Solving linear systems on an industrial scale

- Brief revision and remarks on LU factorisation
- Iterative methods
  - Jacobi method
  - Gauss-Seidel method
  - Relaxation method
- Pivoting procedures

Wednesday, 3 April 2018

Revision: LU factorisation

LU summary: To solve a system $\mathbf{Ax} = \mathbf{b}$:

In case any row swaps are *not* required:

1. Factorise $\mathbf{A} = \mathbf{LU}$, where $\mathbf{L}$ is a lower triangular and $\mathbf{U}$ is an upper triangular matrix. Then $\mathbf{LUx} = \mathbf{b}$.
2. Denoting $\mathbf{y} = \mathbf{Ux}$, solve $\mathbf{Ly} = \mathbf{b}$ for $\mathbf{y}$ (with forward substitution).
3. Having obtained $\mathbf{y}$, solve $\mathbf{Ux} = \mathbf{y}$ for $\mathbf{x}$ (with backward substitution).

Revision: LU factorisation

To solve a system $\mathbf{Ax} = \mathbf{b}$ in the most general case:

1. Perform row swaps first (importantly, for both $\mathbf{A}$ and $\mathbf{b}$): $\mathbf{PAx} = \mathbf{Pb}$
2. Factorise $\mathbf{PA} = \mathbf{LU}$, where $\mathbf{L}$ is a lower triangular and $\mathbf{U}$ is an upper triangular matrix. Then $\mathbf{LUx} = \mathbf{Pb}$.
3. Denoting $\mathbf{y} = \mathbf{Ux}$, solve $\mathbf{Ly} = \mathbf{Pb}$ for $\mathbf{y}$ (with forward substitution).
4. Having obtained $\mathbf{y}$, solve $\mathbf{Ux} = \mathbf{y}$ for $\mathbf{x}$ (with backward substitution).

The general strategy for row swaps is in arranging the largest numbers along the matrix diagonal, as much as possible.

Revision: Elementary matrices

Gaussian reduction is represented with elementary matrices, e.g.

$$\mathbf{E}_{22}|_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_{23} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{E}_{21}|_k = \begin{bmatrix} 1 & 0 & 0 \\ k & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Generally,

$$e_{ii} = k \quad \Leftrightarrow \quad \mathsf{R}_i \to kR_i \qquad \text{multiplies row } i \text{ by } k$$

$$\begin{cases} e_{ij} = e_{ji} = 1 \\ e_{ii} = e_{jj} = 0 \end{cases} \quad \Leftrightarrow \quad \mathsf{R}_i \leftrightarrow R_j \qquad \text{swaps rows } i \text{ and } j$$

$$e_{ij} = k \quad \Leftrightarrow \quad \mathsf{R}_i \to (R_i + kR_j) \qquad \text{adds } k \times \text{row } j \text{ to row } i$$

Subsequent multiplication (from the left) $\mathbf{E}_m \dots \mathbf{E}_2 \, \mathbf{E}_1 \, \mathbf{A}$ by a chain of appropriate elementary matrices reduces $\mathbf{A}$ into REF.

Revision: Gaussian reduction and LU factorisation

In case row swaps are *not* required (or after all the swaps):

- Row reduction process is: $\mathbf{E}_{32}\,\mathbf{E}_{31}\,\mathbf{E}_{21}\,\mathbf{A} = \mathbf{U}$
- This is equivalent to

$$\mathbf{A} = \left(\mathbf{E}_{32}\,\mathbf{E}_{31}\,\mathbf{E}_{21}\,\right)^{-1}\mathbf{U} = \mathbf{E}_{21}^{-1}\,\mathbf{E}_{31}^{-1}\,\mathbf{E}_{32}^{-1}\,\mathbf{U} = \mathbf{L}\mathbf{U}$$

- So with a row-reduction algorithm, we obtain

$$\mathbf{L} = \mathbf{E}_{21}^{-1}\,\mathbf{E}_{31}^{-1}\,\mathbf{E}_{32}^{-1}$$

$$\mathbf{U} = \mathbf{E}_{32}\,\mathbf{E}_{31}\,\mathbf{E}_{21}\,\mathbf{A}$$

- The inverted elementary matrices revert the original action.
- Except from row swaps, elementary matrices and their inverses are lower triangular.

Recall that slide with mistake

These subsequent multiplications reduce $\mathbf{A}$ into echelon form

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -7 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{bmatrix}$$

Multiplication of such elementary matrices produces a low triangular matrix. Regarding the last equation as $\mathbf{L}^{-1}\mathbf{A} = \mathbf{U}$

$$\text{where} \quad \mathbf{L}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \quad \text{and so} \quad \mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix}$$

this result provides a factorisation $\mathbf{A} = \mathbf{LU}$.

Revision: LU factorisation methods

Methods for constructing $\mathbf{A} = \mathbf{L}\mathbf{U}$:

- **Doolittle** method implies that the diagonal elements of the lower triangular matrix $\mathbf{L}$ are equal to 1.
- **Crout** method by contrast, requires that the diagonal elements of the upper triangular matrix $\mathbf{U}$ are equal to 1.
- **Cholesky** method is only applicable to symmetric matrices $\mathbf{A} = \mathbf{A}^{\mathsf{T}}$ and provides a quick $\mathbf{A} = \mathbf{U}^{\mathsf{T}}\mathbf{U}$ decomposition.

If these algorithms fail, retrying upon swapping the rows can help.

Iterative solution of linear systems

On a large scale, even the efficient methods so far discussed, may have some disadvantages.

- Matrix inversion only yields $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ for square non-singular matrices. It is time consuming, and mostly suitable for $\mathbf{Ax} = \mathbf{b}_i$.
- Gaussian elimination is faster, but it is not that convenient for variable right-hand side $\mathbf{Ax} = \mathbf{b}_i$.
- LU decompositions may also be not optimal for huge systems, particularly in case row swaps are required.

Sometimes an approximate solution can be found by making an initial approximation $\mathbf{x}_0$ to the solution, and then carrying out an **iterative procedure**.

Fixed point iteration

Recall Newton's method for solving $f(x) = 0$: $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$

In general, fixed point iteration uses $x_{k+1} = g(x_k)$ for a suitable g.

Theorem: (1D fixed point theorem):
Suppose that the equation $g(x) = x$ has a solution $x = s$, and there is an interval $\Xi = [s - \xi,\ s + \xi]$ in which $g'(x)$ exists and satisfies $|g'(x)| \leqslant m < 1$; and that $g\big(x \in \Xi\big) \in \Xi$. Then

- $\forall\, x_0 \in \Xi$ the sequence of iterations

$$x_{k+1} = g(x_k), \quad k = 0, 1, 2, \ldots \quad \in \Xi$$

- $\lim\limits_{k \to \infty} x_k = s$
- s is the only solution to $g(x) = x$ within Ξ

Fixed point iteration

This theorem makes it sometimes possible to obtain an approximate solution to a nonlinear equation $f(x) = 0$ by

- making an initial guess x_0,
- rewriting the equation in the form $x = g(x)$
- performing iterations:

$$
\begin{aligned}
x_1 &= g(x_0) \\
x_2 &= g(x_1) \\
&\vdots \\
x_{k+1} &= g(x_k) \\
&\vdots
\end{aligned}
$$

until the sequence $\{x_k\}$ appears to converge.

Fixed point iteration: example

Example: Find the smaller root of $x^2 - 10x + 1 = 0$.

Rewrite the equation as $x = g(x) = \dfrac{1 + x^2}{10}$, so $g'(x) = \dfrac{x}{5}$.

Clearly, if $|x| \leqslant m < 5$ then $|g'(x)| \leqslant m/5 < 1$
so the conditions of the theorem are satisfied.

Take $x_0 = 0$; then $x_1 = \frac{1+x_0^2}{10} = 0.1$

$x_2 = \frac{1+x_1^2}{10} = \frac{1+0.1^2}{10} = 0.101$

$x_3 = \frac{1+x_2^2}{10} = \frac{1+(0.101)^2}{10} = 0.1010201$

$x_4 = \cdots = 0.10102050606040 1$

Application of the exact formula gives the smaller root as
$x = 5 - \sqrt{24} = 0.10102051443364 4 \ldots$.

Iterative solution of linear systems

For iterative solution of linear systems, we need:

- iteration algorithm
- convergence control
- conditions guaranteeing convergence
- practical stopping criterion

The key question for iterative solution of a linear system, is when to terminate the procedure.

To decide when the calculation can be halted, we need a measure of how close an approximate solution is to the true solution.

This requires some basic knowledge of the norms.

Some basics regarding the norm of a vector

A norm of a vector $\mathbf{x}$ (x_i with $i = 1 \ldots n$) has the properties:

- $\|\mathbf{x}\| \geqslant \mathbf{0} \ \forall \mathbf{x}$
- $\|\mathbf{x}\| = \mathbf{0}$ if and only if $\mathbf{x} = \mathbf{0}$
- $\|\alpha\mathbf{x}\| = |\alpha| \, \|\mathbf{x}\| \ \forall \alpha$ and $\forall \mathbf{x}$
- $\|\mathbf{x} + \mathbf{y}\| \leqslant \|\mathbf{x}\| + \|\mathbf{y}\| \ \forall \mathbf{x}, \mathbf{y}$

There are many ways to define a norm, for example:

- l_1 norm: $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$
- l_2 norm (usual Euclidean length): $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$
- l_∞ (*supremum*) norm: $\|\mathbf{x}\|_\infty = \max_{1 \leqslant i \leqslant n} |x_i|$

Examples of norms

Example: for $\mathbf{x} = \begin{pmatrix} 1 \\ -2 \\ 3 \\ -4 \end{pmatrix}$

$$\|\mathbf{x}\|_1 = |1| + |-2| + |3| + |-4| = 10$$

$$\|\mathbf{x}\|_2 = \sqrt{1^2 + (-2)^2 + 3^2 + (-4)^2} = \sqrt{30}$$

$$\|\mathbf{x}\|_\infty = |-4| = 4$$

A specific norm to use depends on the nature of a problem.

Norm as a distance between vectors

A generalised *distance* between $\mathbf{x}$ and $\mathbf{y}$ is the norm of $\mathbf{x}-\mathbf{y}$, e.g.

- l_1 distance: $\|\mathbf{x}-\mathbf{y}\|_1 = \sum_{i=1}^{n} |x_i - y_i|$
- l_2 distance (Euclidean distance): $\|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
- l_∞ (*supremum*) distance: $\|\mathbf{x}-\mathbf{y}\|_\infty = \max_{1 \leqslant i \leqslant n} |x_i - y_i|$

This is used, in particular, to check convergence of iterations.

Convergence of a sequence

Definition:

A sequence $\left\{\mathbf{x}^{(k)}\right\}$ converges to a limit $\mathbf{v}$ with respect to the norm, if $\forall\, \epsilon > 0 \quad \exists\, N \in \mathbb{N}$ such that $\forall\, k > N$

$$\left\|\mathbf{x}^{(k)} - \mathbf{v}\right\| < \epsilon$$

Stopping criteria

When implementing an iterative method for solving $\mathbf{Ax} = \mathbf{b}$, we can stop iterations when the difference between successive approximations $\mathbf{x}^{(k)}$, $\mathbf{x}^{(k+1)}$ becomes sufficiently small.

Commonly used stopping criteria (ϵ is a small positive number):

- Stop when the difference between successive iterates satisfies

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \epsilon$$

- Stop when the difference between successive iterates satisfies

$$\frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^{(k+1)}\|} < \epsilon$$

(note: also impose an upper bound on the number of iterations)

Jacobi iteration method

Consider a linear system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ \vdots \quad & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. \end{aligned}$$

This system can be rewritten in the following form

$$\begin{aligned} x_1 &= \frac{1}{a_{11}} \left(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n \right), \\ x_2 &= \frac{1}{a_{22}} \left(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n \right), \\ \vdots \quad & \vdots \\ x_n &= \frac{1}{a_{nn}} \left(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n(n-1)}x_{n-1} \right). \end{aligned}$$

Jacobi iteration method

Once we decide on the initial approximation $x_i^{(0)}$, $i = 1, 2, ..., n$, the next iteration is generated by

$$x_i^{(1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(0)} \right).$$

From $x_i^{(1)}$ we proceed to the next iteration $x_i^{(2)}$:

$$x_i^{(2)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(1)} \right).$$

$\mathbf{x}^{(0)} = \mathbf{0}$ is often used as the first approximation: $x_i^{(0)} = 0 \ \forall\, i$.

Jacobi iteration method

In this way, a sequence of approximations $\left\{x_i^{(k)}\right\}$ is generated:

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

These iterations can be stopped when

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \epsilon \qquad \text{or} \qquad \frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^{(k+1)}\|} < \epsilon$$

depending on the desired precision (defined by the choice of ϵ).

An open question however is, whether the sequence $\left\{x_i^{(k)}\right\}$ converges to the true solution.

Convergence of Jacobi method

A sufficient condition for convergence is the *diagonal dominance* of the corresponding matrix of the system $\mathbf{Ax} = \mathbf{b}$.

Recall: a square matrix $\mathbf{A}$ is strictly diagonally dominant if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \forall\, i$$

Theorem: If the matrix $\mathbf{A}$ of a linear system $\mathbf{Ax} = \mathbf{b}$ is strictly diagonally dominant, then given any initial guess $x_i^{(0)}$, the Jacobi iterations $\left\{x_i^{(k)}\right\}$ will converge to the true solution of the system.

Jacobi iteration: Example

Solve the following linear system by Jacobi iterations, using the stopping criterion $\|x^{(k+1)} - x^{(k)}\|_\infty < 3 \cdot 10^{-5}$.

$$\begin{aligned} 3x_1 + x_2 + x_3 &= 5 \\ 2x_1 + 6x_2 + x_3 &= 9 \\ x_1 + x_2 + 4x_3 &= 6 \end{aligned}$$

Solution: The matrix of the system is diagonally dominant: $3 > 1 + 1,\ 6 > 2 + 1,\ 4 > 1 + 1.$

To use Jacobi iteration, we rewrite the system in the form

$$\begin{aligned} x_1 &= \frac{1}{3}(5 - x_2 - x_3) \\ x_2 &= \frac{1}{6}(9 - 2x_1 - x_3) \\ x_3 &= \frac{1}{4}(6 - x_1 - x_2) \end{aligned}$$

Jacobi iteration: Example

This leads to the iterative scheme

$$
\begin{aligned}
x_1^{(k+1)} &= \frac{1}{3}\left(5 - x_2^{(k)} - x_3^{(k)}\right) \\
x_2^{(k+1)} &= \frac{1}{6}\left(9 - 2x_1^{(k)} - x_3^{(k)}\right) \\
x_3^{(k+1)} &= \frac{1}{4}\left(6 - x_1^{(k)} - x_2^{(k)}\right)
\end{aligned}
$$

For the starting set we take: $x_1^{(0)} = 0, \quad x_2^{(0)} = 0, \quad x_3^{(0)} = 0\,.$

Then the first iteration is: $x_1^{(1)} = \frac{5}{3}, \quad x_2^{(1)} = \frac{3}{2}, \quad x_3^{(1)} = \frac{3}{2}\,.$

Jacobi iteration: Example

From the above first iteration $x_1^{(1)} = 5/3,\ x_2^{(1)} = 3/2,\ x_3^{(1)} = 3/2,$ the second iteration is

$$x_1^{(2)} = \frac{1}{3}\left(5 - \frac{3}{2} - \frac{3}{2}\right) = \frac{2}{3}$$
$$x_2^{(2)} = \frac{1}{6}\left(9 - 2 \cdot \frac{5}{3} - \frac{3}{2}\right) = \frac{25}{36}$$
$$x_3^{(2)} = \frac{1}{4}\left(6 - \frac{5}{3} - \frac{3}{2}\right) = \frac{17}{24}$$

Continuing these iterations, we eventually get

$$x_1^{(20)} = 0.9999906 \qquad x_2^{(20)} = 0.9999920 \qquad x_3^{(20)} = 0.9999922$$

This satisfies the stopping criterion, so the algorithm terminates after 20 iterations. The true solution is actually $x_1 = x_2 = x_3 = 1$.

Gauss-Seidel iterations

- Jacobi iterations can be slow. Each step is entirely based on the x_i values from the previous step: for a k-th iteration:
  - first $x_1^{(k)}$ is calculated from $x_i^{(k-1)}$,
  - then $x_2^{(k)}$ is obtained from $x_i^{(k-1)}$, including the "old" $x_1^{(k-1)}$,
  - and so on, subsequent calculations within a given iteration only use the variables obtained in the previous iteration.

- Instead, it is possible to use the results from a "new" iteration as soon it these are available, that is:
  - to calculate $x_1^{(k)}$ from $x_i^{(k-1)}$,
  - then $x_2^{(k)}$ from $x_1^{(k)}$ and $x_{i\geqslant 3}^{(k-1)}$,
  - and so on, using every available "new" variable for all the subsequent calculations.

This is the idea behind a faster process, Gauss-Seidel iteration.

Gauss-Seidel iterations

With Gauss-Seidel iterations, we can use the most recent approximation as soon as we have obtained it:

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

Similarly to Jacobi iterations, Gauss-Seidel iterations converge to the true solution from any initial approximation $\mathbf{x}^0$, if the matrix of the system $\mathbf{A}$ is diagonally dominant.

Note: The condition that matrix $\mathbf{A}$ is diagonally dominant is a *sufficient* condition for convergence of the Gauss-Seidel and Jacobi algorithms, but not a *necessary* condition.

There are linear systems which are not diagonally dominant, but Gauss-Seidel iteration will still converge to the true solution, provided the starting approximation is sufficiently close to it.

Gauss-Seidel iterations: Example

The Jacobi iterative scheme for the previous example was

$$
\begin{aligned}
x_1^{(k+1)} &= \frac{1}{3}\left(5 - x_2^{(k)} - x_3^{(k)}\right) \\
x_2^{(k+1)} &= \frac{1}{6}\left(9 - 2x_1^{(k)} - x_3^{(k)}\right) \\
x_3^{(k+1)} &= \frac{1}{4}\left(6 - x_1^{(k)} - x_2^{(k)}\right)
\end{aligned}
$$

For Gauss-Seidel iterations, the scheme changes to

$$
\begin{aligned}
x_1^{(k+1)} &= \frac{1}{3}\left(5 - x_2^{(k)} - x_3^{(k)}\right) \\
x_2^{(k+1)} &= \frac{1}{6}\left(9 - 2x_1^{(k+1)} - x_3^{(k)}\right) \\
x_3^{(k+1)} &= \frac{1}{4}\left(6 - x_1^{(k+1)} - x_2^{(k+1)}\right)
\end{aligned}
$$

Gauss-Seidel iterations: Example

With this scheme, the calculations run as follows:

$$x_1^{(1)} = \frac{1}{3}(5 - 0 - 0) = \frac{5}{3}$$
$$x_2^{(1)} = \frac{1}{6}\left(9 - 2 \cdot \frac{5}{3} - 0\right) = \frac{17}{18}$$
$$x_3^{(1)} = \frac{1}{4}\left(6 - \frac{5}{3} - \frac{17}{18}\right) = \frac{61}{72}$$

With the next iteration we get

$$x_1^{(2)} = \frac{1}{3}\left(5 - \frac{17}{18} - \frac{61}{72}\right) = \frac{77}{72}$$
$$x_2^{(2)} = \frac{1}{6}\left(9 - 2 \cdot \frac{77}{72} - \frac{61}{72}\right) = \frac{433}{432}$$
$$x_3^{(2)} = \frac{1}{4}\left(6 - \frac{77}{72} - \frac{433}{432}\right) = \frac{1697}{1728}$$

Gauss-Seidel iterations: Example

After 7 iterations we eventually get:

$$x_1^{(7)} \approx 1.00000,\ x_2^{(7)} \approx 1.00000,\ x_3^{(7)} \approx 1.00000,$$

which satisfies the stopping criterion.

In fact, on the fourth iteration we already have

$$x_1^{(4)} \approx 1.0001,\ x_2^{(4)} \approx 1.0002,\ x_3^{(4)} \approx 0.99991.$$

Note that the convergence rates will differ for each variable.

- Generally, Gauss-Seidel method provides a much faster convergence than Jacobi iteration.

Relaxation methods

The idea behind relaxation methods is that instead of directly substituting the latest values obtained by an iterative method, the values are used according to certain **weighting**.

There are two main relaxation techniques.

- *Successive over-relaxation* (SOR)
- *Successive under-relaxation* (SUR)

"Successive over-relaxation" is a technique which is often used for numerical solution of partial differential equations.

Relaxation methods

- From the example with Gauss-Seidel iteration it was clear that the convergence efficiency is not uniform across the variables.
- At any given iteration, some of the values $x_i^{(k)}$ will be closer to the actual solution than some other ones.
- This suggests seeking a procedure which effectively "updates" some variables to a greater extent than the other ones.
- The idea behind relaxation methods is to introduce a parameter, called the relaxation parameter, which imposes a specific weighting on the variables.
- The algorithm is a generalisation of the Gauss-Seidel method.

Relaxation methods

Algorithm: Starting from an initial approximation $\mathbf{x}^0$, the approximations are updated according to the iterative scheme

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

An important distinction is that here $x_i^{(k+1)}$ also depends on $x_i^{(k)}$, whereas for the Gauss-Seidel scheme $x_i^{(k+1)}$ only depends on $x_{j \neq i}^{(k)}$.

- ω is the relaxation parameter. The method is called
  - successive under-relaxation (SUR) for $0 < \omega < 1$
  - successive over-relaxation (SOR) for $1 < \omega < 2$

  For $\omega = 1$ the algorithm reduces to Gauss-Seidel iteration.

  For $\omega > 2$ the algorithm diverges.

Relaxation methods

Algorithm: Starting from an initial approximation $\mathbf{x}^0$, the approximations are updated according to the iterative scheme

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

- There is an optimal value for ω, for which the algorithm converges towards the actual solution most rapidly.
- There are methods for calculating the optimal value, but these are extremely laborious and generally are not worth the effort.
- In practice, ω is selected through trials and errors.

Relaxation method: Example

Solve the following linear system, trying $\omega = 0.7$ and $\omega = 1.1$:

$$\begin{aligned} -5x_1 - x_2 + 2x_3 &= 1 \\ 2x_1 + 6x_2 - 3x_3 &= 2 \\ 2x_1 + x_2 + 7x_3 &= 32 \end{aligned}$$

As usual, we take $\mathbf{x}^{(0)} = \mathbf{0}$, and the relaxation scheme is

$$\begin{aligned} x_1^{(k+1)} &= x_1^{(k)} - \frac{\omega}{5}\left(1 + 5x_1^{(k)} + x_2^{(k)} - 2x_3^{(k)}\right) \\ x_2^{(k+1)} &= x_2^{(k)} + \frac{\omega}{6}\left(2 - 2x_1^{(k+1)} - 6x_2^{(k)} + 3x_3^{(k)}\right) \\ x_3^{(k+1)} &= x_3^{(k)} + \frac{\omega}{7}\left(32 - 2x_1^{(k+1)} - x_2^{(k+1)} - 7x_3^{(k)}\right) \end{aligned}$$

The exact solution is $x_1 = 1$, $x_2 = 2$, and $x_3 = 4$.

Relaxation method: Example

With $\omega = 0.7$ we get

$$x_1^{(1)} = 0 - \frac{0.7}{5}\big(1 + 0 + 0 - 0\big) = -0.14$$
$$x_2^{(1)} = 0 + \frac{0.7}{6}\big(2 - 2 \cdot (-0.14) - 6 \cdot 0 + 3 \cdot 0\big) = 0.266$$
$$x_3^{(1)} = 0 + \frac{0.7}{7}\big(32 - 2 \cdot (-0.14) - 0.266 - 7 \cdot 0\big) = 3.2014$$

$$x_1^{(2)} = -0.14 - \frac{0.7}{5} \cdot \big(1 + 5 \cdot (-0.14) + 0.266 - 2 \cdot 3.2014\big) = 0.677152$$
$$x_2^{(2)} = 0.266 + \frac{0.7}{6} \cdot \big(2 - 2 \cdot 0.677152 - 6 \cdot 0.266 + 3 \cdot 3.2014\big) = 1.27562$$
$$x_3^{(2)} = 3.2014 + \frac{0.7}{7} \cdot \big(32 - 2 \cdot 0.677152 - 1.27562 - 7 \cdot 3.2014\big) = 3.89743$$

The exact solution is $x_1 = 1$, $x_2 = 2$, and $x_3 = 4$.

Relaxation method

The process converges to a 10^{-4} precision after 11 iterations:

$$x_1^{(10)} = 1.00012 \qquad x_2^{(10)} = 1.99983 \qquad x_3^{(10)} = 3.99997$$

For $\omega = 1.1$ the process converges after 13 iterations:

$$x_1^{(12)} = 0.99995 \qquad x_2^{(12)} = 1.99998 \qquad x_3^{(12)} = 4.00002$$

The optimal value is $\omega \approx 0.88$, converging upon 8 iterations:

$$x_1^{(7)} = 1.00005 \qquad x_2^{(7)} = 1.99994 \qquad x_3^{(7)} = 3.99999$$

For $\omega > 1.4$ the relaxation method does not converge here.

The exact solution is $x_1 = 1$, $x_2 = 2$, and $x_3 = 4$.

Pivoting procedures

All numerical calculations are processed with a round-off error.

For an exaggerated example, assume a computer only keeps 3 significant digits, like $0.435 + 0.00132 = 0.436$.

Consider solving the following example with the above rounding:

$$\begin{cases} 0.0001x_1 & + & x_2 & = & 1, \\ x_1 & + & x_2 & = & 2. \end{cases}$$

The augmented matrix is

$$\begin{bmatrix} 0.0001 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

Row operation $\mathsf{R}_2 \to \mathsf{R}_2 - 10^4 \cdot \mathsf{R}_1$ yields

$$\begin{bmatrix} 0.0001 & 1 & 1 \\ 0 & -9999 & -9998 \end{bmatrix}$$

Pivoting procedures

From

$$\begin{bmatrix} 0.0001 & 1 & 1 \\ 0 & -9999 & -9998 \end{bmatrix}$$

the last equation is

$$-9999x_2 = -9998$$

so $x_2 = 9998/9999 = 0.99989998998\ldots$

However, because of the rounding the computer will give $x_2 = 1$.

The difference from the exact solution is only at the fourth digit.

But now, with backward substitution using the first equation,

$$0.0001x_1 + x_2 = 1$$

it would then obtain $x_1 = 0$, which is quite wrong:

the numbers $x_1 = 0$ and $x_2 = 1$ do not satisfy the initial system.

Pivoting procedures

Take the same system, but swap the rows first: $\mathsf{R}_1 \leftrightarrow \mathsf{R}_2$:

$$\left[\begin{array}{ccc} 1 & 1 & 2 \\ 0.0001 & 1 & 1 \end{array}\right]$$

Using the row operation $\mathsf{R}_2 \to 10^4\mathsf{R}_2 - \mathsf{R}_1$

$$\left[\begin{array}{ccc} 1 & 1 & 2 \\ 0 & 9999 & 9998 \end{array}\right]$$

From here, the computer result is again $x_2 = 9998/9999 \approx 1$.

Now, backward substitution using $x_1 + x_2 = 2$ leads to $x_1 = 1$,
very close to the exact $x_1 = 10000/9999 \approx 1.00010001\ldots$

The difference is on the fourth digit, now for each variable.

Pivoting procedures

By swapping the rows a "reasonably" accurate result was achieved

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

while the initial approach produced a grossly erroneous result

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Note that the exact solution is

$$\mathbf{x} = \begin{bmatrix} \frac{10000}{9999} \\ \\ \frac{9998}{9999} \end{bmatrix}$$

This type of swapping the rows is called *partial pivoting*.

Pivoting procedures

- At each step of Gaussian elimination, we need to compare pivots in a given column and swap the rows (if necessary) to *put the row with the largest pivot first*.
- This procedure is called *elimination with partial pivoting* and this is one of the basic algorithms of numerical linear algebra.
- In contrast to the *partial pivoting* strategy, with a *full pivoting* or *complete pivoting* strategy, both rows and columns are swapped (this leads to swapping of the order of unknowns).
- Full pivoting is more reliable but slower then partial pivoting. Partial pivoting is quite adequate for many applications.

Sensitive matrices

Consider two linear systems with the augmented matrices

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 1.0001 & 2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1.0001 & 2.0001 \end{bmatrix}$$

The solution of the first one is $x_1 = 2$ and $x_2 = 0$,
whereas the solution of the second is $x_1 = 1$ and $x_2 = 1$.

The slight change of the right hand side produced a substantial change to the solution.

- Some matrices are extremely sensitive to small changes. These matrices are called *ill-conditioned matrices*.
- The determinants of such matrices are usually either very small or very large (in this example, $\det = 10^{-4}$).
- Special "balancing" numerical techniques need to be applied to avoid numerical instabilities.
- Even balanced matrices can be sensitive to numerical methods (with the previous partial pivoting example, $\det = -0.9999$).

Next lecture

See you next Wednesday

10 April 2019

Assignment 2 is due this week

Assignment 3 is due next week