Numerical Matrix Analysis

Notes #24 — Iterative Methods: Overview

Peter Blomgren
⟨blomgren@sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Spring 2024
(Revised: April 22, 2024)

Outline

1. Iterative Methods
   - Overview

Iterative Methods: A Birds-eye View

The size and complexity of linear and non-linear systems that arise from modern applications (especially 3+1-Dimensional models) make direct (LU/QR/SVD) methods intractable in many settings.

Instead of using $\mathcal{O}(m^3)$ operations to find a solution using direct methods, in many cases it is possible to find a **good approximation** (maybe even indistinguishable from the "correct" solution in a floating-point environment) a lot faster using iterative methods.

Yousef Saad, *"Iterative Methods for Sparse Linear Systems,"* 2nd edition, Society for Industrial and Applied Mathematics, 2003, ISBN: 0-89871-534-2, MSRP $89.00.

R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst, *"Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods,"* 2nd Edition, Society for Industrial and Applied Mathematics, 1994, URL `http://www.netlib.org/linalg/html_templates/Templates.html`

The Boundary of "Tractable" Is Moving (Direct Methods)

What does it mean for m to be "very large?" — where m defines an $(m \times m)$ dense matrix.

Year	m	Defining Force
1950	20	Wilkinson
1965	200	Forsythe and Moler [Book, 1967]
1980	2,000	LINPACK
1995	20,000	LAPACK
2010	???,000	??? parallel computing, MPI/CUDA ???
2025	?,???,000	??? parallel computing, MPI/CUDA/FPGA ???

Over the time-span (1950 → 1995) m increased by a factor of 10^3. The speedup in processing speed (computer hardware) over the same period, was about $\sim 10^9$.

Since our deterministic methods scale as $\mathcal{O}(m^3)$ this makes sense, $(10^3)^3 = 10^9$.

If matrix problems scaled as $\mathcal{O}(m^2)$, we would have $m_{1995} \sim 3,000,000$. With **iterative methods** this is sometimes achievable.

The Curse of Higher Dimensions (Direct Methods)

Say you are simulating a 3D-space+Time problem (effectively a 4D-computation)... and you need to refine your computational grid to achieve higher "resolution" in your solution.

Why? — There are many reasons for wanting to do this, *e.g.*

- You may be interested in small-scale behaviour
- You need to numerically demonstrate convergence
- You need to demonstrate numerical stability

The reasonable thing to do is to cut the grid-spacing in half, $\delta x_{1,2,3} \to \frac{1}{2}\delta x_{1,2,3}$, thus doubling the number of points in each spatial direction)

Depending on the computational scheme you are using you also have to refine in the temporal direction $\delta t \to \frac{1}{2^k}\delta t$, $k \geq 1$.

The Curse of Higher Dimensions (Direct Methods)

In the best case setting $k = 1$. So that

$$\begin{array}{rcl|rcl}
\delta x_{1,2,3} & \rightarrow & \frac{1}{2}\delta x_{1,2,3} & n(x_{1,2,3}) & \rightarrow & 2n(x_{1,2,3}) \\
\delta t & \rightarrow & \frac{1}{2}\delta t & n(t) & \rightarrow & 2n(t) \\
\hline
 & & & m & \rightarrow & 2^4 m
\end{array}$$

For an $\mathcal{O}(m^3)$ algorithm, the computational effort just went up by a factor of $(2^4)^3 = 2^{12} = 4,096$.

Sometimes it is possible to "decouple" the time and space growth in 3D+Time algorithms; which "limits" the factor to "only" $2(2^3)^3 = 2^{10} = 1,024$.

That's still 20 years of computational improvements according to *Moore's "Law."*

Iterative Methods: Stationary Methods 1 of 2

Jacobi Iteration, for $A\vec{x} = \vec{b}$ "Classic" — Do not use!

The Jacobi method is based on solving for every variable locally with respect to the other variables; one iteration of the method corresponds to solving for every variable once. The resulting method is easy to understand and implement, but convergence is slow

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[\vec{b}_i - \sum_{j \neq i} a_{ij} x_j^{(k-1)} \right], \quad i = 1, \dots, n$$

Gauss-Seidel Iteration, for $A\vec{x} = \vec{b}$ "Classic" — Do not use!

The Gauss-Seidel method is like the Jacobi method, except that it uses updated values as soon as they are available. In general, if the Jacobi method converges, the Gauss-Seidel method will converge faster than the Jacobi method, though still relatively slowly.

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[\vec{b}_i - \sum_{j<i} a_{ij} x_j^{(k)} - \sum_{j>i} a_{ij} x_j^{(k-1)} \right], \quad i = 1, \dots, n$$

Iterative Methods: Stationary Methods 2 of 2

SOR, for $A\vec{x} = \vec{b}$ "Classic" — Do not use!

Successive Over-relaxation (SOR) can be derived from the Gauss-Seidel method by introducing an extrapolation parameter. For the optimal choice of ω, SOR may converge faster than Gauss-Seidel by an order of magnitude

$$x_i^{(k)} = \omega \widehat{x}_i^{(k)} + (1-\omega) x_i^{(k-1)}, \quad \text{where} \quad \widehat{x}_i^{(k)} = \text{the GS iterate}$$

SSOR, for $A\vec{x} = \vec{b}$ "Classic" — Do not use!

Symmetric Successive Over-relaxation (SSOR) has no advantage over SOR as a stand-alone iterative method; however, it is useful as a preconditioner for nonstationary methods. — SSOR is a forward SOR sweep followed by a backward SOR sweep in which the unknowns are updated in the reverse order.

Krylov Subspaces, Arnoldi and Lanczos

Given a vector $\vec{x}$ and a matrix A, the associated **Krylov vector sequence** is $\{\vec{x},\, A\vec{x},\, \dots,\, A^{k-1}\vec{x},\, \dots\}$, and the corresponding **Krylov subspaces** $K(A, \vec{x}; k) = \mathrm{span}\{\vec{x},\, A\vec{x},\, \dots,\, A^{k-1}\vec{x}\}$.

Most of the iterative algorithms described in the following slides for solution of the linear system $A\vec{x} = \vec{b}$ are derived from analysis (minimization of residuals) over Krylov subspaces, specifically

Symmetry	**Linear System** $\mathbf{A\tilde{x} = \tilde{b}}$	**Eigenvalue Problem** $\mathbf{A\tilde{x} = \lambda\tilde{x}}$
$\mathbf{A = A^*}$	`CG`	`Lanczos`

$\mathbf{A \neq A^*}$	`GMRES`
	`CGNE` / `CGNR`
	`BiCG`, etc...
		`Arnoldi`

San Diego State University

The Arnoldi Iteration

Arnoldi is to Householder-based Hessenberg transformations, as Gram-Schmidt is to Householder-based QR-factorization:

Problem	$\mathbf{A = QR}$	$\mathbf{A = QHQ^*}$
Orthogonal Transformation $\rightsquigarrow$ **Structure**	Householder $\mathbf{Q}^* A = R$	Householder $\mathbf{Q}^* A\mathbf{Q} = H$
"Structure" Transformation $\rightsquigarrow$ **Orthogonalization**	Gram-Schmidt $A\mathbf{R}^{-1} = Q$	Arnoldi $AQ_n = Q_{n+1}\mathbf{H_n}$

By "enforcing" the desired structure on $\mathbf{H_n}$, the "by-product" is the orthogonal matrix Q.

We will discuss the Arnoldi iteration in more detail in the next lecture.

Lanczos vs. Arnoldi

When $A = A^*$, we can find an orthonormal similarity transform so that $A = QTQ^*$, where T is tri-diagonal. This special case makes the Lanczos iteration much faster (**3-term recurrence**) than the Arnoldi iteration ("infinite" ***n*-recurrence**):

```
$\vec{b} = \vec{b}_0$, $\vec{q}_1 = \vec{b}/\|\vec{b}\|$
for n = 1,...
   $\vec{v} = A\vec{q}_n$
   for j = 1,...,n
      $\mathbf{h_{jn}} = \mathbf{\tilde{q}_j}^*\mathbf{\tilde{v}}$
      $\mathbf{\tilde{v}} = \mathbf{\tilde{v}} - \mathbf{h_{jn}\tilde{q}_j}$
   endfor(j)
   $h_{n+1,n} = \|v\|$
   $\vec{q}_{n+1} = \vec{v}/h_{n+1,n}$
endfor(n)
```

Arnoldi iteration

```
$\vec{b} = \vec{b}_0$, $\vec{q}_1 = \vec{b}/\|\vec{b}\|$,
$\beta_0 = 0$, $q_0 = 0$
for n = 1,...
   $\vec{v} = A\vec{q}_n$
   $\alpha_n = \vec{q}_n^*\vec{v}$
   $\mathbf{\tilde{v}} = \mathbf{\tilde{v}} - \beta_{\mathbf{n-1}}\mathbf{\tilde{q}_{n-1}} - \alpha_{\mathbf{n}}\mathbf{\tilde{q}_n}$
   $\beta_n = \|v\|$
   $\vec{q}_{n+1} = \vec{v}/\beta_n$
endfor(n)
$T = \mathrm{diag}(\vec{\alpha}) + \mathrm{diag}(\vec{\beta}, \pm 1)$
```

Lanczos iteration

Iterative Methods: Non-Stationary Methods 1 of 6

Conjugate Gradient, “CG”

The conjugate gradient method derives its name from the fact that it generates a sequence of conjugate (or orthogonal) vectors.

These vectors are the residuals of the iterates. They are also the gradients of a quadratic functional, the minimization of which is equivalent to solving the linear system.

CG is an extremely effective method when the coefficient matrix is **symmetric positive definite**, since storage for only a limited number of vectors is required. (See slide 25 for implementation details.)

Minimum Residual, “MINRES”

A computational alternative for CG for coefficient matrices that are **symmetric but possibly indefinite.**

SAN DIEGO STATE UNIVERSITY

Iterative Methods: Non-Stationary Methods 2 of 6

Conjugate Gradient on the Normal Equations, "CGNE" / "CGNR"

These methods are based on the application of the CG method to one of two forms of the normal equations for $A\vec{x} = \vec{b}$.

CGNE solves the system for $(AA^*)\vec{y} = \vec{b}$, and then computes the solution $\vec{x} = A^*\vec{y}$.

CGNR solves $(A^*A)\vec{x} = A^*\vec{b}$ for the solution vector $\vec{x}$.

When the coefficient matrix A is non-symmetric and nonsingular, the normal equations matrices AA^* and A^*A will be symmetric and positive definite, and hence CG can be applied. The convergence may be slow, since the spectrum of the normal equations matrices will be less favorable than the spectrum of A.

Chebyshev Iteration

The Chebyshev Iteration recursively determines polynomials with coefficients chosen to minimize the norm of the residual in a min-max sense. The coefficient matrix must be positive definite and knowledge of the extremal eigenvalues is required. This method has the advantage of requiring no inner products.

Iterative Methods: Non-Stationary Methods 3 of 6

Generalized Minimal Residual, “GMRES”

The Generalized Minimal Residual method computes a sequence of orthogonal vectors (like MINRES), and combines these through a least-squares solve and update.

However, unlike MINRES (and CG) it requires storing the whole sequence, so that a large amount of storage is needed.

For this reason, restarted versions of this method are used. In restarted versions, computation and storage costs are limited by specifying a fixed number of vectors to be generated. This method is useful for **general non-symmetric matrices.** (See slide 24 for implementation details.)

Iterative Methods: Non-Stationary Methods 4 of 6

BiConjugate Gradient, “BiCG”

The Biconjugate Gradient method generates two CG-like sequences of vectors, one based on a system with the original coefficient matrix A, and one on A^*.

Instead of orthogonalizing each sequence, they are made mutually orthogonal, or “bi-orthogonal”. This method, like CG, uses limited storage. It is useful when the matrix is **non-symmetric and nonsingular**.

However, convergence may be irregular, and there is a possibility that the method will break down. BiCG requires a multiplication with the coefficient matrix and with its *transpose* at each iteration. (See slide 26 for implementation details.)

San Diego State University

Iterative Methods: Non-Stationary Methods

Conjugate Gradient Squared, “CGS”

The Conjugate Gradient Squared method is a variant of BiCG that applies the updating operations for the A-sequence and the A^*-sequences both to the same vectors.

Ideally, this would double the convergence rate, but in practice convergence may be much more irregular than for `BiCG`, which may sometimes lead to unreliable results. A practical advantage is that the method *does not need the multiplications with the transpose* of the coefficient matrix.

Biconjugate Gradient Stabilized, “BiCGSTAB”

The Biconjugate Gradient Stabilized method is a variant of `BiCG`, like `CGS`, but using different updates for the A^*-sequence in order to obtain smoother convergence than `CGS`.

Iterative Methods: Non-Stationary Methods

Quasi-Minimal Residual , "QMR"

The Quasi-Minimal Residual method applies a least-squares solve and update to the BiCG residuals, thereby smoothing out the irregular convergence behavior of BiCG, which may lead to more reliable approximations.

In full glory, it has a **look-ahead** strategy built in that avoids the BiCG breakdown. Even without look ahead, QMR largely avoids the breakdown that can occur in BiCG.

On the other hand, it does not effect a true minimization of either the error or the residual, and while it converges smoothly, it often does not improve on the BiCG in terms of the number of iteration steps.

Transpose-Free Quasi-Minimal Residual, "TFQMR"

A variant of the QMR algorithm which avoids multiplication by A^*.

Other Approaches 1 of 5

Preconditioning

A preconditioner is any form of explicit or implicit modification of an original linear system that makes it easier to solve by a given iterative method.

Finding a good preconditioner for a given linear system is a combination of art, science, and white magic ☺.

One step of the Jacobi, Gauss-Seidel, SOR, and SSOR iterations can be viewed as the application of a (rudimentary) preconditioner.

Successful preconditioning techniques are often based on incomplete factorizations.

Other Approaches 2 of 5

Multigrid methods

Multigrid (MG) methods apply to linear systems arising from discretizations of Partial Differential Equations (PDEs). MG techniques use discretizations with different mesh sizes, to achieve fast convergence.

Roughly, a solution is computed on a coarse mesh, that solution (interpolated to a finer mesh) is used as an initial guess for the fine-mesh solution, etc...

Warning: There are some serious dragons hiding in the details!

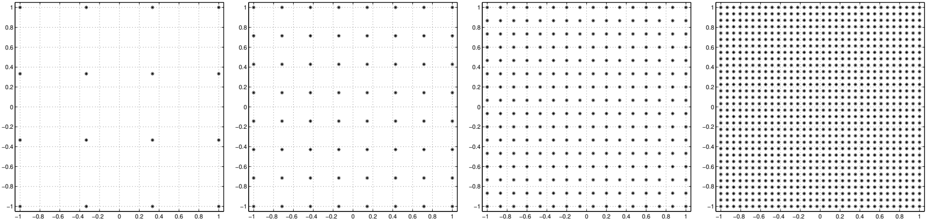
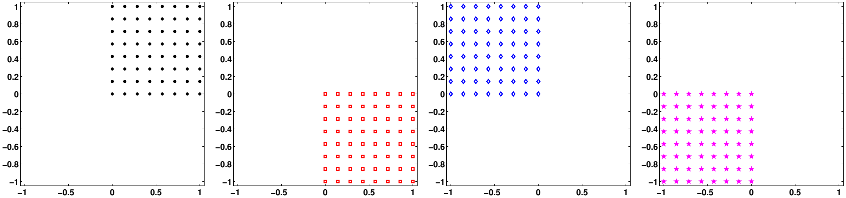
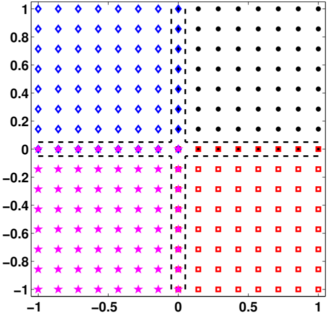


Figure: A successive refinement of the grid.

Domain Decomposition methods





Domain decomposition methods revolve around the divide-and-conquer principle.

The (discretized) PDE is solved over sub-domains, and then the global solution is patched together — very special attention must be paid at the numerical boundaries between the sub-domains.

Other Approaches

Fast Multipole Method, FMM

- Introduced by Rokhlin & Greengard in 1987. *"One of the 10 most significant advances in computing in the 20th century."*
- An algorithm for achieving fast matrix-vector products for particular dense matrices.
- Developed based on ideas similar to the Fast Fourier Transform (FFT), and in some sense multi-grid methods...
  - FFT — matrix entries are uniformly sampled complex exponentials.
  - FMM — matrix entries are derived from particular functions, satisfying known translation theorems.
- Standard matrix-vector multiply $\sim \mathcal{O}(m^2)$.
- FMM matrix-vector multiply $\sim \mathcal{O}(m \log(m))$.

Other Approaches

Kaczmarz's Algorithm, and beyond

- [Kaczmarz (1937)] — Projection based algorithm; "solve" using one column of the matrix at a time — Suitable for VERY LARGE matrices
- [Randomized Kaczmarz: Strohmer–Vershynin (2009)] ≈ stochastic gradient descent (SDG)... When $\lambda(A) > 0$
- [Gower–Richtarik (2015)] — A randomized iterative method for solving a consistent system of linear equations; K(1937) is a special case. Other special cases include randomized coordinate descent, randomized Gaussian descent and randomized Newton method. Block versions and versions with importance sampling of all these methods also arise as special cases.

What do I know about these algorithms??? I know they exist. :-)

Algorithm Reference: Arnoldi Iteration

Algorithm (Arnoldi Iteration)

```
$\vec{b} = \vec{b}_0 = \mathrm{arbitrary}, \quad \vec{q}_1 = \vec{b}/\|\vec{b}\|$
for n = 1,...
    $\vec{v} = A\vec{q}_n$
    for j = 1,...,n
        $h_{jn} = \vec{q}_j^{\,*}\vec{v}$
        $\vec{v} = \vec{v} - h_{jn}\vec{q}_j$
    endfor(j)
    $h_{n+1,n} = \|v\|$
    $\vec{q}_{n+1} = \vec{v}/h_{n+1,n}$
endfor(n)
```

Algorithm Reference: GMRES

Algorithm (GMRES)

```
$\vec{q}_1 = \vec{b}/\|\vec{b}\|$
for n = 1:...
    → Step $n$ of Arnoldi Iteration ←
    $\vec{y}_n = \arg\min_{\vec{y}} \left\| H_n \vec{y} - \|\vec{b}\| \vec{e}_1 \right\|$
    $\vec{x}_n = Q_n \vec{y}_n$
endfor
```

Algorithm Reference: CG

Algorithm (Conjugate Gradient CG Iteration)

```
$\vec{x}_0 = 0,\ \vec{p}_0 = \vec{r}_0 = \vec{b}$
for n = 1:...
    $\alpha_n = \frac{\|\vec{r}_{n-1}\|^2}{\vec{p}^*_{n-1} A \vec{p}_{n-1}}$      Step length
    $\vec{x}_n = \vec{x}_{n-1} + \alpha_n \vec{p}_{n-1}$      Approximate solution
    $\vec{r}_n = \vec{r}_{n-1} - \alpha_n A \vec{p}_{n-1}$      Updated residual
    $\beta_n = \frac{\|\vec{r}_n\|^2}{\|\vec{r}_{n-1}\|^2}$      Improvement this step
    $\vec{p}_n = \vec{r}_n + \beta_n \vec{p}_{n-1}$      New search direction
endfor
```

Algorithm Reference: BiCG

Algorithm (Bi-Conjugate Gradient BiCG Iteration)

```
$\vec{x}_0 = 0,\ \vec{p}_0 = \vec{r}_0 = \vec{b},\ \vec{q}_0 = \vec{s}_0 = \text{arbitrary}$
for n = 1:...
    $\alpha_n = \frac{\vec{s}^*_{n-1}\vec{r}_{n-1}}{\vec{q}^*_{n-1}A\vec{p}_{n-1}}$                  Step length
    $\vec{x}_n = \vec{x}_{n-1} + \alpha_n\vec{p}_{n-1}$                  Approximate solution
    $\vec{r}_n = \vec{r}_{n-1} - \alpha_n A\vec{p}_{n-1}$                Updated residual, $\vec{r}$
    $\vec{s}_n = \vec{s}_{n-1} - \overline{\alpha_n}A^*\vec{q}_{n-1}$    Updated residual, $\vec{s}$
    $\beta_n = \frac{\vec{s}^*_n\vec{r}_n}{\vec{s}^*_{n-1}\vec{r}_{n-1}}$                    Improvement this step
    $\vec{p}_n = \vec{r}_n + \beta_n\vec{p}_{n-1}$                       New search direction, $\vec{p}$
    $\vec{q}_n = \vec{s}_n + \overline{\beta_n}\vec{q}_{n-1}$            New search direction, $\vec{q}$
endfor
```

Acknowledgment

Slides 7–8, and 12–17 are blatantly “borrowed” from the html-version of *“Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.”*