Numerical Solution of Linear Systems

Chen Greif

Department of Computer Science
The University of British Columbia
Vancouver B.C.

Tel Aviv University
December 17, 2008

Outline

A Dense Matrix

Top Ten Algorithms in Science (Dongarra and Sullivan, 2000)

1. Metropolis Algorithm (Monte Carlo method)
2. Simplex Method for Linear Programming
3. Krylov Subspace Iteration Methods
4. The Decompositional Approach to Matrix Computations
5. The Fortran Optimizing Compiler
6. QR Algorithm for Computing Eigenvalues
7. Quicksort Algorithm for Sorting
8. Fast Fourier Transform
9. Integer Relation Detection Algorithm
10. Fast Multipole Method

- Red: Algorithms within the exclusive domain of NLA research.
- Blue: Algorithms strongly (though not exclusively) connected to NLA research.

Outline

Two types of methods

Numerical methods for solving linear systems of equations can generally be divided into two classes:

- **Direct methods**. In the absence of roundoff error such methods would yield the exact solution within a finite number of steps.
- **Iterative methods**. These are methods that are useful for problems involving special, very large matrices.

Gaussian Elimination and LU Decomposition

Assumptions:

- The given matrix A is real, $n \times n$ and nonsingular.
- The problem $A\mathbf{x} = \mathbf{b}$ therefore has a unique solution $\mathbf{x}$ for any given vector $\mathbf{b}$ in $\mathcal{R}^n$.

The basic direct method for solving linear systems of equations is **Gaussian elimination**. The bulk of the algorithm involves only the matrix A and amounts to its decomposition into a product of two matrices that have a simpler form. This is called an **LU decomposition**.

How to

- solve linear equations when A is in upper triangular form. The algorithm is called *backward substitution*.
- transform a general system of linear equations into an upper triangular form, where backward substitution can be applied. The algorithm is called *Gaussian elimination*.

Backward Substitution

Start easy:

- If A is diagonal:

$$A = \begin{pmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{pmatrix},$$

then the linear equations are uncoupled and the solution is obviously

$$x_i = \frac{b_i}{a_{ii}}, \quad i = 1, 2, \ldots, n.$$

Triangular Systems

An **upper triangular** matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \ddots & \vdots \\ & & \ddots & \vdots \\ & & & a_{nn} \end{pmatrix},$$

where all elements below the main diagonal are zero:
$a_{ij} = 0, \forall i > j$.
Solve *backwards*: The last row reads $a_{nn}x_n = b_n$, so $x_n = \frac{b_n}{a_{nn}}$.
Next, now that we know x_n, the row before last can be written as $a_{n-1,n-1}x_{n-1} = b_{n-1} - a_{n-1,n}x_n$, so $x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}$. Next the previous row can be dealt with, etc. We obtain the **backward substitution** algorithm.

Computational Cost (Naive but Useful)

What is the cost of this algorithm? In a simplistic way we just count each floating point operation (such as $+$ and $*$) as a *flop*. The number of flops required here is

$$1+\sum_{k=1}^{n-1}((n-k-1)+(n-k)+2) \approx 2\sum_{k=1}^{n-1}(n-k) = 2\frac{(n-1)n}{2} \approx n^2.$$

Simplistic but not ridiculously so: doesn't take into account data movement between elements of the computer's memory hierarchy. In fact, concerns of data locality can be crucial to the execution of an algorithm. The situation is even more complex on multiprocessor machines. *But still: gives an idea...*

LU Decomposition

The Gaussian elimination procedure *decomposes* A into a product of a unit lower triangular matrix L and an upper triangular matrix U. This is the famous **LU decomposition**. Together with the ensuing backward substitution the entire solution algorithm for $A\mathbf{x} = \mathbf{b}$ can therefore be described in three steps:

1. *Decomposition*:
$$A = LU$$
2. *Forward substitution*: solve
$$L\mathbf{y} = \mathbf{b}.$$
3. *Backward substitution*: solve
$$U\mathbf{x} = \mathbf{y}.$$

Pivoting

In a nutshell, perform permutations to increase numerical stability.
Trivial but telling examples:
For

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

or

$$A_\varepsilon = \begin{pmatrix} \varepsilon & 1 \\ 1 & 0 \end{pmatrix}$$

G.E. will fail (for A) or perform poorly (for A_ε).

Nothing wrong with the problem, it's the algorithm to blame!

- Partial pivoting (not always stable but standard)
- Complete pivoting (stable but too expensive)
- Rook pivoting (I like it!)

Special Matrices

- **Symmetric Positive Definite.** A matrix A is symmetric positive definite (SPD) if $A = A^T$ and $\mathbf{x}^T A\mathbf{x} > 0$ for any nonzero vector $\mathbf{x} \neq 0$. (All SPD matrices necessarily have positive eigenvalues.)
  In the context of linear systems – **Cholesky Decomposition**:

$$A = FF^T.$$

  - No pivoting required
  - Half the storage and work. (But still $O(n^2)$ and $O(n^3)$ respectively.)

Special Matrices (Cont.)

- **Narrow Banded.**

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1q} & & & & \\ \vdots & \ddots & \ddots & \ddots & & & \\ a_{p1} & \ddots & \ddots & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots & \ddots & a_{n-q+1,n} \\ & & & \ddots & \ddots & \ddots & \vdots \\ & & & & a_{n,n-p+1} & \cdots & a_{nn} \end{pmatrix}.$$

  - Significant savings, if bandwidth is small: $O(n)$ work and storage.

A Useful Numerical Tip

Never invert a matrix explicitly unless your life depends on it.
In MATLAB, choose `backslash` over `inv`.
Reasons:

- More accurate and efficient
- For banded matrices, great saving in storage

LU vs. Gaussian Elimination (why store L?)

If you did all the work, might as well record it!

One good reason: solving linear systems with multiple right hand sides.

Permutations and Reordering Strategies

Riddle: Which matrix is better to work with?

$$A = \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & 0 & 0 & 0 \\ \times & 0 & \times & 0 & 0 \\ \times & 0 & 0 & \times & 0 \\ \times & 0 & 0 & 0 & \times \end{pmatrix}.$$

$$B = \begin{pmatrix} \times & 0 & 0 & 0 & \times \\ 0 & \times & 0 & 0 & \times \\ 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}.$$

B is a matrix obtained by swapping the first and the last row and column of A.

Permutation Matrices

$$(PAP^T)(P\mathbf{x}) = P\mathbf{b}.$$

Look at $P\mathbf{x}$ rather than $\mathbf{x}$, as per the performed permutation.

- If A is symmetric then so is PAP^T. We can define the latter matrix as B and rewrite the linear system as

$$B\mathbf{y} = \mathbf{c},$$

  where $\mathbf{y} = P\mathbf{x}$ and $\mathbf{c} = P\mathbf{b}$.
- In the example $B = PAP^T$ where P is a permutation matrix associated with the vector $\mathbf{p} = (n, 2, 3, 4, \ldots, n-2, n-1, 1)^T$.

Graphs

At least two possible ways of aiming to reduce the storage and computational work:

- Reduce the bandwidth of the matrix.
- Reduce the expected fill-in in the decomposition stage.

One of the most commonly used tools for doing it is **graph theory**.

- In the process of Gaussian elimination, each stage can be described as follows in terms of the matrix graph: upon zeroing out the (i, j) entry of the matrix (that is, with entry (j, j) being the current pivot in question), all the vertices that are the 'neighbors' of vertex j will cause the creation of an edge connecting them to vertex i, if such an edge does not already exist.
- This shows why working with B is preferable over working with A in the example: for instance, when attempting to zero out entry $(5, 1)$ using entry $(1, 1)$ as pivot, in the graph of $A^{(1)}$ all vertices j connected to vertex 1 will generate an edge $(5, j)$. For A, this means that new edges $(2, 5)$, $(3, 5)$, $(4, 5)$ are now generated, whereas for B no new edge is generated because there are no edges connected to vertex 1 other than vertex 5 itself.

Edges and Vertices

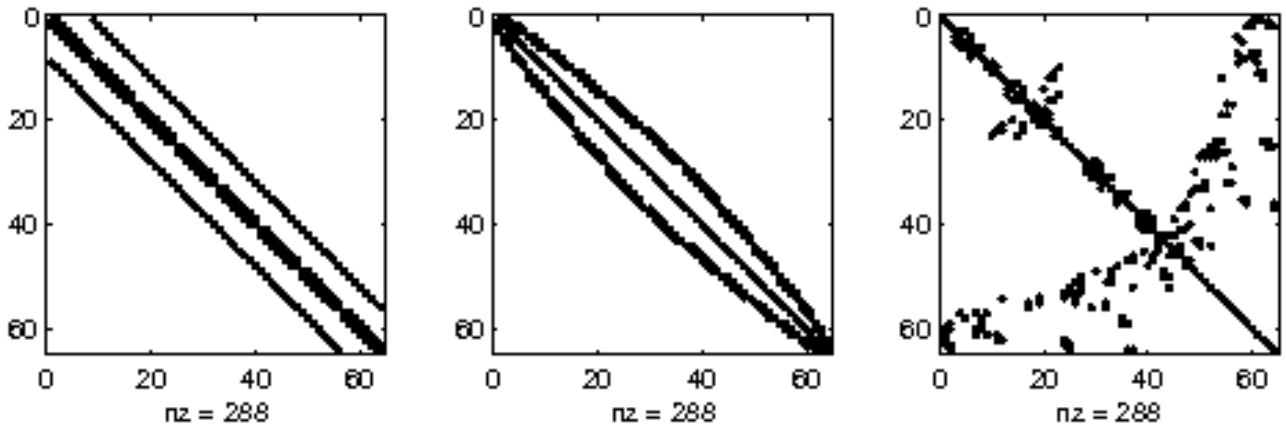
- The *degree* of a vertex is the number of edges emanating from the vertex. It is in our best interest to postpone dealing with vertices of a high degree as much as possible.
- For the matrix A in the example all the vertices except vertex 1 have degree 1, but vertex 1 has degree 4 and we start off the Gaussian elimination by eliminating it; this results in disastrous fill-in.
- On the other hand for the B matrix we have that all vertices except vertex 5 have degree 1, and vertex 5 is the last vertex we deal with. Until we hit vertex 5 there is no fill, because the latter is the only vertex that is connected to the other vertices. When we deal with vertex 5 we identify vertices that should hypothetically be generated, but they are already in existence to begin with, so we end up with no fill whatsoever.

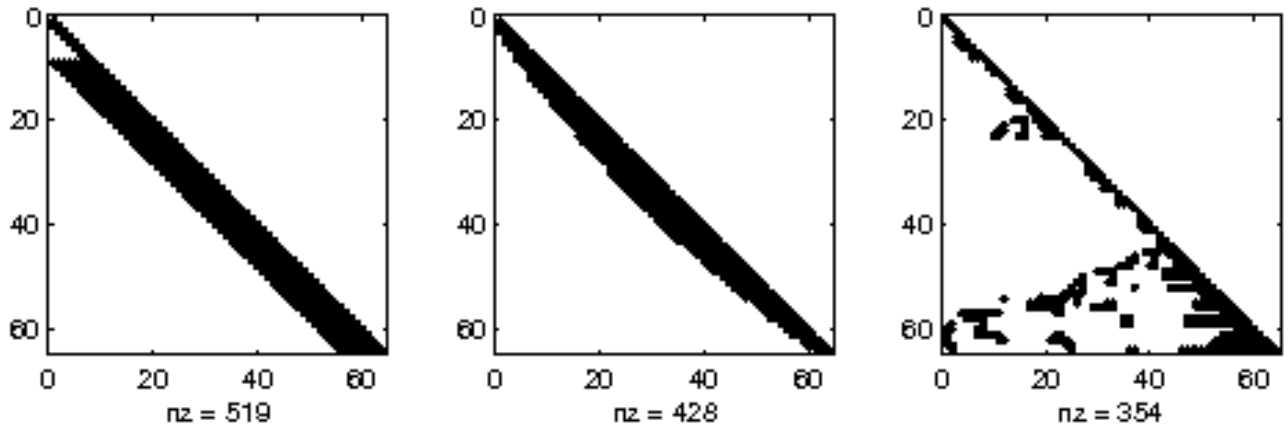
Optimality criteria

- How to assure that the amount of work for determining the ordering does not dominate the computation. As you may already sense, determining an 'optimal' graph may be quite a costly adventure.
- How to deal with 'tie breaking' situations. For example, if we have an algorithm based on the degrees of vertices, what if two or more of them have the same degree: which one should be labeled first?

Ordering strategies

- *Reverse Cuthill McKee* (RCM): aims at minimizing bandwidth.
- *minimum degree* (MMD) or *approximate minimum degree* (AMD): aims at minimizing the expected fill-in.


0
20
40
60
0
20
40
60
nz = 288
0
20
40
60
0
20
40
60
nz = 288
0
20
40
60
0
20
40
60
nz = 288


0
20
40
60
0
20
40
60
nz = 519
0
20
40
60
0
20
40
60
nz = 428
0
20
40
60
0
20
40
60
nz = 354

Outline

Suppose that, using some algorithm, we have computed an approximate solution $\hat{\mathbf{x}}$. We would like to be able to evaluate the absolute error $\|\mathbf{x} - \hat{\mathbf{x}}\|$, or the relative error

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|}.$$

- We do not know the error; seek an upper bound, and rely on computable quantities, such as the *residual*

$$\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}}.$$

- A stable Gaussian elimination variant will deliver a residual with a small norm. The question is, what can we conclude from this about the error in $\mathbf{x}$?

How does the residual $\mathbf{r}$ relate to the error in $\hat{\mathbf{x}}$ in general?

$$\mathbf{r} \quad = \quad \mathbf{b} - A\hat{\mathbf{x}} = A\mathbf{x} - A\hat{\mathbf{x}} = A(\mathbf{x} - \hat{\mathbf{x}}).$$

So

$$\mathbf{x} - \hat{\mathbf{x}} = A^{-1}\mathbf{r}.$$

Then

$$\|\mathbf{x} - \hat{\mathbf{x}}\| = \|A^{-1}\mathbf{r}\| \leq \|A^{-1}\|\|\mathbf{r}\|.$$

This gives a bound on the absolute error in $\hat{\mathbf{x}}$ in terms of $\|A^{-1}\|$.
But usually the relative error is more meaningful. Since
$\|\mathbf{b}\| \leq \|A\|\|\mathbf{x}\|$ implies $\frac{1}{\|\mathbf{x}\|} \leq \frac{\|A\|}{\|\mathbf{b}\|}$, we have

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \|A^{-1}\|\|\mathbf{r}\| \ \frac{\|A\|}{\|\mathbf{b}\|}.$$

Condition Number

We therefore define the **condition number** of the matrix A as

$$\kappa(A) = \|A\|\|A^{-1}\|$$

and write the bound obtained on the relative error as

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}.$$

In words, the relative error in the solution is bounded by the condition number of the matrix A times the relative error in the residual.

Properties (and myths)

- Range of values:

$$1 = \|I\| = \|A^{-1}A\| \leq \kappa(A),$$

  (i.e. a matrix is ideally conditioned if its condition number equals 1), and $\kappa(A) = \infty$ for a singular matrix.
- *Orthogonal matrices* are perfectly conditioned.
- If A is SPD, $\kappa_2(A) = \frac{\lambda_1}{\lambda_n}$.
- Condition number is defined for *any* (even non-square) matrices by the singular values of the matrix.
- When something goes wrong with the numerical solution - blame the condition number! (and hope for the best)
- One of the most important areas of research: *preconditioning.* (To be discussed later.)
- **What's a well-conditioned matrix and what's an ill-conditioned matrix?**

Outline

Trouble in Paradise

The Gaussian elimination algorithm and its variations such as the LU decomposition, the Cholesky method, adaptation to banded systems, etc., is the approach of choice for many problems. There are situations, however, which require a different treatment.

Drawbacks of Direct Solution Methods

- The Gaussian elimination (or LU decomposition) process may introduce **fill-in**, i.e. L and U may have nonzero elements in locations where the original matrix A has zeros. If the amount of fill-in is significant then applying the direct method may become costly. This in fact occurs often, in particular when the matrix is banded and is sparse within the band.
- Sometimes we do not really need to solve the system exactly. (e.g. nonlinear problems.) Direct methods cannot accomplish this because by definition, to obtain a solution the process must be completed; there is no notion of an early termination or an inexact (yet acceptable) solution.

Drawbacks of Direct Solution Methods (Cont.)

- Sometimes we have a pretty good idea of an approximate guess for the solution. For example, in time dependent problems (*warm start* with previous time solution). Direct methods cannot make good use of such information.
- Sometimes only matrix-vector products are given. In other words, the matrix is not available explicitly or is very expensive to compute. For example, in digital signal processing applications it is often the case that only input and output signals are given, without the transformation itself explicitly formulated and available.

Motivation for Iterative Methods

What motivates us to use iterative schemes is the possibility that inverting A may be very difficult, to the extent that it may be worthwhile to invert a much 'easier' matrix several times, rather than inverting A directly only once.

A Common Example: Laplacian (Poisson Equation)

$$A = \begin{pmatrix} J & -I & & & \\ -I & J & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & J & -I \\ & & & -I & J \end{pmatrix},$$

where J is a tridiagonal $N \times N$ matrix

$$J = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}$$

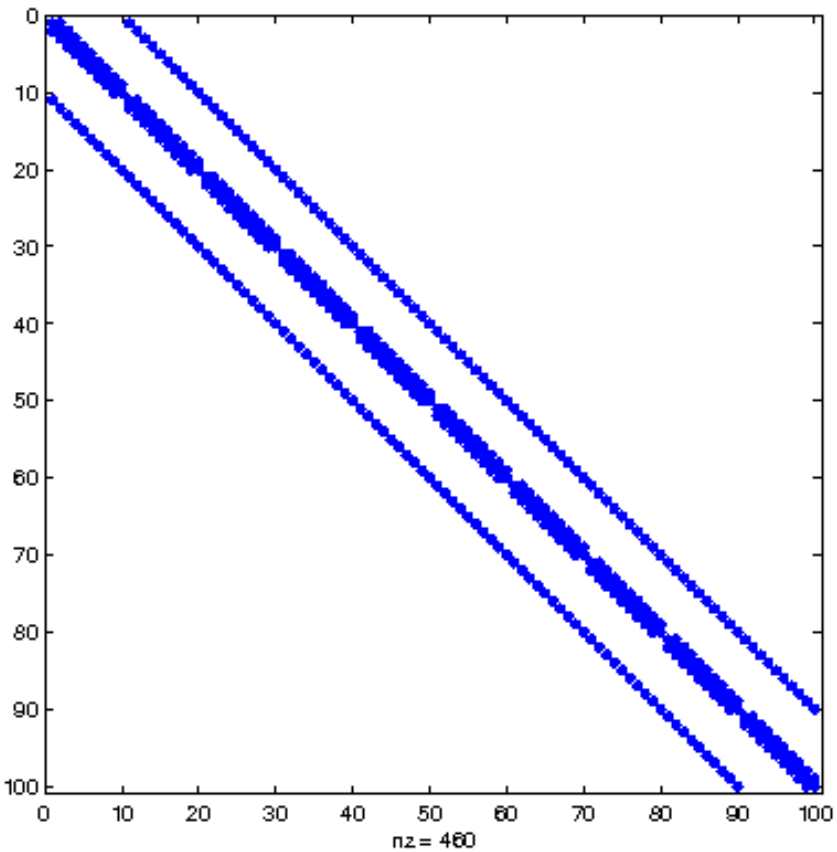
and I denotes the identity matrix of size N.

A Small Example

For instance, if $N = 3$ then

$$A = \left(\begin{array}{ccc|ccc|ccc} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ \hline -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{array}\right).$$

Sparsity Pattern



Stationary Methods

Given $A\mathbf{x} = \mathbf{b}$, we can rewrite as $\mathbf{x} = (I - A)\mathbf{x} + \mathbf{b}$, which yields the iteration

$$\mathbf{x}_{k+1} = (I - A)\mathbf{x}_k + \mathbf{b}.$$

From this we can generalize: for a given *splitting* $A = M - N$, we have $M\mathbf{x} = N\mathbf{x} + \mathbf{b}$, which leads to the fixed point iteration

$$M\mathbf{x}_{k+1} = N\mathbf{x}_k + \mathbf{b}.$$

The Basic Mechanism

Suppose that $A = M - N$ is a splitting, and that $M\mathbf{z} = \mathbf{r}$ is much easier to solve than $A\mathbf{x} = \mathbf{b}$. Given an initial guess $\mathbf{x}_0$,

$$\mathbf{e}_0 = \mathbf{x} - \mathbf{x}_0$$

is the error and

$$A\mathbf{e}_0 = \mathbf{b} - A\mathbf{x}_0 := \mathbf{r}_0.$$

Notice that $\mathbf{r}_0$ is computable whereas $\mathbf{e}_0$ is not, because $\mathbf{x}$ is not available. Since $\mathbf{x} = \mathbf{x}_0 + \mathbf{e}_0 = \mathbf{x}_0 + A^{-1}\mathbf{r}_0$, set

$$M\tilde{\mathbf{e}} = \mathbf{r}_0,$$

and then

$$\mathbf{x}_1 = \mathbf{x}_0 + \tilde{\mathbf{e}}$$

is our new guess. $\mathbf{x}_1$ is hopefully closer to $\mathbf{x}$ than $\mathbf{x}_0$.

The Tough Task of Finding a Good M

The matrix M should satisfy two contradictory requirements:

- It should be close to A in some sense (or rather, M^{-1} should be close to A^{-1}).
- It should be much easier to invert than A.

Jacobi, Gauss-Seidel, and Friends

It all boils down to the choice of M. If $A = D + E + F$ is split into diagonal D, strictly upper triangular part E and strictly lower triangular part F, then:

- Jacobi: $M = D$.
- Gauss-Seidel: $M = D + E$.
- SOR: a parameter dependent 'improvement' of Gauss-Seidel.

Convergence

- It is easy to show that (asymptotic) convergence is governed (barring 'nasty' matrices) by the question what the eigenvalues of $T = M^{-1}N = I - M^{-1}A$ are. They must be smaller than 1 in magnitude, and the smaller they are the faster we converge.
- Jacobi and Gauss-Seidel are **terribly slow** methods. SOR is not so bad, but requires a choice of a parameter.
- But these methods also have merits.
  - Convergence analysis is easy to perform and can give an idea.
  - Jacobi is beautifully parallelizable, which is why it has been taken out of its grave since HPC has become important.
  - Gauss-Seidel has beautiful smoothing properties and is used in Multigrid.

Nonstationary Methods: Conjugate Gradients and Friends

The trouble with stationary schemes is that they do not make use of information that has accumulated throughout the iteration. How about trying to optimize something throughout the iteration? For example,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{r}_k.$$

Adding $\mathbf{b}$ to both sides and subtracting the equations multiplied by A:

$$\mathbf{b} - A\mathbf{x}_{k+1} = \mathbf{b} - A\mathbf{x}_k - \alpha_k A\mathbf{r}_k.$$

It is possible to find α_k that minimizes the residual. Notice:

$$\mathbf{r}_k = p_k(A)\mathbf{r}_0.$$

Modern method work hard at finding 'the best' p_k.

Nonstationary Methods as an Optimization Problem

The methods considered here can all be written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k,$$

where the vector $\mathbf{p}_k$ is the *search direction* and the scalar α_k is the *step size*. The simplest such non-stationary scheme is obtained by setting $\mathbf{p}_k = \mathbf{r}_k$, i.e. $M_k = \alpha_k I$, with I the identity matrix. The resulting method is called **gradient descent**.
The step size α_k may be chosen so as to minimize the ℓ_2 norm of the residual $\mathbf{r}_k$. But there are other options too.

Conjugate Gradients (for SPD matrices)

Our problem $A\mathbf{x} = \mathbf{b}$ is equivalent to the problem of finding a vector $\mathbf{x}$ that minimizes

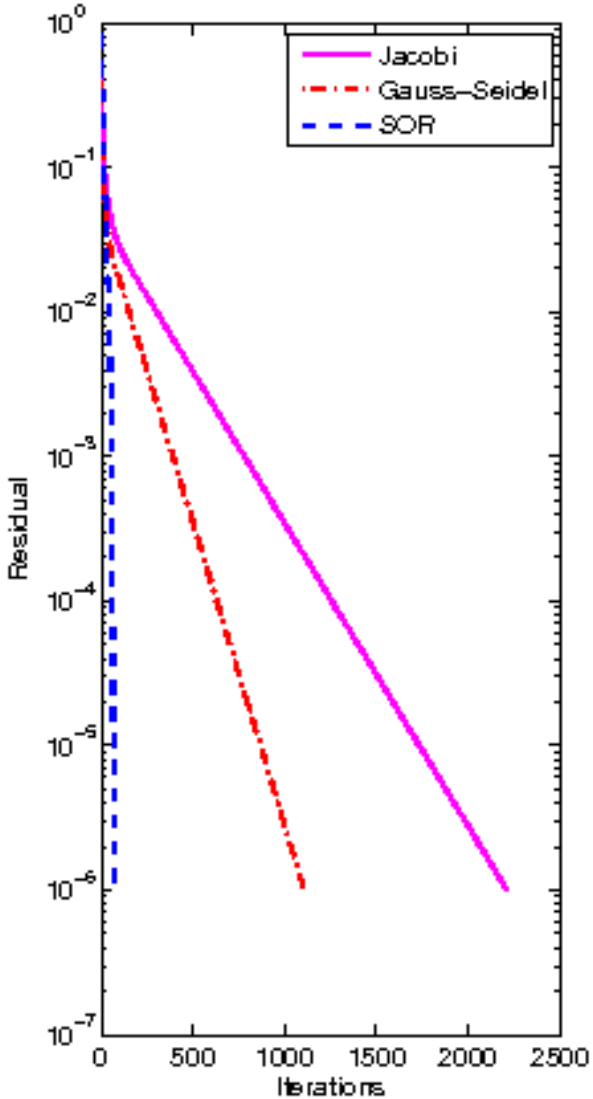
$$\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}.$$

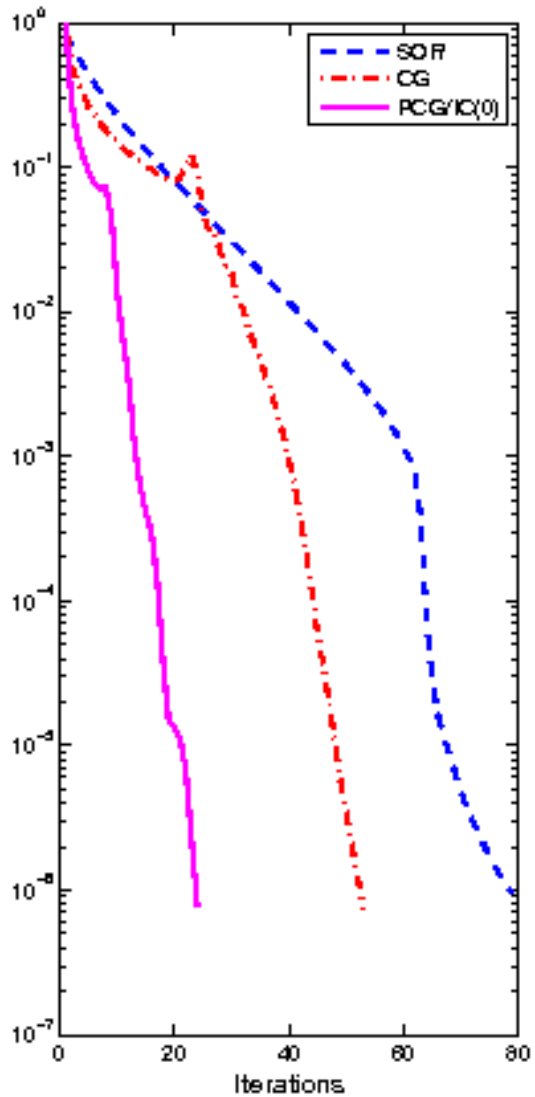
The Conjugate Gradient Method defines search directions that are A-conjugate, and minimizes $\|\mathbf{e}_k\|_A = \sqrt{\mathbf{e}_k^T A\mathbf{e}_k}$. Note that this is well defined only if A is SPD.

The Conjugate Gradient Algorithm

Given an initial guess $\mathbf{x}_0$ and a tolerance *tol*, set at first $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\delta_0 = \langle \mathbf{r}_0, \mathbf{r}_0 \rangle$, $b_\delta = \langle \mathbf{b}, \mathbf{b} \rangle$, $k = 0$ and $\mathbf{p}_0 = \mathbf{r}_0$. Then: While $\delta_k > tol^2 \; b_\delta$,

$$
\begin{aligned}
\mathbf{s}_k &= A\mathbf{p}_k \\
\alpha_k &= \frac{\delta_k}{\langle \mathbf{p}_k, \mathbf{s}_k \rangle} \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\
\mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{s}_k \\
\delta_{k+1} &= \langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle \\
\mathbf{p}_{k+1} &= \mathbf{r}_{k+1} + \frac{\delta_{k+1}}{\delta_k} \mathbf{p}_k \\
k &= k + 1.
\end{aligned}
$$


Jacobi
Gauss–Seidel
SOR
Residual
Iterations
0
500
1000
1500
2000
2500


SOR
CG
PCG/IC(0)
Iterations
0
20
40
60
80

Krylov Subspace Methods

We are seeking to find a solution within the Krylov subspace

$$\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}^k(A; \mathbf{r}_0) \equiv \mathbf{x}_0 + span\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\}.$$

- Find a good basis for the space (riddle: 'good' means what??): Lanczos or Arnoldi will help here.
- Optimality condition:
  - Require that the norm of the residual $\|b - Ax_k\|_2$ is minimal over the Krylov subspace.
  - Require that the residual is orthogonal to the subspace.

Well Known Methods

- CG for SPD matrices
- GMRES for nonsymmetric, MINRES for symmetric indefinite
- BiCG-Stab (non-optimal: based on bidiagonalization)
- QMR: Quasi-Minimal Residuals.
- A few more...

Preconditioning — Motivation

Convergence rate typically depends on two factors:

- Distribution/clustering of eigenvalues **(crucial!)**
- Condition number (the **less** important factor!)

Idea: Since A is given and is beyond our control, define a matrix M such that the above properties are better for $M^{-1}A$, and solve $M^{-1}Ax = M^{-1}b$ rather than $Ax = b$.

Requirements: To produce an effective method the preconditioner matrix M must be easily invertible. At the same time it is desirable to have at least one of the following properties hold:
$\kappa(M^{-1}A) \ll \kappa(A)$, and/or the eigenvalues of $M^{-1}A$ are much better clustered compared to those of A.

Basically, Two Types of Preconditioners

- Algebraic, general purpose (arguably, frequently needed in continuous optimization)
- Specific to the problem (arguably, frequently needed in PDEs)

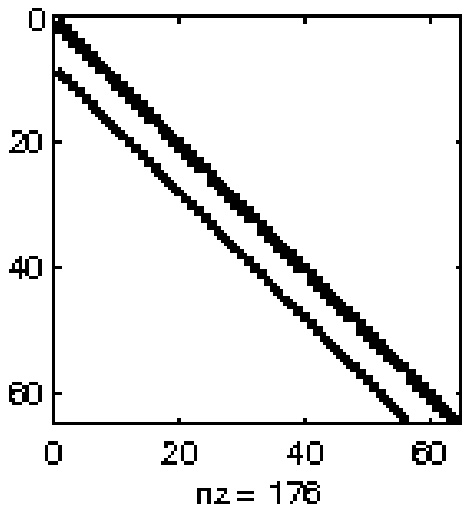
Important Classes of Preconditioners

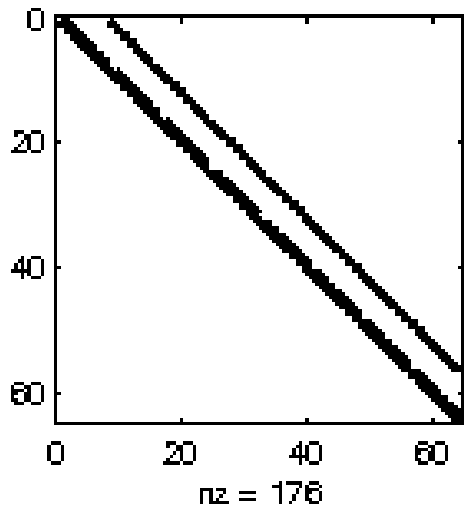
Preconditioning is a combination of art and science...

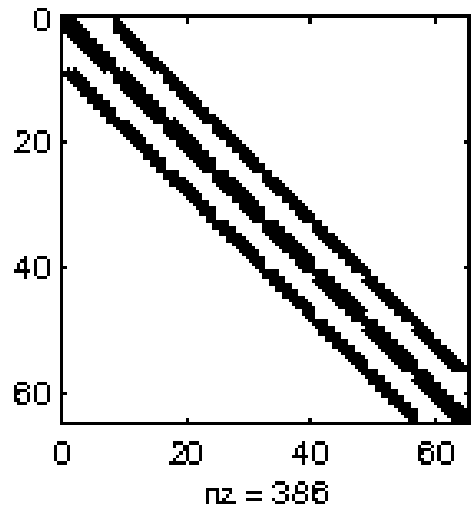
- Stationary preconditioners, such as Jacobi, Gauss-Seidel, SOR.
- Incomplete factorizations.
- Multigrid and multilevel preconditioners. (Advanced)
- Preconditioners tailored to the problem in hand, that rely for example on the properties of the underlying differential operators.

Incomplete Factorizations

Given the matrix A, construct an LU decomposition or a Cholesky decomposition (if A is symmetric positive definite) that follows precisely the same steps as the usual decomposition algorithms, except that a nonzero entry of a factor is generated *only if* the matching entry of A is nonzero!


0
20
40
60
0
20
40
60
nz = 176


0
20
40
60
0
20
40
60
nz = 176


0
20
40
60
0
20
40
60
nz = 386

MATLAB

```
R=cholinc(A,'0');
x=pcg(A,b,tol,maxit,R',R);
```

... and many other commands and features.

Problem-Tailored Preconditioners

Schur complement based methods, e.g. for Stokes and Maxwell.

To be discussed (time permitting).

The END