Lecture 1: Numerical Issues from Inverse Problems (Parameter Estimation, Regularization Theory, and Parallel Algorithms)

Youzuo Lin ¹
Joint work with: Rosemary A. Renaut ²
Brendt Wohlberg ¹
Hongbin Guo ²

1. Los Alamos National Laboratory
2. School of Mathematical and Statistical Sciences, Arizona State University

Graduate Student Workshop on Inverse Problem, 2016
Outline

1. Introduction: Inverse Problem and Regularization
2. Topic 1: Multisplitting for Regularized Least Squares
   - Regularization Parallelization
   - Multiple Right Hand Side Problem
3. Topic 2: Total-Variation Regularization
   - Numerical Methods for Total Variation
   - Optimal Parameter Selection, UPRE Approach
4. Topic 3: Projected Krylov Subspace Solvers on GPU
   - Fine-Grained Parallelism Model for Krylov Subspace Solvers on GPU
   - Optimizations of Krylov Subspace Algorithms on GPU
   - Numerical Results
5. References
Outline

1. Introduction: Inverse Problem and Regularization
2. Topic 1: Multisplitting for Regularized Least Squares
   Regularization Parallelization
   Multiple Right Hand Side Problem
3. Topic 2: Total-Variation Regularization
   Numerical Methods for Total Variation
   Optimal Parameter Selection, UPRE Approach
4. Topic 3: Projected Krylov Subspace Solvers on GPU
   Fine-Grained Parallelism Model for Krylov Subspace Solvers on GPU
   Optimizations of Krylov Subspace Algorithms on GPU
   Numerical Results
5. References
Inverse Problems and Ill-posedness

- **General Linear Inverse Problem**

  \[ b = Af + \eta, \]

  Measured data \( b \) and system (or transform) matrix \( A \)

  \[
  \downarrow
  \]

  Input Data \( f \)
• General Linear Inverse Problem

\[ \mathbf{b} = A \mathbf{f} + \eta, \]

Measured data \( \mathbf{b} \) and system (or transform) matrix \( A \)

\[ \Downarrow \]

Input Data \( \mathbf{f} \)

• Image Restoration - Deconvolution

Transform matrix \( A \) is blurring matrix,
\( \eta \) is the noise
Inverse Problems and Ill-posedness

- **General Linear Inverse Problem**
  \[ b = Af + \eta, \]
  Measured data \( b \) and system (or transform) matrix \( A \)
  \[ \downarrow \]
  Input Data \( f \)

- **Image Restoration - Deconvolution**
  Transform matrix \( A \) is blurring matrix,
  \( \eta \) is the noise

- **Image Reconstruction - Tomography**
  Transform matrix \( A \) is Radon transform, \( \eta \) is the noise
Inverse Problems and Ill-posedness

- **General Linear Inverse Problem**
  \[ b = Af + \eta, \]
  Measured data \( b \) and system (or transform) matrix \( A \)
  ↓
  Input Data \( f \)

- **Image Restoration - Deconvolution**
  Transform matrix \( A \) is blurring matrix, \( \eta \) is the noise

- **Image Reconstruction - Tomography**
  Transform matrix \( A \) is Radon transform, \( \eta \) is the noise
How about $f = A^{-1}b$?
Direct Inverse Filtering

- How about $f = A^{-1}b$?

**Figure:** Left: Blurred and Noisy Image with SNR of 12.21 (dB), Right: Restored Image by Direct Inversion with SNR of -11.17 (dB)
Constrained Least Squares and Regularization Model

Constrained Least Squares:

$$\min \{ J(f) \} ,$$

subject to \( \| Af - b \|_2^2 = \sigma^2. \)

Regularization Model:

$$\min \left\{ \| Af - b \|_2^2 + \lambda^2 J(f) \right\} , \quad \lambda > 0,$$

\( \| Af - b \|_2^2 \) : Data Fidelity Term

\( J(f) \) : Regularization Term

\( \lambda \) : Regularization Parameter
Specific Regularization

**Tikhonov Regularization:**
When $J(f)$ is a quadratic functional $f^T L^T L f$,

$$\min_f \left\{ \| A f - b \|_2^2 + \lambda^2 \| L f \|_2^2 \right\},$$

**Total Variation (TV) Regularization:**
When $J(f)$ is a nonquadratic functional like Total Variation,

$$\min_f \left\{ \| A f - b \|_2^2 + \lambda^2 \| \nabla f \|_1 \right\},$$

**Others:**
Learning Based Regularization, Compressive Sensing etc.
Outline

1 Introduction: Inverse Problem and Regularization

2 Topic 1: Multisplitting for Regularized Least Squares
   Regularization Parallelization
   Multiple Right Hand Side Problem

3 Topic 2: Total-Variation Regularization
   Numerical Methods for Total Variation
   Optimal Parameter Selection, UPRE Approach

4 Topic 3: Projected Krylov Subspace Solvers on GPU
   Fine-Grained Parallelism Model for Krylov Subspace Solvers on GPU
   Optimizations of Krylov Subspace Algorithms on GPU
   Numerical Results

5 References
Topic 1: Multisplitting for Regularized Least Squares [Renaut, Lin and Guo, 2009]

- Regularization Parallelization
  - Bottleneck for Applications
  - Formulation-Multisplitting and Domain Decomposition
  - Local Solver Selection and Computational Cost
  - Update Scheme
- Multiple Right-Hand Sides Problem
  - A Second Look at Multisplitting
  - Hybrid Regularization with Multisplitting and Multiple Right Hand Sides
  - Computational Cost Analysis
  - Application in Image Reconstruction and Restoration
- Numerical Results
Tikhonov Solution and Memory Bottleneck

• Solving for the TK regularization is equivalent to solving for a large scale linear system:

\[(A^T A + \lambda^2 L^T L) f = A^T b.\]

• What about RAM requirement? In the positron emission tomography (PET) image reconstruction example in single precision, for an image of size 256 \(\times\) 256, with projection angles from 0° to 180° with a 2° increment, the required RAM for storing \(A\) is:

\[
\frac{65536 \times 33030 \times 4}{1024^3} \approx 8.02\,\text{GB}!
\]

• What can we do about this? We apply Domain Decomposition idea to our problem.
Linear multisplitting (LMS): Given a matrix $A \in \mathbb{R}^{n \times n}$ and a collection of matrices $M^{(i)}, N^{(i)}, E^{(i)} \in \mathbb{R}^{n \times n}, i = 1, \ldots, p$ satisfying

1. $A = M^{(i)} - N^{(i)}$ for each $i, i = 1, \ldots, p$.
2. $M^{(i)}$ is regular (nonsingular), $i = 1, \ldots, p$.
3. $E^{(i)}$ is a non-negative diagonal matrix, $i = 1, \ldots, p$ and $\sum_{i=1}^{p} E^{(i)} = I$.

Then the collection of triples $(M^{(i)}, N^{(i)}, E^{(i)}), i = 1, \ldots, p$ is called a multisplitting of $A$ and the LMS method is defined by the iteration:

$$x^{(k+1)} = \sum_{i=1}^{p} E^{(i)} (M^{(i)})^{-1} (N^{(i)} x^{(k)} + fb).$$
Formulation - Domain Decomposition

- Domain decomposition is a particular method of multisplitting in which the submatrices $M^{(i)}$ are defined to be consistent with a partitioning of the domain:

$$x = (x_1^T, x_2^T, ..., x_p^T)^T.$$
Domain decomposition is a particular method of multisplitting in which the submatrices $M^{(i)}$ are defined to be consistent with a partitioning of the domain:

$$x = (x_1^T, x_2^T, \ldots, x_p^T)^T.$$
Corresponding to different splitting of the image $x$, we will have splitting on kernel operator $A$ as,

$$A = (A_1, A_2, ..., A_p),$$
Corresponding to different splitting of the image $x$, we will have splitting on kernel operator $A$ as,

$$A = (A_1, A_2, ..., A_p),$$

The linear system $Ax \approx b$ is replaced with the split systems

$$A_i y_i \approx b_i(x),$$

where $b_i(x) = b - \sum_{j \neq i} A_j x_j = b - Ax + A_i x_i$. 
Local Problem - Solving $Ax \approx b$ turns into solving $p$ subproblems,

$$\min_{y \in \mathbb{R}^{n_i}} \| A_i y_i - b_i(x) \|_2, \quad 1 \leq i \leq p.$$
Formulation - Local Problem

- **Local Problem - Solving** $Ax \approx b$ turns into solving $p$ subproblems,

$$\min_{y \in \mathbb{R}^{n_i}} \| A_i y_i - b_i(x) \|_2, \quad 1 \leq i \leq p.$$ 

- **Update Scheme** - The global solution update from local solutions at step $K$ to step $K + 1$ is given by,

$$x^{(k+1)} = \sum_{i=1}^{p} \tau_i^{(k+1)} (x_{local})^{(k+1)}_i,$$

where $(x_{local})^{(k+1)}_i =$

$$(((x_1^{(k)})^T, \ldots, (x_{i-1}^{(k)})^T, (y_i^{(k+1)})^T, (x_{i+1}^{(k)})^T, \ldots, (x_p^{(k)})^T)^T)^T.$$
Formulation - Regularized Multisplitting

- Tikhonov Regularization:

\[
\min_x \left\{ \| Ax - b \|_2^2 + \lambda^2 \| Lx \|_2^2 \right\}, \quad \lambda > 0,
\]
• Tikhonov Regularization:

\[
\min_{x} \left\{ \| Ax - b \|_2^2 + \lambda^2 \| Lx \|_2^2 \right\}, \quad \lambda > 0,
\]

• equivalent to,

\[
\min \left\| \begin{pmatrix} A \\ L\Lambda \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2^2,
\]

where \( \Lambda \) is block diagonal with entries \( \lambda I_{n_i} \).
Similarly, we will have splitting on the operator as,

\[
\begin{pmatrix}
    A \\
    L\Lambda
\end{pmatrix} = \left( \begin{pmatrix}
    A_1 \\
    \lambda_1 L_1
\end{pmatrix} \begin{pmatrix}
    A_2 \\
    \lambda_2 L_2
\end{pmatrix} \cdots \begin{pmatrix}
    A_p \\
    \lambda_p L_p
\end{pmatrix} \right).
\]
Similarly, we will have splitting on the operator as,

\[
\begin{pmatrix}
A \\
L\Lambda
\end{pmatrix} = \begin{pmatrix}
\begin{pmatrix} A_1 \\ \lambda_1 L_1 \end{pmatrix} & \cdots & \begin{pmatrix} A_p \\ \lambda_p L_p \end{pmatrix}
\end{pmatrix}.
\]

\(i^{th}\) block equation of the normal equations is given by

\[
(A^T_i A_i + \lambda_i^2 L_i^T L_i)y_i = A_i^T b_i(x) - \lambda_i \sum_{j=1, j \neq i}^{p} \lambda_j L_i^T L_j x_j.
\]
Local Solver Selection and Computational Cost

- How to select the local solver?

Iterative Solver: Conjugate Gradient for Least Squares (CGLS)

- Computation Cost: $C_{\text{CGLS}} \approx 2n^2i\tilde{m} + K((2(ki + 1)n^2 + 10ki)n^i))$

- Memory Cost: Matrix vector multiplication, only need to store matrix $A$.

Solver: CGLS seems to be reasonable!
Local Solver Selection and Computational Cost

• How to select the local solver?
  • Computation Cost and Memory Cost

Inverse Theory
Inverse Problems Workshop 14 / 64
Local Solver Selection and Computational Cost

- How to select the local solver?
  - Computation Cost and Memory Cost
- Iterative Solver: Conjugate Gradient for Least Squares (CGLS)
Local Solver Selection and Computational Cost

- How to select the local solver?
  - Computation Cost and Memory Cost
- Iterative Solver: Conjugate Gradient for Least Squares (CGLS)
  - Computation Cost: $C_{\text{CGLS}} \approx 2n_i^2\tilde{m} + K(2(k_i + 1)n_i^2 + 10k_in_i)$
  - Memory Cost: Matrix vector multiplication, only need to store matrix $A_i$
Local Solver Selection and Computational Cost

- How to select the local solver?
  - Computation Cost and Memory Cost
- Iterative Solver: Conjugate Gradient for Least Squares (CGLS)
  - Computation Cost: \( C_{\text{CGLS}} \approx 2n_i^2 \tilde{m} + K(2(k_i + 1)n_i^2 + 10k_i n_i) \)
  - Memory Cost: Matrix vector multiplication, only need to store matrix \( A_i \)
- Solver: CGLS seems to be reasonable at this point!
Four Different Update Schemes

- **Block Jacobian Update**
  - Set $\tau_i^{(k+1)} = \tau_i = 1$,
  - so $x^{(k+1)} = ((y_1^{(k)})^T, (y_2^{(k)})^T, \ldots, (y_{i-1}^{(k)})^T, (y_i^{(k+1)})^T, (y_{i+1}^{(k)})^T, \ldots, (y_p^{(k)})^T)^T$. 

- **Convex Update**
  - Set $\tau_i^{(k+1)} = \tau_i = 1$,
  - $x^{(k+1)} = x^{(k)} + \sum_{i=1}^p \tau^{(k+1)}_i \left( y_i^{(k+1)} - x^{(k)}_i \right)$. 

- **Local Update**
  - $x^{(k+1)}_{\text{local}}^{(k+1)} = x^{(k+1)}_i$. 

- **Optimal $\tau$ Update**
  - $\min \tau_r(x^{(k+1)})$, where $r(x^{(k+1)}) = r(x^{(k)}) - \sum_{i=1}^p \tau^{(k+1)}_i (A_i \lambda_i L_i) \delta^{(k+1)}_i$. 

Inverse Theory
Four Different Update Schemes

- **Block Jacobian Update**
  - Set $\tau_i^{(k+1)} = \tau_i = 1$,
  - so
  \[
  x^{(k+1)} = ((y_1^{(k)})^T, (y_2^{(k)})^T, \ldots, (y_{i-1}^{(k)})^T, (y_i^{(k+1)})^T, (y_{i+1}^{(k)})^T, \ldots, (y_p^{(k)})^T)^T.
  \]

- **Convex Update**
  - Set $\tau_i^{(k+1)} = \tau_i = \frac{1}{p}$,
  - 
  \[
  x^{(k+1)} = x^{(k)} + \frac{1}{p}(y^{(k+1)} - x^{(k)}).
  \]
Four Different Update Schemes

- **Block Jacobian Update**
  - Set $\tau^{(k+1)}_i = \tau_i = 1$,
  
  so
  
  $$\mathbf{x}^{(k+1)} = ((\mathbf{y}_1^{(k)})^T, (\mathbf{y}_2^{(k)})^T, \ldots, (\mathbf{y}_{i-1}^{(k)})^T, (\mathbf{y}_i^{(k+1)})^T, (\mathbf{y}_{i+1}^{(k)})^T, \ldots, (\mathbf{y}_p^{(k)})^T)^T.$$ 

- **Convex Update**
  - Set $\tau^{(k+1)}_i = \tau_i = \frac{1}{p}$,
  
  $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \frac{1}{p} (\mathbf{y}^{(k+1)} - \mathbf{x}^{(k)}).$$ 

- **Local Update**
  - $\mathbf{x}^{(k+1)} = (\mathbf{x}_{local})^{(k+1)}_i.$
Four Different Update Schemes

- **Block Jacobian Update**
  - Set $\tau_i^{(k+1)} = \tau_i = 1$, so
  $$x^{(k+1)} = ((y_1^{(k)})^T, (y_2^{(k)})^T, \ldots, (y_{i-1}^{(k)})^T, (y_i^{(k+1)})^T, (y_{i+1}^{(k)})^T, \ldots, (y_p^{(k)})^T)^T.$$  

- **Convex Update**
  - Set $\tau_i^{(k+1)} = \tau_i = \frac{1}{p}$,
  $$x^{(k+1)} = x^{(k)} + \frac{1}{p}(y^{(k+1)} - x^{(k)}).$$

- **Local Update**
  $$x^{(k+1)} = (x_{\text{local}})^{(k+1)}_i.$$  

- **Optimal $\tau$ Update**
  $$\min_{\tau} r(x^{(k+1)}),$$
  where
  $$r(x^{(k+1)}) = r(x^{(k)}) - \sum_{i=1}^{p} \tau_i^{(k+1)} \left( A_i \right) \left( \lambda_i L_i \right) \delta_i^{(k+1)}.$$
The $i^{th}$ block equation of the normal equations is given by

$$(A_i^T A_i + \lambda_i^2 L_i^T L_i)y_i = A_i^T b_i(x) - \lambda_i \sum_{j=1,j \neq i}^p \lambda_j L_i^T L_j x_j.$$  

- The system matrix is unchanged;
Rewrite the right hand side (RHS) as,

\[
b_i(x^{(k+1)}) = b_i(x^{(k)}) - \left( \sum_{j=1, j \neq i}^{p} \tau_j^{(k+1)} B_j^{(k+1)} \right) + OB_i^{(k+1)},
\]

where \( B_j^{(k+1)} = A_j(x_j^{(k+1)} - x_j^{k}) \), \( OB_i^{(k+1)} \) is the update of the overlapped problem.

- The new RHS is updated based on the previous RHS;
• Multiple RHS Problem Setup

\[ AX = B = [b^{(1)}, \ldots, b^{(s)}] , \]

where \( A \in \mathbb{R}^{m \times n} \), \( X \in \mathbb{R}^{n \times s} \) and \( B \in \mathbb{R}^{m \times s} \).
Linear System with Multiple RHS

- Multiple RHS Problem Setup
  \[ AX = B = [b^{(1)}, \ldots, b^{(s)}], \]
  where \( A \in \mathbb{R}^{m \times n}, \ X \in \mathbb{R}^{n \times s} \) and \( B \in \mathbb{R}^{m \times s} \).

- What about solving \( s \) systems independently?
  - Direct Method (LU): Major Computational Cost \( \approx O(n^3) + s \cdot O(n^2) \);
  - Iterative Method (CGLS): Major Computational Cost \( \approx s \cdot O(n^2) \);
Linear System with Multiple RHS

- Multiple RHS Problem Setup

\[ AX = B = [b^{(1)}, \ldots, b^{(s)}], \]

where \( A \in \mathbb{R}^{m \times n}, X \in \mathbb{R}^{n \times s} \) and \( B \in \mathbb{R}^{m \times s} \).

- What about solving \( s \) systems independently?
  - Direct Method (LU): Major Computational Cost \( \approx O(n^3) + s \cdot O(n^2) \);
  - Iterative Method (CGLS): Major Computational Cost \( \approx s \cdot O(n^2) \);

- Several algorithms have been proposed to speed up solving the above system.
Efficient Iterative Algorithm for Solving Multiple RHS

- If $b^{(1)}, \ldots, b^{(s)}$ are random or completely orthogonal to each other, no way to speed up the algorithm!
Efficient Iterative Algorithm for Solving Multiple RHS

- If $b^{(1)}, \ldots, b^{(s)}$ are random or completely orthogonal to each other, no way to speed up the algorithm!

- If $b^{(1)}, \ldots, b^{(s)}$ are close to each other and share information
  - Galerkin-Projection based Krylov subspace methods are proposed for an efficient solver.
  - We choose CG as the specific Krylov subspace method proposed in [Chan and Wan, 1997], because it is well-supported theoretically.
Basic Idea of Projected CG Algorithm for Solving MRHS

- Step 1: Solve the *seed* system.
  - Select the seed;
  - Apply the CGLS algorithm for solving seed system and save the conjugate directions for *Step 2*;
Basic Idea of Projected CG Algorithm for Solving MRHS

- Step 1: Solve the **seed** system.
  - Select the seed;
  - Apply the CGLS algorithm for solving seed system and save the conjugate directions for **Step 2**;

- Step 2: Solve the remaining system by **projection**
  - Use the conjugate directions from the seed systems to span the Krylov subspace
  - Project the systems onto the Krylov subspace for an approximated solution
Basic Idea of Projected CG Algorithm for Solving MRHS

- Step 1: Solve the seed system.
  - Select the seed;
  - Apply the CGLS algorithm for solving seed system and save the conjugate directions for Step 2;

- Step 2: Solve the remaining system by projection
  - Use the conjugate directions from the seed systems to span the Krylov subspace
  - Project the systems onto the Krylov subspace for an approximated solution

- Step 3: **Restart** the seed and **Refine** the solution
  - A new seed might be selected from the remaining systems to restart the procedure.
  - Refine the approximation, if the accuracy is insufficient.
Basic Idea of Projected CG Algorithm for Solving MRHS

- **Step 1:** Solve the *seed* system.
  - Select the seed;
  - Apply the CGLS algorithm for solving seed system and save the conjugate directions for **Step 2**;

- **Step 2:** Solve the remaining system by *projection*
  - Use the conjugate directions from the seed systems to span the Krylov subspace
  - Project the systems onto the Krylov subspace for an approximated solution

- **Step 3:** **Restart** the seed and **Refine** the solution
  - A new seed might be selected from the remaining systems to restart the procedure.
  - Refine the approximation, if the accuracy is insufficient.
Adapt to Our Problem: Solving Each Subproblem from RLSMS, Scheme 1

- **Scheme 1 (Project/Restart)**: Treat the $i^{th}$ subproblem at all iteration as a Multiple RHS with $s = 2$, i.e. at iteration $k = 1$, solve the subproblem as seed, for iteration $k > 1$, solve it as projection and refine it as necessary;
Adapt to Our Problem: Solving Each Subproblem from RLSMS, Scheme 1

- **Scheme 1 (Project/Restart):** Treat the $i^{th}$ subproblem at all iteration as a Multiple RHS with $s = 2$, i.e. at iteration $k = 1$, solve the subproblem as seed, for iteration $k > 1$, solve it as projection and refine it as necessary;

- **Comment:** Straight forward adaptation, but usually the accuracy of approximation remaining after projection is not enough; Need extra steps for further refinement;
• **Scheme 2 (Project/Augment):** Based on seed selection as in **Scheme 1**, in the refinement stages of solving the remaining systems, stores the newly generated conjugate directions with those from solving seeds;
Adapt to Our Problem: Solving Each Subproblem from RLSMS, Scheme 2

- **Scheme 2 (Project/Augment):** Based on seed selection as in **Scheme 1**, in the refinement stages of solving the remaining systems, stores the newly generated conjugate directions with those from solving seeds;

- **Comment:** The motivation of this change is again to improve the Krylov subspace. By updating the conjugate directions, we will be able to add in new information which is not from solving the seed system. Numerical tests indicate this scheme works best with large scale problems!
The computation cost of projected CGLS consists of two other components: one is the cost for solving the seed system, which is the cost of the CGLS; the other is the cost for solving the remaining systems, which includes the projection and the few more steps of CGLS as needed:

\[ C_{ProjCG} \approx 2(K + 2k_i + 2)mn_i + (6(K - 1)k_i + 10k_i + 1)n_i + \mu \cdot CG. \]
**Figure:** Restoration results using different methods for the sample of Phillips size of $1024 \times 1$, noise level 6%.
Figure: Maximum numbers of inner iterations against outer iteration count.
2-D Seismic Reconstruction, Phantom Size $64 \times 64$

**Figure:** SNR (Without Splitting): 11.65 dB, SNR (CGLS/Zero): 11.68 dB, SNR (Project/Restart): 11.63 dB, SNR (Project/Augment): 11.60 dB.
2-D Seismic Reconstruction, Global and Local Iteration Result

**Figure**: Maximum numbers of inner iterations against outer iteration count.
1 Introduction: Inverse Problem and Regularization

2 Topic 1: Multisplitting for Regularized Least Squares
   - Regularization Parallelization
   - Multiple Right Hand Side Problem

3 Topic 2: Total-Variation Regularization
   - Numerical Methods for Total Variation
   - Optimal Parameter Selection, UPRE Approach

4 Topic 3: Projected Krylov Subspace Solvers on GPU
   - Fine-Grained Parallelism Model for Krylov Subspace Solvers on GPU
   - Optimizations of Krylov Subspace Algorithms on GPU
   - Numerical Results

5 References
Topic 2: Total-Variation Regularization

- Numerical Methods for Total Variation
  - Iteratively Reweighted Norm Algorithm
  - Lagged Diffusivity Fixed Point Iteration Algorithm
- Optimal Regularization Parameter Selection,
  - Trace Approximation
  - UPRE Approach
- Numerical Results
Approximate L1 norm by weighted L2 norm, TV regularization becomes:

$$\min_f \left\{ \frac{1}{2} \| Af - b \|_2^2 + \frac{\lambda}{2} \| W_r^{1/2} Df \|_2^2 \right\},$$

$$\iff \min_f \left\{ \frac{1}{2} \left\| \begin{pmatrix} I & 0 \\ 0 & W_r^{1/2} \end{pmatrix} \left( \begin{pmatrix} A \\ \sqrt{\lambda}D \end{pmatrix} f - \begin{pmatrix} b \\ 0 \end{pmatrix} \right) \right\|_2^2 \right\},$$

where $W_r = \text{diag}(2((D_x f)^2 + (D_y f)^2)^{-\frac{1}{2}}).$

This minimization problem can be solved by CG solver.

Approximate L1 norm by, \[ \sum \psi(f) = \sum \sqrt{\left( \frac{df}{dx} + \beta^2 \right)} \]

Gradient \[ = A^T(Af - b) + \lambda L(f)f, \]

Approximate Hessian \[ = A^T A + \lambda L(f), \]

where \[ L(f) = \triangle x D^T \text{diag}(\psi'(f)) D. \]

Based on the Quasi-Newton idea, we solve for \( f \) iteratively by,

\[ f_{k+1} = f_k - (A^T A + \lambda L(f_k))^{-1} \text{Gradient}, \]

which is called as “Lagged Diffusivity Fixed Point Iteration”.

Image Deblurring Comparison using TV and Tikhonov

**Figure:** Left: Original Clean Image, Right: Blurred and Noisy Image
Image Deblurring Comparison using TV and Tikhonov

**Figure:** *Left:* TK based Deblurred Result, *Right:* TV Based Deblurred Result
Problem Description - Simple Example

True Solution (-) and TV Regularized Solution (--), $\lambda = 0.000001$

SNR plot, current SNR: 17.013637 dB
Problem Description-Simple Example

True Solution (--) and TV Regularized Solution (--), \( \lambda = 0.000012 \)

SNR plot, current SNR: 17.015130 dB
Problem Description-Simple Example

True Solution (- -) and TV Regularized Solution (- -), $\lambda = 0.000100$

SNR plot, current SNR: 17.640445 dB
Problem Description-Simple Example

True Solution (-) and TV Regularized Solution (-), $\lambda = 0.004764$

SNR plot, current SNR: 10.526417 dB
Problem Description - Simple Example

True Solution (--) and TV Regularized Solution (--), $\lambda = 0.042376$

SNR plot, current SNR: 22.243069 dB
Problem Description-Simple Example

True Solution (--) and TV Regularized Solution (--), $\lambda = 0.073192$

SNR plot, current SNR: 23.447632 dB
Problem Description - Simple Example

True Solution (-) and TV Regularized Solution (-), $\lambda = 0.095172$

SNR plot, current SNR: 24.055030 dB
Problem Description-Simple Example

True Solution (--) and TV Regularized Solution (--), \( \lambda = 0.125305 \)

SNR plot, current SNR: 23.723666 dB
Problem Description-Simple Example

True Solution (--) and TV Regularized Solution (--), \( \lambda = 0.210264 \)

SNR plot, current SNR: 20.905694 dB
Problem Description-Simple Example

True Solution (--) and TV Regularized Solution (-), $\lambda = 0.650960$

SNR plot, current SNR: 14.010823 dB
Problem Description-Simple Example

True Solution (--) and TV Regularized Solution (--), $\lambda = 2.551407$

SNR plot, current SNR 5.355005 dB
Problem Description - Simple Example

True Solution (--) and TV Regularized Solution (--), $\lambda = 10.000000$

SNR plot, current SNR: 0.008276 dB
Fundamental Idea of UPRE

Derived from the Mean Squared Error (MSE) of the predictive error,

\[ \frac{1}{n} \| P_\lambda \|^2 = \frac{1}{n} \| A f_\lambda - A f_{\text{true}} \|^2, \]

where \( f_\lambda \) is the regularized solution,
\( f_{\text{true}} \) is the true solution.
To avoid using \( f_{\text{true}} \), we define a functional which is the unbiased estimator of the above, and name it UPRE (Unbiased Predictive Risk Estimator).

\[ \lambda_{\text{opt}} = \arg \min_\lambda \text{UPRE}(\lambda) \]
Vogel in [1] proved that, because the solution for Tikhonov Regularization is linearly dependent on the right hand side of (2), i.e.,

\[ f_{TK} = R_{TK}(b + \eta), \]

where \( R_{TK} = (A^T A + \lambda I)^{-1} A^T \), we will have

\[
\text{UPRE}_{\text{Tikhonov}}(\lambda) = E\left(\frac{1}{n}||P_\lambda||^2\right) = \frac{1}{n}||r_\lambda||^2 + \frac{2\sigma^2}{n} \text{trace}(A_\lambda) - \sigma^2,
\]

where \( r_\lambda = Af_\lambda - b \), \( A_\lambda = A(A^T A + \lambda I)^{-1} A^T \), and \( \sigma \) is the variance of the noise.
UPRE for TV

- Difficulty 1: Nonquadratic penalty functional in TV term.
- Hint: Linearization Approximation!
- In Lagged Diffusivity approach,

\[ f_{TV} = R_{TV}(b + \eta), \]

where \( R_{TV} = (A^T A + \frac{\lambda}{2} L(f_k))^{-1} A^T \)

the influence matrix,

\[ A_\lambda = A(A^T A + \frac{\lambda}{2} L(f_k))^{-1} A^T. \]

\( A_\lambda \) can be proved to be symmetric, which will give us the same derivation of the UPRE functional for TV.
Difficulty 2: How to compute
\[
\text{trace}(A_\lambda) = \text{trace}(A(A^T A + \frac{\lambda}{2} L(f_k))^{-1} A^T)
\]

Hint: Krylov Space Method!

Basic Idea:
\[
\text{trace}(f(M)) \simeq E(u^T f(M)u),
\]
where \(u\) is a discrete multivariate random variable, where each entry takes the values -1 and +1 with probability 0.5, and the matrix \(M\) is symmetric positive definite (SPD).
Golub [2] pointed out that,

\[ E(u^T f(M)u) = \sum_{i=1}^{k} \omega_i f(\theta_i), \]

where \( \theta_i \) are the eigenvalues of \( M \), and \( \omega_i \) are squares of the first components of the normalized eigenvectors of \( M \).

Start the Lanczos procedure on \( M \), and find out the most significant eigenvectors until accuracy is satisfied.
Trace Approximation Results

**Figure:** Left: Accurate Trace Vs. Estimated Trace, Right: Time Cost for Computing Accurate Trace Vs. Time Cost for Computing Estimated Trace
Deblurred Image, TV (lagged) -- lambda = 0.0015, SNR: 15.1 dB. Run time: 10.8s

Deblurred Image, TV (lagged) -- lambda = 8.5000, SNR: 10.9 dB. Run time: 40.6s

**Figure:** **Left:** Use Optimal Parameter, **Right:** Use Random Parameter
More General Results for Deblurring

**Figure:** Plot of Optimal SNR and Estimated SNR versus original noisy SNR on satellite image.

**Figure**

Plot of Optimal SNR and Estimated SNR versus original noisy SNR on satellite image.
Figure: Plot of Optimal SNR and Estimated SNR versus original noisy SNR on satellite image. Ref: “G. Gilboa, N. Sochen, Y. Y. Zeevi, Estimation of optimal PDE-based denoising in the SNR sense, IEEE Transactions on Image Processing 15(8), 2006.”
Conclusions

• Tikhonov Regularization:
  • Pros: easy to implement, cheap cost;
  • Cons: the result is usually smoothed out;

• Total Variation (TV) Regularization:
  • Pros: Preserve image edges provided with reasonable parameter, piecewise constant
  • Cons: Expensive cost
1. Introduction: Inverse Problem and Regularization

2. Topic 1: Multisplitting for Regularized Least Squares
   - Regularization Parallelization
   - Multiple Right Hand Side Problem

3. Topic 2: Total-Variation Regularization
   - Numerical Methods for Total Variation
   - Optimal Parameter Selection, UPRE Approach

4. Topic 3:Projected Krylov Subspace Solvers on GPU
   - Fine-Grained Parallelism Model for Krylov Subspace Solvers on GPU
   - Optimizations of Krylov Subspace Algorithms on GPU
   - Numerical Results

5. References
Topic 3: Projected Krylov Subspace Solvers on GPU [Lin and Renaut, 2010]

- Problem Description and Krylov Subspace Solvers
  - 3-D Medical Image Reconstruction
  - Inverse Linear Systems with Multiple Right-hand Sides
  - Conjugate Gradient Algorithm
- Fine-Grained Parallelism Model for Krylov Subspace Solvers on GPU
  - GPU Structure
  - BLAS Performance on GPU
  - CG on GPU (ver 0)
- Optimizations of Krylov Subspace Algorithms on GPU
  - Projected CG
  - Modified Projected CG
- Numerical Results
3-D Medical Image Reconstruction

Figure: Image Slices from a 3-D Scan
Inverse Linear Systems with Multiple Right-hand Sides

- Multiple Right-hand Sides System:

\[ Ax = B = [b^{(1)}, \ldots, b^{(s)}], \]

where \( b^i \) stands for the \( i^{th} \) measurement with respect to the \( i^{th} \) slice as in a 3-D medical image.

- Tikhonov regularization is applied to the above systems.

- Regularization parameter is assumed to be close enough to the optimal value.
Conjugate Gradient Algorithm

- \( k = 0, r_0 = b - Ax_0; \)
  while (\( ||r_k||_2 > \text{TOL} \times ||b||_2 \))
    \( k = k + 1; \)
    if (\( k == 1 \))
      \( p_1 = r_0; \)
    else
      \( \beta_k = \langle r_{k-1}, r_{k-1} \rangle / \langle r_{k-2}, r_{k-2} \rangle; \)
      \( p_k = r_{k-1} + \beta_k p_{k-1}; \)
    end;
    \( \alpha_k = \langle r_{k-1}, r_{k-1} \rangle / \langle p_k, Ap_k \rangle; \)
    \( x_k = x_{k-1} + \alpha_k p_k; \)
    \( r_k = r_{k-1} - \alpha_k Ap_k; \)
  end (while).
Figure: NVIDIA Tesla C1060
Streaming Multiprocessors (SM): 30
Streaming Processors (SP): 8
Register: 16 K per SM
Shared Memory: 16 KB per SM
Constant Memory: 64 KB
Global Memory: 4 GB
Bandwidth: 102 GB/Sec

GFLOPS (PEAK)
Single Precision: 933
Double Precision: 78

Figure: Structure of NVIDIA Tesla C1060
BLAS Performance on GPU - Matrix Vector Multiplication

- Testing Environment
  Operating System: Ubuntu, 64 bit
  CPU (Host): Intel Quad Core Xeon E5506, 2.13 GHz
  GPU (Device): NVIDIA Tesla C1060
  Programming on Host: Matlab using Single / Multi Core(s)
  Programming on Device: CUDA + CUBLAS, Matlab + Jacket
BLAS Performance on GPU - Matrix Vector Multiplication

- Testing matrices are selected from **Matrix Market**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Prob. Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcsstm19</td>
<td>817</td>
</tr>
<tr>
<td>1138_bus</td>
<td>1138</td>
</tr>
<tr>
<td>bcsstm23</td>
<td>3134</td>
</tr>
<tr>
<td>bcsstk16</td>
<td>4884</td>
</tr>
<tr>
<td>s1rmt3m1</td>
<td>5489</td>
</tr>
<tr>
<td>bcsstk18</td>
<td>15439</td>
</tr>
</tbody>
</table>
BLAS Performance on GPU - Matrix Vector Multiplication

**Figure:** Time Cost

**Figure:** Speedup Ratio
BLAS Performance on GPU - Matrix Vector Multiplication

Figure: Relative Error
BLAS Performance on GPU - BLAS 1, BLAS 2 and BLAS 3

Figure: GFLOPS Comparison

- **Comment**: BLAS 3 outperforms BLAS 1 and BLAS 2 significantly.
A synthetic problem setup using Chan’s example [Chan and Wan, 1997]

\[ A = \text{diag}(1, \ldots, n), \; n = 10240. \]
\[ b_i(t) = \sin(t + i\Delta t), \; i = 1, \ldots, n, \]
and \( \Delta t = \frac{2\pi}{100} \).

\[ \text{TOL} = 1.0 \times 10^{-5}. \]

Results:

CG on Host: 31.90 sec on a single core or 17.53 sec on a multi-core.
CG on Device: 5.49 sec.
CG on GPU (ver 0) - A closer look

Total Time: 5.49 sec;

Mat-Vec: 3.98 sec (72.5%);

Dot-Prod: 0.57 sec (10.4%);

SAXPY: 0.16 sec (2.9%);

Comment: Mat-Vec is too expensive!
Projected CG (PrCG) - Algorithm

- By applying Lanczos-Galerkin projection method to CG, we have the PrCG [Chan and Wan, 1997].

\[
i = 0;
\]
\[
r_i^{(q,1)} = b(q) - A x_i^{(q,1)};
\]
\[
\text{for } i = 1 \text{ to } k
\]
\[
\alpha_i^{(q,1)} = \frac{< p_i^{(1)}, r_i^{(q,1)} >}{< p_i^{(1)}, A p_i^{(1)} >};
\]
\[
x_i^{(q,1)} = x_i^{(q,1)} + \alpha_i^{(q,1)} p_i^{(1)};
\]
\[
r_{i+1}^{(q,1)} = r_i^{(q,1)} - \alpha_i^{(q,1)} A p_i^{(1)};
\]
\[
\text{end (for)};
\]
\[
\text{% Restart the system if needed};
\]
Projected CG (PrCG)

Total Time: 1.91 sec;

Mat-Vec: 1.01 sec (52.8%);

Dot-Prod: 0.45 sec (23.6%);

SAXPY: 0.29 sec (15.2%);

Comment: SAXPY and Dot-Prod are becoming relatively expensive.
Augmented Projected CG (APrCG)

Using the orthogonality properties that $\langle p_j^{(1)}, Ap_i^{(1)} \rangle = 0, \quad j \neq i,$ and $\langle p_j^{(1)}, r_i^{(q,1)} \rangle = 0, \quad j < i,$ take the dot product with $p_i^{(1)}$ on both sides of

$$r_i^{(q,1)} = b(q) - \sum_{i=1}^{k} \alpha_i^{(q,1)} Ap_i^{(1)},$$

we can rewrite $\alpha_i$ as,

$$\alpha_i^{(q,1)} = \frac{\langle p_i^{(1)}, b(q) \rangle}{\langle p_i^{(1)}, Ap_i^{(1)} \rangle}.$$
Augmented Projected CG (APrCG) - Algorithm

- Let $P_k = [p_1, \ldots, p_k]$ and $AP_k = [Ap_1, \ldots, Ap_k]$.

\[
i = 0; \\
r_i^{(q,1)} = b^{(q)} - A x_i^{(q,1)}; \\
\alpha = \langle P_k, r_0^{(q,1)} \rangle / \text{diagVec}(\langle P_k, AP_k \rangle); \\
\Lambda = \text{diag}(\alpha); \\
x^{(q,1)} = x_0^{(q,1)} + \text{sum}(P_k \cdot \Lambda); \\
r^{(q,1)} = r_0^{(q,1)} - \text{sum}(AP_k \cdot \Lambda);\]
Augmented Projected CG (APrCG)

**Figure:** PrCG

**Figure:** APrCG
Figure: Slice Show of A 3D Shepp-Logan Phantom
### 3D Shepp-Logan Phantom - Results

<table>
<thead>
<tr>
<th>Slice</th>
<th>CG</th>
<th>PrCG</th>
<th>APrCG</th>
<th>ImpRate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>SNR</td>
<td>Cost</td>
<td>SNR</td>
</tr>
<tr>
<td>65</td>
<td>17.41</td>
<td>13.67</td>
<td>2.16</td>
<td>13.67</td>
</tr>
<tr>
<td>66</td>
<td>16.34</td>
<td>13.80</td>
<td>2.74</td>
<td>13.83</td>
</tr>
<tr>
<td>67</td>
<td>14.35</td>
<td>13.91</td>
<td>3.11</td>
<td>13.95</td>
</tr>
<tr>
<td>68</td>
<td>13.03</td>
<td>13.80</td>
<td>3.53</td>
<td>13.83</td>
</tr>
</tbody>
</table>

**Table:** Reconstruction of Four Consecutive Slices from 65 to 68. The 64\textsuperscript{th} slice is selected as seed.
Figure: Reconstruction Results on Host and Device, slice index = 66.
Outline

1 Introduction: Inverse Problem and Regularization

2 Topic 1: Multisplitting for Regularized Least Squares
   Regularization Parallelization
   Multiple Right Hand Side Problem

3 Topic 2: Total-Variation Regularization
   Numerical Methods for Total Variation
   Optimal Parameter Selection, UPRE Approach

4 Topic 3: Projected Krylov Subspace Solvers on GPU
   Fine-Grained Parallelism Model for Krylov Subspace Solvers on GPU
   Optimizations of Krylov Subspace Algorithms on GPU
   Numerical Results

5 References