NE 155, Classes 13 & 14, S15
Solutions for Linear Systems
February 20 and 23 and 12, 2015

Solutions of Linear Systems

Recall: Given $\mathbf{A}\vec{x} = \vec{b}$, how do you find $\vec{x}$?

We talked about how to do this directly, now we'll talk about how to do this other ways.

- Iterative Methods
  - produce a sequence of vectors, $\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots$ based on the prescription

$$\vec{x}^{(k+1)} = F\big(\vec{x}^{(k)}, \vec{b}\big)\,, \qquad \text{where } \lim_{k\to\infty} \vec{x}^{(k)} = \vec{x}$$

  - can require many FLOPs; storage requirements for $\mathbf{A}$ can be an issue (if stored)
  - may parallelize well
  - *often* used in practice
  - used in smoothing methods and as pre-conditioners
  - Jacobi, Gauss-Seidel, SOR, Multigrid, ...
- Semi-Iterative Methods
  - produce a sequence of vectors, $\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots$ based on the prescription

$$\vec{x}^{(k+1)} = F\big(\vec{x}^{(k)}, \vec{x}^{(k-1)}, \ldots, \vec{x}^{(0)}, \vec{b}\big)\,, \qquad \text{where } \lim_{k\to\infty} \vec{x}^{(k)} = \vec{x}$$

    The function does not necessarily have to use all previous $\vec{x}$ iterates.
  - can require many FLOPs; storage requirements for $\mathbf{A}$ can be an issue (if stored)
  - may parallelize well
  - Commonly used in conjugate gradients, generalized minimal residuals methods

Iterative Methods

We need iterative methods when our system of linear equations is large. Recall: produce a sequence of vectors, $\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots$ based on the prescription

$$\vec{x}^{(k+1)} = F(\vec{x}^{(k)}, \vec{b}) \,, \qquad \text{where } \lim_{k\to\infty} \vec{x}^{(k)} = \vec{x}$$

We will write this as

$$\vec{x}^{(k+1)} = \mathbf{P}\vec{x}^{(k)} + \tilde{\vec{b}} \,, \qquad k = 0, 1, \ldots$$

We will split $\mathbf{A}$ to execute our process.

$$\begin{aligned}
(\mathbf{A} + \mathbf{S})\vec{x} &= \mathbf{S}\vec{x} + \vec{b} \\
\mathbf{Q}\vec{x} &= \mathbf{S}\vec{x} + \vec{b} \qquad \text{where } \mathbf{Q} = \mathbf{A} + \mathbf{S} \\
\vec{x} &= \underbrace{\mathbf{Q}^{-1}\mathbf{S}}_{\mathbf{P}}\vec{x} + \underbrace{\mathbf{Q}^{-1}\vec{b}}_{\tilde{\vec{b}}} \qquad \text{assuming regular } \mathbf{Q}
\end{aligned}$$

The **fixed-point** iterative process is:

$$\begin{aligned}
\vec{x}^{(0)} &= \text{ arbitrary} \\
\vec{x}^{(k+1)} &= \mathbf{P}\vec{x}^{(k)} + \tilde{\vec{b}}
\end{aligned}$$

To determine convergence, we define the iterative error as follows, noting that $\vec{x}$ is defined as the solution to $\mathbf{A}\vec{x} = \vec{b}$.

$$\vec{e}^{(k)} = \vec{x}^{(k)} - \vec{x}$$

Subtract the exact solution from the method

$$\begin{aligned}
\vec{x}^{(k+1)} &= \mathbf{P}\vec{x}^{(k)} + \tilde{\vec{b}} \\
-(\vec{x} &= \mathbf{P}\vec{x} + \tilde{\vec{b}}) \\
\vec{x}^{(k+1)} - \vec{x} &= \mathbf{P}(\vec{x}^{(k)} - \vec{x})
\end{aligned}$$

$$\begin{aligned}
\text{Thus, } \vec{e}^{(k+1)} &= \mathbf{P}\vec{e}^{(k)} \\
\text{Further, } \vec{e}^{(k+1)} &= \mathbf{P}^2\vec{e}^{(k-1)} = \mathbf{P}^3\vec{e}^{(k-2)} = \cdots = \mathbf{P}^{k+1}\vec{e}^{(0)}
\end{aligned}$$

Now we can use norms to determine a rate of convergence:

$$||\vec{e}^{(k+1)}|| = ||\mathbf{P}^{k+1}\vec{e}^{(0)}|| \leq ||\mathbf{P}^{k+1}||||\vec{e}^{(0)}|| \,,$$

and the rate of convergence is therefore governed by $||\mathbf{P}^{k+1}||$.

There is a theorem (offered without proof):

$$\lim_{k\to\infty} ||\mathbf{P}^k||^{1/k} = \rho(\mathbf{P}) \,,$$

where $\rho(\mathbf{P})$ is the spectral raidus of $\mathbf{P}$, and this implies

$$||\mathbf{P}^k|| \approx \rho^k(\mathbf{P}) \,.$$

Alternative: Recall that $||\mathbf{P}^k||_2 = \rho(\mathbf{P}^k)$ and that $\rho^k(\mathbf{P}) = \rho(\mathbf{P}^k)$. We also know that all norms can be related, so finding an error bound in one norm bounds all norms.

Thus, a sufficient and necessary condition for convergence is:

$$\rho^{k+1}(\mathbf{P}) < 1 \rightarrow \rho(\mathbf{P}) < 1$$

Note: $\mathbf{Q}^{-1}$ must be easy to compute. We often select $\mathbf{Q}$ to be diagonal, triangular, or tri-diagonal. Further, $\mathbf{Q}$ must be chosen such that $\rho(\mathbf{P}) < 1$.

Richardson Iteration / Source Iteration

One of the simplest methods out there is Richardson/Source Iteration. There are two very similar versions. In *non-stationary* Richardson iteration that uses some scalar ω, the kth step is

$$\begin{aligned} \vec{x}^{(k+1)} &= (\mathbf{I} - \omega^{(k)}\mathbf{A})\vec{x}^{(k)} + \omega^{(k)}\vec{b} \,, \qquad \text{which gives} \\ \mathbf{P}_R &= (\mathbf{I} - \omega^{(k)}\mathbf{A}) \,. \end{aligned}$$

If $\omega^{(k)}$ is constant, then this is the *stationary* Richardson method.

The algorithm for this method is, for $i = 1, \ldots, n$:

$$x_i^{(k+1)} = x_i^{(k)} - \omega^{(k)} \sum_{j=1}^{n} a_{ij} x_j^{(k)} + \omega^{(k)} b_i \; .$$

People still use these methods. I've written a preconditioner that uses Richardson Iteration. The spectral radius of $\mathbf{P}_R$ determines the speed of convergence and is $c = \Sigma_s / \Sigma$. For problems dominated by scattering, this method will converge very slowly [3]. Fortunately, there are other more sophisticated methods available.

Jacobi Iteration

Let $\mathbf{D} = \text{diag}(\mathbf{A})$, then

$$\begin{aligned} \mathbf{D}\vec{x}^{k+1} &= (\mathbf{D} - \mathbf{A})\vec{x}^{(k)} + \vec{b} \\ \vec{x}^{k+1} &= \mathbf{D}^{-1}(\mathbf{D} - \mathbf{A})\vec{x}^{(k)} + \mathbf{D}^{-1}\vec{b} \end{aligned}$$

Now, $\mathbf{P}_J = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$ and $\tilde{\vec{b}} = \mathbf{D}^{-1}\vec{b}$.

The algorithm for this method is, for $i = 1, \ldots, n$:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}) \; .$$

Jacobi will converge (perhaps quite slowly) if $\mathbf{P}_J$'s spectral radius is < 1, but that's often difficult to estimate [4].

A sufficient but not necessary condition for convergence is that $\mathbf{A}$ or $\mathbf{A}^T$ is **diagonally dominant**:

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \qquad \forall i \; .$$

Jacobi isn't a very sophisticated method, but it's easy to parallelize. Why might that be?
The Jacobi method is order independent since all terms on the right are at the old iteration level.

Gauss Seidel Iteration

Gauss Seidel is a simple modification of Jacobi, but becomes a bit more complicated. How can you think of changing Jacobi than might incorporate more information that we already have?

The algorithm for this method is, for $i = 1, \ldots, n$:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}) \, .$$

In matrix notation, this looks like:

$$(\mathbf{D} + \mathbf{L})\vec{x}^{k+1} = -\mathbf{U}\vec{x}^{(k)} + \vec{b} \, ,$$

where we have separated $\mathbf{A}$ as $\mathbf{L} + \mathbf{D} + \mathbf{U}$. This makes $\mathbf{P}_{GS} = (\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}$. Gauss Seidel converges twice as fast as Jacobi (not proven here). However, both methods may converge very slowly - especially in highly scattering systems [4].

The spectral radii for some materials of practical interest determined using cross sections with 41 thermal upscattering groups are: graphite = 0.9984, heavy water = 0.9998, and iron = 0.6120 [1], [3].

A sufficient but not necessary condition to converge is that $\mathbf{A}$ be **symmetric** and **positive definite**.

An $n \times n$ Hermitian matrix $\mathbf{A}$ is said to be positive definite if $\vec{z}^H \mathbf{A} \vec{z}$ is real and positive for all non-zero complex vectors $\vec{z}$. This also simplifies to the pure-real case.

Gauss Seidel is not so easy to parallelize. Why not?
The GS method is order dependent, containing terms on the right hand side at both the new and old iterates.

Successive Over Relaxation (SOR)

SOR is similar to GS, but now we add a splash of control to try to converge faster than GS (sort of like weighted Richardson).

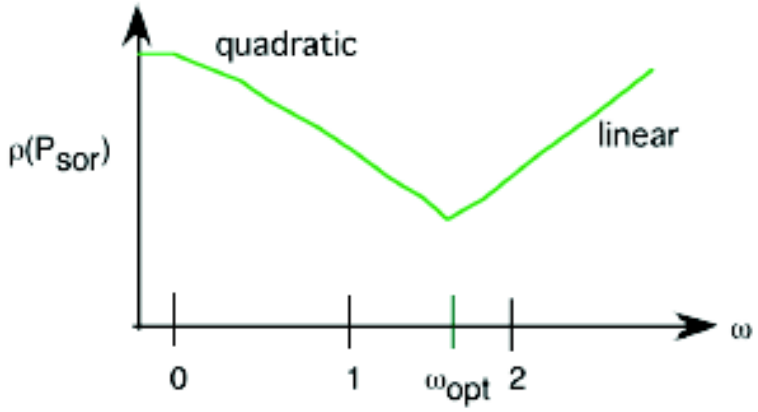
$$(\mathbf{D} + \omega\mathbf{L})\vec{x}^{k+1} = [(1-\omega)\mathbf{D} - \omega\mathbf{U}]\vec{x}^{(k)} + \omega\vec{b} \, ,$$

where $0 < \omega < 2$. Now $\mathbf{P}_{SOR} = (\mathbf{D} + \omega\mathbf{L})^{-1}[(1-\omega)\mathbf{D} - \omega\mathbf{U}]$.

For $i = 1, \ldots, n$:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}) \, .$$

Whether and how much better we do that GS is determined by ω. Determining the best ω, ω_{opt} is problem-dependent and is not practical to do in general.



Number of Iterations?

How many iterations does it take to reduce error by a factor of ϵ?

$$\begin{aligned} ||\vec{e}^{(k)}|| &\leq \epsilon ||\vec{e}^{(0)}|| \\ ||\vec{e}^{(k)}|| &\leq \rho(\mathbf{P})^k ||\vec{e}^{(0)}|| \\ \rho(\mathbf{P})^k &\approx \epsilon \qquad \text{take the log of both sides} \\ k\log(\rho(\mathbf{P})) &\approx \log(\epsilon) \\ k &\approx \frac{\log(\epsilon)}{\log(\rho(\mathbf{P}))} \end{aligned}$$

Preconditioning

Condition number information is derived from Trefethen and Bau [5]. Conditioning is one way to express the perturbation behavior of the mathematical problem.

A *well-conditioned* problem is one in which all small perturbations of x lead to only small changes in $f(x)$.

An *ill-conditioned* problem is one in which some small perturbation of x leads to a large change in $f(x)$.

The condition number is a quantity used to express how well-conditioned a matrix or problem is. A small condition number corresponds to a well-conditioned problem, and vice versa.

The **condition number** of a matrix $\mathbf{A}$ is defined as

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

If the 2-norm is used, then $||\mathbf{A}||_2 = \sigma_1$, $||\mathbf{A}^{-1}||_2 = 1/\sigma_m$, and $\kappa_2(\mathbf{A}) = \sigma_1/\sigma_m$; σ_m is the mth singular value of $\mathbf{A}$. If $\mathbf{A}$ is singular, its condition number is infinity.

The base-b logarithm of κ is an estimate of how many base-b digits are lost in solving a linear system with that matrix. In other words, it estimates worst-case loss of precision. A system is said to be singular if the condition number is infinite, and ill-conditioned if it is too large, where "too large" means roughly $\log(\kappa) >\sim$ the precision of matrix entries.

The general idea of **preconditioning** is to transform the system of interest into another equivalent system that has more favorable properties, for example one with a smaller condition number.

A **preconditioner** is a matrix that induces such a transformation by improving the spectral properties of the problem being solved.

Let $\mathbf{G}$ be a non-singular preconditioner, then [2] $\mathbf{A}\vec{x} = \vec{b}$ can be transformed as

$$\mathbf{G}^{-1}\mathbf{A}x = \mathbf{G}^{-1}b\ .$$

Strictly speaking, Jacobi is "preconditioned Richardson iteration", with $\mathbf{G} = \mathbf{D}$.

Functionally, a good preconditioner should make the system easier to solve and result in faster

convergence. It should also be cheap to construct and apply.

There are many different types of preconditioners, but they can be put into two general categories: **matrix-based** and **physics-based**.

Matrix-based preconditioners rely entirely on the structure of the matrix $\mathbf{A}$ regardless of the physics of the problem. That is, these methods do not change when the underlying problem changes. This can be a very useful property because matrix-based methods are then broadly applicable and do not require any understanding of the physical problem [5].

Physics-based preconditioning uses knowledge about the physics of the problem in question to guide the creation of the preconditioner. This means that some methods only work with certain kinds of problems, and that the preconditioners may have to be tailored or adapted for different applications. However, such methods take advantage of knowing something about the problem and can be more effective than matrix-based methods for the range of problems for which they are intended [5].

Examples

Let's look at

$$\begin{pmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 24 \\ 30 \\ -24 \end{pmatrix}$$

with both GS and SOR. In both cases we'll start with $\vec{x}^{(0)} = [1, 1, 1]^T$. The exact solution is $[3, 4, -5]^T$.

GS: for $i = 1, \ldots, n$:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}) \, .$$

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{4}(24 - 3x_2^{(k)} - 0x_3^{(k)}) = 6 - \frac{3}{4}x_2^{(k)} \\ x_2^{(k+1)} &= \frac{1}{4}(30 - 3x_1^{(k+1)} - (-1)x_3^{(k)}) = \frac{30}{4} - \frac{3}{4}x_1^{(k+1)} + \frac{1}{4}x_3^{(k)} \\ x_3^{(k+1)} &= \frac{1}{4}(-24 - 0x_1^{(k+1)} - (-1)x_2^{(k+1)}) = -6 + \frac{1}{4}x_2^{(k+1)} \end{aligned}$$

SOR: for $i = 1, \ldots, n$:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}) \, .$$

Let $\omega = 1.25 = 5/4$

$$x_1^{(k+1)} = (1-\frac{5}{4})x_1^{(k)} + \frac{\frac{5}{4}}{4}(24 - 3x_2^{(k)} - 0x_3^{(k)}) = \frac{15}{2} - \frac{1}{4}x_1^{(k)} - \frac{15}{16}x_2^{(k)}$$
$$x_2^{(k+1)} = (1-\frac{5}{4})x_2^{(k)} + \frac{\frac{5}{4}}{4}(30 - 3x_1^{(k+1)} - (-1)x_3^{(k)}) = \frac{150}{16} - \frac{15}{16}x_1^{(k+1)} - \frac{1}{4}x_2^{(k)} + \frac{5}{16}x_3^{(k)}$$
$$x_3^{(k+1)} = (1-\frac{5}{4})x_3^{(k)} + \frac{\frac{5}{4}}{4}(-24 - 0x_1^{(k+1)} - (-1)x_2^{(k+1)}) = -\frac{15}{2} + \frac{5}{16}x_2^{(k+1)} - \frac{1}{4}x_3^{(k)}$$

Table 1: Gauss Seidel

k	0	1	2	3	4	|error|
x_1	1	5.25	3.140625	3.087891	3.05493	0.05493
x_2	1	3.8125	3.8828125	3.92676	3.95422	0.04578
x_3	1	-5.046875	-5.0292969	-5.01831	-5.00715	0.00715

Table 2: SOR

k	0	1	2	3	4	|error|
x_1	1	6.3125	2.6223	3.1333	2.95705	0.04295
x_2	1	3.51953	3.95853	4.01026	4.00748	0.00748
x_3	1	-6.65015	-4.60042	-5.096686	-4.97349	0.02651

References

[1] M. L. Adams and E. W. Larsen. Fast Iterative Methods for Discrete-Ordinates Particle Transport Calculations. *Progress in Nuclear Energy*, 40(1):3–159, 2002.

[2] M. Benzi. Preconditioning Techniques for Large Linear Systems: A Survey. *Journal of Computational Physics*, 182:418–477, 2002.

[3] T. M. Evans, A. S. Stafford, R. N. Slaybaugh, and K. T. Clarno. Denovo – A New Three-Dimensional Parallel Discrete Ordinates Code in SCALE, 2009.

[4] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. SIAM, Philadelphia, PA, 2007.

[5] L. N. Trefethen and D. Bau, III. *Numerical Linear Algebra*. SIAM, Philadelphia, PA, 1997.