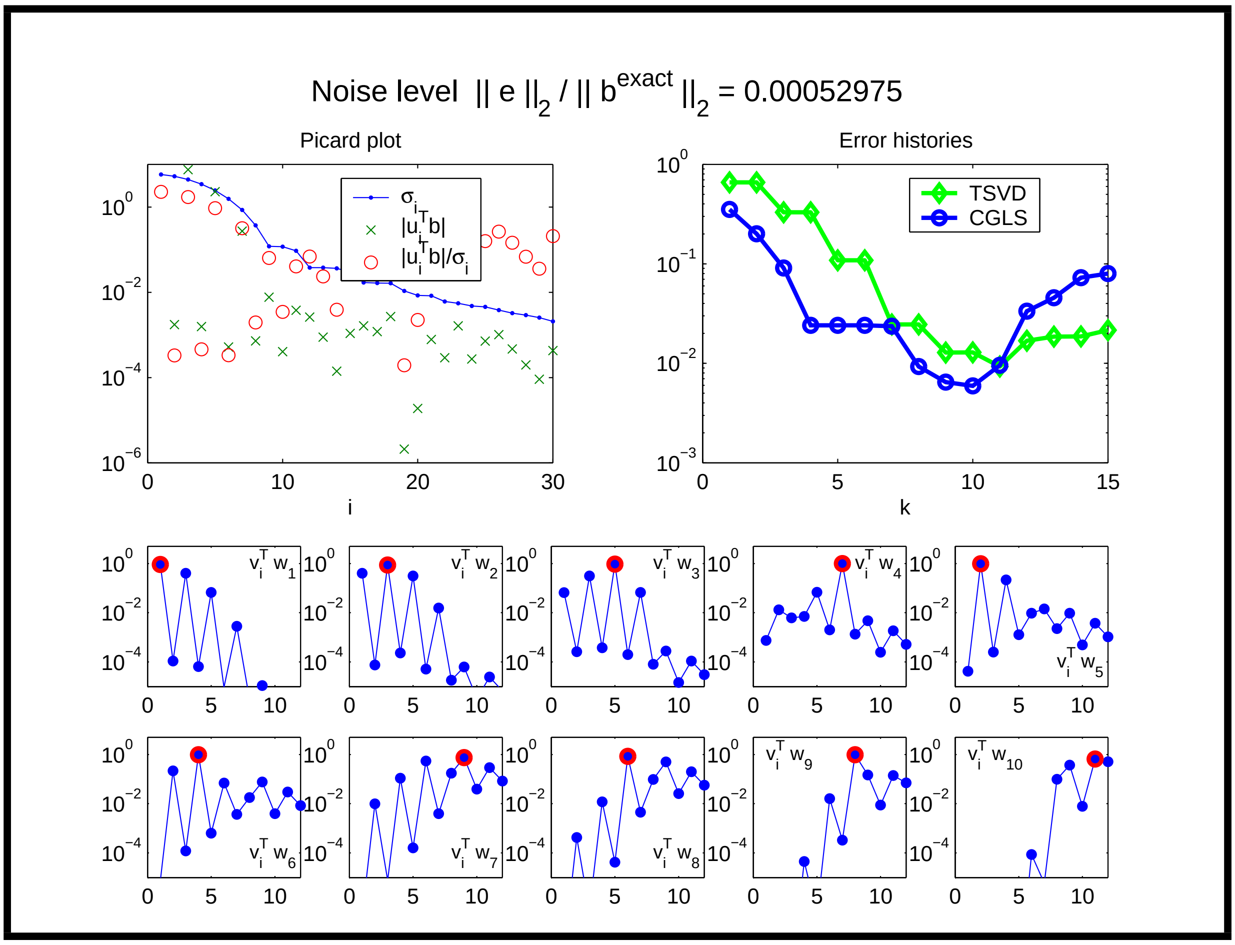
CGLS Solutions to the Gravity Problem



CGLS Focuses on the Significant Components

No noise





Other Iterations – GMRES and RRGMRES

Sometimes difficult or inconvenient to write a matrix-free black-box function for multiplication with A^T. Can we avoid this?

The GMRES method for square nonsymmetric matrices is based on the Krylov subspace

$$\mathcal{K}_k = \text{span}\{b, Ab, A^2 b, \ldots, A^{k-1} b\}.$$

The presence of the noisy data $b = b^{\text{exact}} + e$ in this subspace is unfortunate: the solutions include the noise component e!

A better subspace, underlying the RRGMRES method:

$$\vec{\mathcal{K}}_k = \text{span}\{A\,b, A^2\,b, \ldots, A^k\,b\}.$$

Now the noise vector is multiplied with A (smoothing) at least once.

Symmetric matrices: use MR-II (a simplified variant).

Tomography (a Case for Iterative Methods) §7.7

Tomography is the science of computing reconstructions from projections, i.e., data obtained by integrations along rays (typically straight lines) that penetrate a domain Ω.

The unknown function $f(\mathbf{t}) = f(t_1, t_2)$ represents some material parameter, and the damping of a signal penetrating a part $d\tau$ of a ray at position $\mathbf{t}$ is proportional to the product to $f(\mathbf{t})\, d\tau$.

The data consist of measurements of the damping of signals following well-defined rays through the domain Ω.

Formulation of Tomography Problem

The ith observation b_i, $i = 1, \dots, m$, represents the damping of a signal that penetrates Ω along a straight line, ray^i.

All the point $\mathbf{t}^i$ on ray^i are given by

$$\mathbf{t}^i(\tau) = \mathbf{t}^{i,0} + \tau\, \mathbf{d}^i, \qquad \tau \in \mathbb{R},$$

where $\mathbf{t}^{i,0}$ is an arbitrary point on the ray, and $\mathbf{d}^i$ is a (unit) vector that points in the direction of the ray.

The damping is proportional to the integral of the function $f(\mathbf{t})$ along the ray. Specifically, for the ith observation, the damping associated with the ith ray is given by

$$b_i = \int_{-\infty}^{\infty} f\big(\mathbf{t}^i(\tau)\big)\, d\tau, \qquad i = 1, \dots, m,$$

where $d\tau$ denotes the integration along the ray.

Discretization of 2D Tomography Problem

We consider a square domain $\Omega = [0,1] \times [0,1]$.

We can discretize this problem by dividing Ω into an $N \times N$ array of pixels, and in each pixel (k,ℓ) we assume that the function $f(\mathbf{t})$ is a constant $f_{k\ell}$:

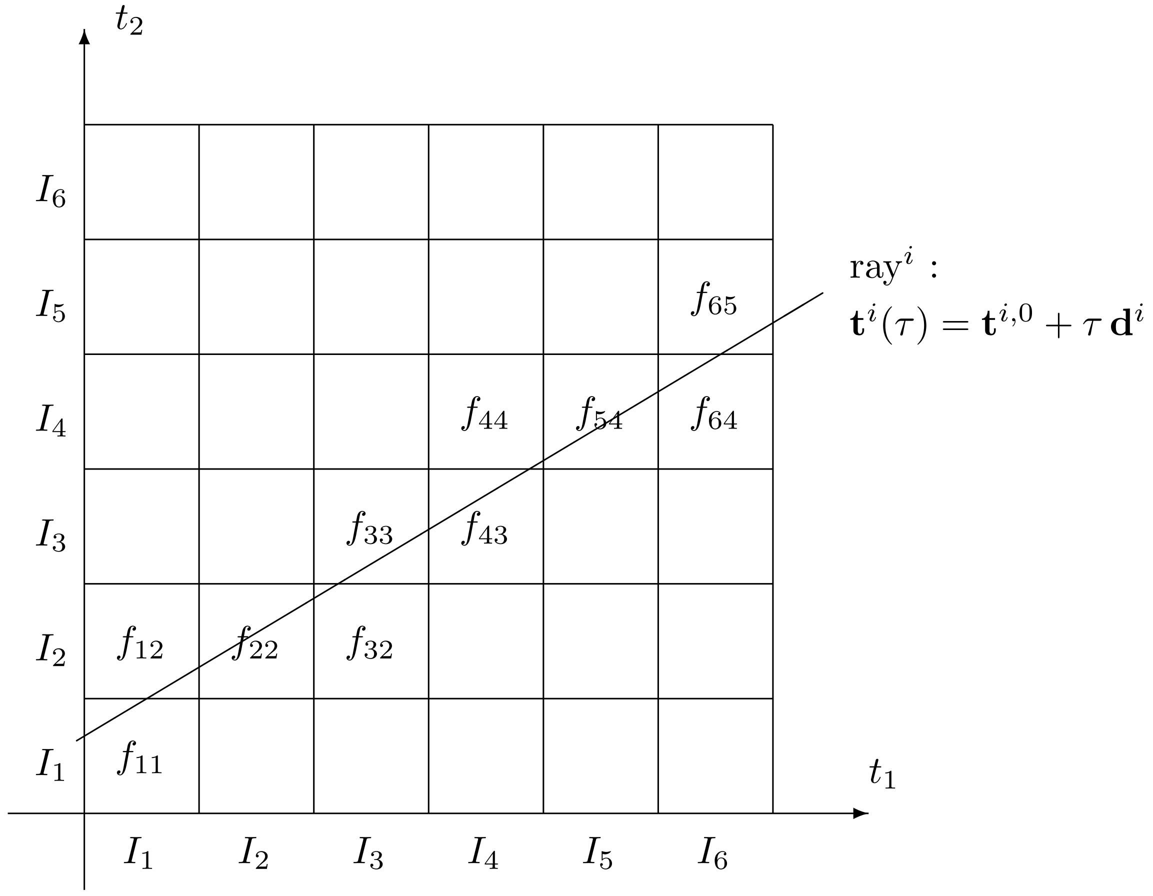
$$f(\mathbf{t}) = f_{k\ell} \qquad \text{for} \qquad t_1 \in I_k \;\&\; t_2 \in I_\ell,$$

where we have defined the interval $I_k = [\,(k{-}1)/N\,,\, k/N\,]$, $k = 1,\ldots,N$ (and similarly for I_ℓ).

With this assumption about $f(\mathbf{t})$ being piecewise constant, the expression for the kth measurement takes the simpler form

$$b_i = \sum_{(k,\ell)\in \text{ray}^i} f_{k\ell}\, \Delta L^{(i)}_{k\ell}, \qquad \Delta L^{(i)}_{k\ell} = \text{length of ray}_i \text{ in pixel } (k,\ell)$$

for $i = 1,\ldots,m$.



t_2
t_1
I_1 I_2 I_3 I_4 I_5 I_6
f_{11} f_{12} f_{22} f_{32} f_{33} f_{43} f_{44} f_{54} f_{64} f_{65}
ray^i :
$\mathbf{t}^i(\tau) = \mathbf{t}^{i,0} + \tau\, \mathbf{d}^i$

Arriving at the System of Linear Equations

The above equation is, in fact, a system of linear equations in the N^2 unknowns $f_{k\ell}$. We introduce the vector x of length $n = N^2$ whose elements are the (unknown) function values $f_{k\ell}$:

$$x_\ell = f_{k\ell}, \qquad \ell = (k-1)\,N + \ell.$$

This corresponds to stacking the columns of the $N \times N$ matrix F. Moreover we organize the measurements b_i into a vector b.

There is clearly a linear relationship between the data b_k and the unknowns in the vector x, meaning that we can always write

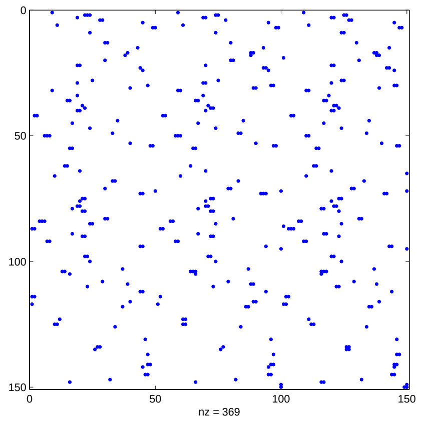
$$b_i = \sum_{j=1}^{n} a_{ij}\, x_j, \qquad i = 1, \ldots, m.$$

This is a system of linear equations $A\,x = b$ with an $m \times n$ matrix.

The elements of A are given by

$$a_{ij} = \begin{cases} \Delta L_{k\ell}^{(i)}, & (k,\ell) \in \text{ray}_i \\ 0 & \text{else} \end{cases}$$

where index i denotes the ith observation and j denotes the pixel number in an ordering with $j = (k-1)\,N + \ell$. The matrix A is very sparse.


nz = 369