Numerical Linear Algebra

This document comprises of two parts:

- Detailed notes on numerical linear algebra as studied in the Further Numerical Methods lecture
- Module 3 assignment which is timetabled as Examination 2.

1 Introduction

This forms a vast topic as a branch of numerical analysis. It also ranks amongst the most important and frequently used. Here a brief overview is provided that is commensurate with Wilmott's notes as part of Further Numerical Methods.

Definition An **upper triangular** matrix U has zero elements below the principal diagonal, i.e.

$$u_{ij} = 0 \ \forall \ i > j.$$

For an upper triangular matrix U consider the system $U\underline{x} = \underline{b}$ in component form

$$\begin{aligned}
u_{11}x_1 + u_{12}x_2 + \ldots\ldots\ldots\ldots\ldots\ldots + u_{1n}x_n &= b_1 \\
u_{22}x_2 + \ldots\ldots\ldots\ldots\ldots\ldots + u_{2n}x_n &= b_2 \\
u_{33}x_3 + \ldots\ldots\ldots\ldots\ldots\ldots + u_{3n}x_n &= b_3 \\
\ddots \qquad\qquad\qquad\qquad\qquad\qquad \vdots &= \vdots \\
\ddots \qquad\qquad\qquad\qquad\qquad\qquad \vdots &= \vdots \\
\ddots \qquad\qquad\qquad\qquad\qquad\qquad \vdots &= \vdots \\
u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= b_{n-1} \\
u_{nn}x_n &= b_n
\end{aligned}$$

The determinant of the matrix U , can now easily be obtained from

$$\det(U) = \prod_{i=1}^{n} u_{ii}.$$

Recall that U has an inverse if $\det(U) \neq 0$, hence for U to be non-singular requires the strict condition $u_{ii} \neq 0 \ \ \forall \ i$. This linear system can now easily be solved by **back-substitution**. The algorithm for this scheme becomes

$$\begin{aligned}
x_n &= \frac{b_n}{u_{nn}} \\
x_i &= \frac{1}{u_{ii}}\left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j\right) \ \text{for } i = n-1, \ n-2, \ \ldots, 1
\end{aligned}$$

Definition A **lower triangular** matrix L has zero elements above the principal diagonal, i.e.

$$l_{ij} = 0 \ \ \forall \ i < j.$$

Now suppose L is a lower triangular matrix and consider the system $L\underline{x} = \underline{b}$.

A *lower triangular* matrix L has zero elements above the main diagonal, i.e.

$$l_{ij} = 0 \ \ \forall \ i < j.$$

Now suppose L is a lower triangular matrix and consider the system $L\underline{x} = \underline{b}$.

$$
\begin{array}{rcl}
l_{11}x_1 & = & b_1 \\
l_{21}x_1 + l_{22}x_2 & = & b_2 \\
\vdots \quad \vdots \quad \ddots & = & \vdots \\
\vdots \quad \vdots \quad \quad \ddots & = & \vdots \\
\vdots \quad \vdots \quad \quad \quad \ddots & = & \vdots \\
l_{n1}x_1 + l_{n2}x_2 + \ldots\ldots\ldots + l_{nn}x_n & = & b_n
\end{array}
$$

Again, because

$$\det(L) = \prod_{i=1}^{n} l_{ii},$$

L is non-singular iff $l_{ii} \neq 0 \ \forall\, i$. We solve this system by **forward-substitution**. The scheme is given by

$$
\begin{array}{rcl}
x_1 & = & \dfrac{b_1}{l_{11}} \\
x_i & = & \dfrac{1}{l_{ii}}\left(b_i - \displaystyle\sum_{j=1}^{i-1} l_{ij}x_j\right) \quad \text{for } \ i = 2,\ 3\ldots..,\ n
\end{array}
$$

Definition An $n \times n$ matrix is called a *band matrix* if integers p and q exist such that $p,\ q \in (1, n)$ with the property that $a_{ij} = 0$ whenever $i + p \leq j$ or $j + q \leq i$. The band width w of a banded matrix is defined to be $w = p + q - 1$.

So for example, the matrix

$$A = \begin{pmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & -5 & -6 \end{pmatrix}$$

is a banded matrix with $p = q = 2$ and band width $w = 3$.

So the definition of a banded matrix forces such matrices to have all its non-zero elements to be clustered close to the main diagonal.

p is then called the *upper band-width* and q the *lower band-width.*

Solving equations with banded matrices can be much more efficient than with full matrices because the triangular factors in the LU decomposition are also banded. This results in greater efficiency in terms of storage and amount of computations required. Banded matrices arise from the discretization of differential equations, because the finite difference method use to approximate derivatives only involve values at nearby mesh points.

From an applied mathematics perspective, the special case occurring when $p = q = 2$ and band width $w = 3$ is the most useful for us. These matrices are called *tridiagonal* since they have the form

$$
\begin{pmatrix}
a_{11} & a_{12} & 0 & \cdots & \cdots & 0 \\
a_{21} & a_{22} & a_{23} & \ddots & & \vdots \\
0 & a_{32} & a_{33} & a_{34} & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & & \ddots & \ddots & \ddots & a_{n-1,n} \\
0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn}
\end{pmatrix}
$$

2 The LU Decomposition

It is often advantageous to think of Gaussian elimination as constructing a lower tridiagonal matrix L and an upper triangular matrix U, so that $LU = A$. To illustrate this method consider the following *tridiagonal* linear system

$$\begin{pmatrix} b_0 & c_0 & 0 & \cdots & \cdots & 0 \\ a_1 & b_1 & c_1 & & & \vdots \\ 0 & a_2 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & \cdots & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ \vdots \\ p_{n-1} \\ p_n \end{pmatrix} \tag{1}$$

We can write this in the form

$$\mathbf{My} = \mathbf{p}$$

and then consider the matrix factorisation $\mathbf{M} = \mathbf{LU}$

$$= \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ l_1 & 1 & 0 & & & \vdots \\ 0 & l_2 & 1 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & l_{n-1} & 1 & 0 \\ 0 & \cdots & \cdots & 0 & l_n & 1 \end{pmatrix} \begin{pmatrix} d_0 & u_0 & 0 & \cdots & \cdots & 0 \\ 0 & d_1 & u_1 & & & \vdots \\ 0 & 0 & d_2 & u_2 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & 0 & d_{n-1} & u_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & d_n \end{pmatrix}$$

$$= \begin{pmatrix} d_0 & u_0 & 0 & \cdots & \cdots & 0 \\ l_1 d_0 & l_1 u_0 + d_1 & u_1 & & & \vdots \\ 0 & l_2 d_1 & l_2 u_1 + d_2 & u_2 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & l_{n-1} d_{n-2} & l_{n-1} u_{n-2} + d_{n-1} & u_{n-1} \\ 0 & \cdots & \cdots & 0 & l_n d_{n-1} & l_n u_{n-1} + d_n \end{pmatrix}$$

where L is the lower triangular matrix above and U is the upper triangular matrix above.

Equating the elements of the original tridiagonal matrix M with the elements of the product matrix LU, we find that

$$\begin{aligned} d_0 &= b_0 \\ u_i &= c_i, \quad i = 0, 1, 2, ..., n-1 \\ l_i &= \frac{a_i}{d_{i-1}}, \quad d_i = b_i - l_i u_{i-1}, \quad i = 1, 2, ..., n \end{aligned} \tag{2}$$

We then solve the original system given by (1) by solving two smaller problems. The tridiagonal system (1) is written in the form

$$\mathbf{My} = (\mathbf{LU})\,\mathbf{y} = \mathbf{L}\,(\mathbf{Uy}) = \mathbf{p}$$

We introduce an intermediate vector $\mathbf{z} = \mathbf{Uy}$ so that (1) becomes

$$\mathbf{Lz} = \mathbf{p}, \quad \mathbf{Uy} = \mathbf{z}.$$

First we solve the problem

$$\mathbf{Lz} = \mathbf{p}$$

for the intermediate vector $\mathbf{z}$, i.e. we solve the system

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ l_1 & 1 & 0 & & & \vdots \\ 0 & l_2 & 1 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & l_{n-1} & 1 & 0 \\ 0 & \cdots & \cdots & 0 & l_n & 1 \end{pmatrix} \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ \vdots \\ \vdots \\ z_{n-1} \\ z_n \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ \vdots \\ p_{n-1} \\ p_n \end{pmatrix} \tag{3}$$

Trivially, the z_i can be found by forward substitution so

$$z_0 = p_0, \quad z_i = p_i - l_i z_{i-1}, \quad i = 1, 2, ..., n$$

This determines the vector $\mathbf{z}$.

Having obtained $\mathbf{z}$ we find $\mathbf{y}$ by solving $\mathbf{Uy} = \mathbf{z}$;

$$\begin{pmatrix} d_0 & u_0 & 0 & \cdots & \cdots & 0 \\ 0 & d_1 & u_1 & & & \vdots \\ 0 & 0 & d_2 & u_2 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & 0 & d_{n-1} & u_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & d_n \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ \vdots \\ z_{n-1} \\ z_n \end{pmatrix} \tag{4}$$

The solution of (4) is trivially obtained by backward substitution

$$y_n = \frac{z_n}{d_n}, \quad y_i = \frac{z_i - u_i y_{i+1}}{d_i}, \quad i = n-1, n-2, ..., 2, 1$$

This gives us the solution, $\mathbf{y}$, of our original problem (1), $\mathbf{My} = \mathbf{p}$.

This method can be also extended to decompose non-sparse matrices, i.e. $A = LU$, thus opening up a wider class of associated methods.

$$= \begin{pmatrix} l_{11} & 0 & \cdots & \cdots & \cdots & 0 \\ l_{21} & l_{22} & 0 & & & \vdots \\ \vdots & l_{32} & & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & l_{n-1} & l_{n-1,n-1} & 0 \\ l_{n1} & l_{n2} & \cdots & \cdots & \cdots & l_{nn} \end{pmatrix} \times$$

$$\begin{pmatrix} u_{11} & u_{12} & \cdots & \cdots & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & \cdots & u_{2n} \\ 0 & 0 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & 0 & u_{n-1,n-1} & u_{n-1,n} \\ 0 & \cdots & \cdots & 0 & 0 & u_{nn} \end{pmatrix}$$

If $l_{ii} = 1 \ \forall \ 1 \leq i \leq n$ as mentioned in the earlier discussion, this ensures a unique solution and is called *Doolittle's method*. *Crout's method* requires $u_{ii} = 1 \ \forall \ 1 \leq i \leq n$.

The case where $l_{ii} = u_{ii} \ \forall \ 1 \leq i \leq n$ is known as *Cholesky's method*. This requires the matrix $\mathbf{A}$ to be symmetric and *positive definite*. Hence we can find an upper triangular matrix $\mathbf{U}$ such that $\mathbf{A} = \mathbf{U}^{\mathrm{T}}\mathbf{U}$

Example:

Consider the matrix $\begin{pmatrix} 2 & -1 & 1 \\ 3 & 3 & 9 \\ 3 & 3 & 5 \end{pmatrix}$ which can be factorised into

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1.5 & 1 & 0 \\ 1.5 & 1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & -1 & 1 \\ 0 & 4.5 & 7.5 \\ 0 & 3 & -4 \end{pmatrix}$$

3 Iterative Techniques

Seldom used for low dimensional problems - Gaussian elimination preferred.

For large systems with a high percentage of zero entries these are efficient in terms of

1. Computer Storage

2. Computational Time

Consider $(n \times n)$ linear system: $A\underline{x} = \underline{b}$

Initial approximation $\underline{x}^{(0)}$ to solution $\underline{x}$ which generates a sequence of vectors $\left\{\underline{x}^{(k)}\right\}_{k\,=\,0}^{\infty}$ that converges to $\underline{x}$.

Most techniques involve a process which converts $A\underline{x} = \underline{b}$ to

$$\underline{x} = T\underline{x} + \underline{c}$$

where T is an $(n \times n)$ matrix and $\underline{c}$ is a vector.

The initial vector $\underline{x}^{(0)}$ is selected and a sequence of approximations generated by computing

$$\underline{x}^{(k+1)} = T\underline{x}^{(k)} + \underline{c}$$

for each $k = 1,\ 2,\ 3,\ ....$

Now consider $A\underline{x} = \underline{b}$ with

$$A = \begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix}, \quad \underline{b} = \begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix}$$

$$\begin{array}{lcl} \mathbf{E}_1 & : & 10x_1 - x_2 + 2x_3 = 6 \\ \mathbf{E}_2 & : & -x_1 + 11x_2 - x_3 + 3x_4 = 25 \\ \mathbf{E}_3 & : & 2x_1 - x_2 + 10x_3 - x_4 = -11 \\ \mathbf{E}_4 & : & 3x_2 - x_3 + 8x_4 = 15 \end{array}$$

which has an exact solution

$$\underline{x} = \begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$$

To convert $A\underline{x} = \underline{b}$ to the form $\underline{x} = T\underline{x} + \underline{c}$; solve $\mathbf{E}_\mathrm{i}$ for each x_i $(i = 1, 2, 3, 4)$ to give

$$\left.\begin{array}{c} x_1^{(k+1)} = \frac{1}{10}\left\{x_2^{(k)} - 2x_3^{(k)} + 6\right\} \\ x_2^{(k+1)} = \frac{1}{11}\left\{x_1^{(k)} + x_3^{(k)} - 3x_4^{(k)} + 25\right\} \\ x_3^{(k+1)} = \frac{1}{10}\left\{-2x_1^{(k)} + x_2^{(k)} + x_4^{(k)} - 11\right\} \\ x_4^{(k+1)} = \frac{1}{8}\left\{-3x_2^{(k)} + x_3^{(k)} + 15\right\} \end{array}\right\} \qquad (5)$$

So (5) can be written as

$$\left.\begin{array}{c} x_1^{(k+1)} = \frac{1}{10}\left\{x_2^{(k)} - 2x_3^{(k)}\right\} + \frac{3}{5} \\ x_2^{(k+1)} = \frac{1}{11}\left\{x_1^{(k)} + x_3^{(k)} - 3x_4^{(k)}\right\} + \frac{25}{11} \\ x_3^{(k+1)} = \frac{1}{10}\left\{-2x_1^{(k)} + x_2^{(k)} + x_4^{(k)}\right\} - \frac{11}{10} \\ x_4^{(k+1)} = \frac{1}{8}\left\{-3x_2^{(k)} + x_3^{(k)}\right\} + \frac{15}{8} \end{array}\right\}$$

$\Rightarrow$

$$T = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{1}{5} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{1}{5} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \qquad \underline{c} = \begin{bmatrix} \frac{3}{5} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

For initial approximation let $\underline{x}^{(0)} = \underline{0}^{\mathrm{T}}$ and generate $\underline{x}^{(1)}$ by substituting $(0,\ 0,\ 0,\ 0)^{\mathrm{T}}$ in (12) to get

$$(0.6,\ 2.2727, -1.1000,\ 1.8750) = \underline{x}^{(1)}$$

and $\underline{x}^{(2)}$ can now be obtained.

In general obtain $\underline{x}^{(k)} = \left(x_1^{(k)},\ x_2^{(k)},\ x_3^{(k)},\ x_4^{(k)}\right)^{\mathrm{T}}$.

By the very nature of iterative techniques, these continue indefinitely, hence one requires a convergence criterion to terminate the computation once the imposed condition is satisfied. So we need to know when to stop iterating, i.e. does the sequence converge to the solution of the given system of equations? To answer this, we need a means of measuring the distance between $n-$ dimensional vectors.

4 Vector Norms

In two and three dimensions, the size of vectors called the modulus is generally obtained by Pythagoras. So if

$$\underline{v} = (a_1,\ a_2,\ a_3)$$

then the modulus of this vector $|\underline{v}| = \sqrt{(a_1^2+,\ a_2^2,\ a_3^2)}$ which gives us the distance from the origin.

Now consider $\underline{x} = (x_1,\ x_2, ..., x_n)^{\mathrm{T}}$. Let $\mathbb{R}^n$ denote the set of all $n-$ dimensional vectors (so all vector components are real). To define distance in $\mathbb{R}^n$ we use the notion of a *norm.* A pair of double vertical lines $\|\cdot\|$ is used to denote a norm, i.e. size of an $n-$ dimensional vector.

More formally a *vector norm* on $\mathbb{R}^n$ is a function, $\|\cdot\|$, such that

$$\|\cdot\| : \mathbb{R}^n \longrightarrow \mathbb{R}$$

with the following properties:

(i) $\|\underline{x}\| \geq 0 \quad \forall \underline{x} \in \mathbb{R}^n$

(ii) $\|\underline{x}\| = 0$ iff $\underline{x} = \underline{0}$

(iii) $\|\alpha\underline{x}\| = |\alpha|\,\|\underline{x}\| \quad \forall \alpha \in \mathbb{R}$ and $\underline{x} \in \mathbb{R}^n$

(iv) $\left\|\underline{x} + \underline{y}\right\| \leq \|\underline{x}\| + \left\|\underline{y}\right\| \quad \forall \underline{x},\ \underline{y} \in \mathbb{R}^n$ (Triangle inequality)

If $\underline{x} = (x_1,\ x_2, \ldots\ldots\ldots,\ x_n)^{\mathrm{T}}$, then the vector norm $\|\underline{x}\|_p$ for $p = 1, 2, \ldots$ is defined as

$$\|\underline{x}\|_p = \left\{ \sum_{i=1}^{n} |x_i|^p \right\}^{1/p}$$

Then there are a number of ways of defining a *norm*, depending on the value of p.

The special case $\|\underline{x}\|_\infty$ is given by

$$\|\underline{x}\|_\infty = \max_{1 \leq\ i\ \leq\ n} |x_i| \qquad\qquad l_\infty \text{ norm.}$$

The most commonly encountered vector norm (also known as the modulus of a vector) is $\|\underline{x}\|_2$ and defined by

$$\|\underline{x}\|_2 = \left\{ \sum_{i=1}^{n} x_i^2 \right\}^{1/2} \qquad\qquad l_2 \text{ norm}$$

For obvious reasons this is also called the *Euclidean norm.*

Examples

1. $\underline{x} = \left(\sin k,\ \cos k,\ 2^k\right)^{\mathrm{T}}$; k is a positive integer

$$\begin{aligned} \|\underline{x}\|_2 &= \sqrt{\sin^2 k + \cos^2 k + 4^k} = \sqrt{1 + 4^k} \\ \|\underline{x}\|_\infty &= \max\left|\sin k,\ \cos k,\ 2^k\right| = 2^k \end{aligned}$$

2. $\underline{x} = (2,\ 1,\ 3,\ -4)$

$$\|\underline{x}\|_2 = \sqrt{30} \qquad \text{and} \qquad \|\underline{x}\|_\infty = 4$$

These and other types of vector norms are further summarized in the following table, using a particular vector $\underline{v} = (1,\ 2,\ 3)$

Name	Symbol	value	numerical value
$l_1 - \text{norm}$	$\|\underline{x}\|_1$	6	6.000
$l_2 - \text{norm}$	$\|\underline{x}\|_2$	$\sqrt{14}$	3.742
$l_3 - \text{norm}$	$\|\underline{x}\|_3$	$6^{2/3}$	3.302
$l_4 - \text{norm}$	$\|\underline{x}\|_4$	$2^{1/4}\sqrt{7}$	3.146
$l_\infty - \text{norm}$	$\|\underline{x}\|_\infty$	3	3.00

If $\underline{x} = (x_1,\ x_2, \ldots\ldots\ldots,\ x_n)^{\mathrm{T}}$, $\underline{y} = (y_1,\ y_2, \ldots\ldots\ldots,\ y_n)^{\mathrm{T}} \in \mathbb{R}^n$, then the definitions for l_2 and l_∞ can be easily extended to consider distances between $\underline{x}$ and $\underline{y}$

$$\left\|\underline{x} - \underline{y}\right\|_2 = \left\{ \sum_{i=1}^{n} \left(x_i - y_i\right)^2 \right\}^{1/2}$$

$$\left\|\underline{x} - \underline{y}\right\|_\infty = \max_{1 \leq\ i\ \leq\ n} |x_i - y_i|$$

Having introduced the concept of a vector norm, we can now define the most appropriate form for our convergence criteria, by making use of the l_∞ norm. One such relative condition makes use of the infinity norm l_∞ as a test for convergence, where

$$\frac{\left\|\underline{x}^{(k+1)} - \underline{x}^{(k)}\right\|_\infty}{\left\|\underline{x}^{(k)}\right\|_\infty} < \varepsilon.$$

ε is the specified tolerance (> 0) and $\|\underline{x}\|_\infty = \max\limits_{1 \le i \le n} |x_i|$.

In the numerical example, actual computations yield for $k = 9$ & 10

$k = 9$	$k = 10$	
0.9997	1.0001	based on $\varepsilon = 10^{-3}$
2.0004	1.9998	
-1.0004	-0.9998	
1.0006	0.9998	

$$\begin{aligned}\frac{\left\|\underline{x}^{(10)} - \underline{x}^{(9)}\right\|_\infty}{\left\|\underline{x}^{(9)}\right\|_\infty} &= \frac{\left\|(4, -6, 6, -8)\, 10^{-4}\right\|_\infty}{2.0004} = \frac{8 \times 10^{-4}}{2.0004} \\ &= 3.9992 \times 10^{-4} < \varepsilon\end{aligned}$$

The iterative technique used is called the *JACOBI* method.

It consists of solving the i^{th} equation in $A\underline{x} = \underline{b}$ for x_i (provided $a_{ii} \neq 0$) to obtain

$$x_i = \sum_{j=1}^{n} \left(\frac{-a_{i\ j} x_j}{a_{ii}} \right) + \frac{b_i}{a_{ii}} \qquad i = 1, ..., n \quad ; \quad j \neq i$$

and generating each $x_i^{(k+1)}$ from components of $\underline{x}^{(k)}$ $(k \geq 1)$ by

$$x_i^{(k+1)} = \tfrac{1}{a_{ii}} \left[\sum_{j=1}^{n} \left(-a_{i\ j}\ x_j^{(k)} \right) + b_i \right],\ i = 1, ..., n;\ j \neq i$$

We can now refine this method very simply by using the most up-to-date x_i.

To compute $x_i^{(k+1)}$, components of $\underline{x}^{(k)}$ are used.

Since for $i > 1$, $x_1^{(k+1)}$, ..., $x_{i-1}^{(k+1)}$ have already been computed and are likely to be better approximations to the actual solutions x_1, ..., x_{i-1} than $x_1^{(k)}$, ..., $x_{i-1}^{(k)}$, it is far better to calculate $x_i^{(k+1)}$ using the most recently calculated values, i.e.

$$\begin{aligned} x_i^{(k+1)} &= \tfrac{1}{a_{ii}} \left[-\sum_{j=1}^{i-1} \left(a_{i\ j}\ x_j^{(k+1)} \right) - \sum_{j=i+1}^{n} \left(a_{i\ j}\ x_j^{(k)} \right) + b_i \right], \\ (i &= 1, ..., n) \end{aligned}$$

This is called the *GAUSS-SEIDEL* method.

Re-doing the previous example using the G-S method iterative technique:

$$\begin{aligned} 10x_1 - x_2 + 2x_3 &= 6 \\ -x_1 + 11x_2 - x_3 + 3x_4 &= 25 \\ 2x_1 - x_2 + 10x_3 - x_4 &= -11 \\ 3x_2 - x_3 + 8x_4 &= 15 \end{aligned}$$

which is written as

$$\left.\begin{aligned} x_1^{(k+1)} &= \tfrac{1}{10}\left\{x_2^{(k)} - 2x_3^{(k)} + 6\right\} \\ x_2^{(k+1)} &= \tfrac{1}{11}\left\{x_1^{(k+1)} + x_3^{(k)} - 3x_4^{(k)} + 25\right\} \\ x_3^{(k+1)} &= \tfrac{1}{10}\left\{-2x_1^{(k+1)} + x_2^{(k+1)} + x_4^{(k)} - 11\right\} \\ x_4^{(k+1)} &= \tfrac{1}{8}\left\{-3x_2^{(k+1)} + x_3^{(k+1)} + 15\right\} \end{aligned}\right\}$$

Again letting $\underline{x}^{(0)} = \underline{0}^{\mathrm{T}}$

Here

$$\begin{aligned} \underline{x}^{(4)} &= (1.0009,\ 2.0003,\ -1.0003,\ 0.9999) \\ \underline{x}^{(5)} &= (1.0001,\ 2.0000,\ -1.0000,\ 1.0000) \end{aligned}$$

$$\frac{\left\|\underline{x}^{(5)} - \underline{x}^{(4)}\right\|_\infty}{\left\|\underline{x}^{(4)}\right\|_\infty} = 4 \times 10^{-4} \quad (< \varepsilon)$$

We note in this example that Jacobi's method required twice as many iterations for the same level of accuracy.

If A is strictly diagonally dominant than for any choice of $\underline{x}^{(0)}$ the sequence of solutions generated by both Gauss-Seidel and Jacobi converge to the unique solution.

5 Successive-Over-Relaxation (SOR) Methods

A large part of numerical analysis is based upon the need to improve the efficiency and speed of existing techniques. Although we have seen from the Gauss-Seidel method that it is superior to Jacobi in this regard, we can nevertheless continue to look for improvements in convergence speed. In this section we will briefly look at a method called *successive-over-relaxation.*

Let $\widehat{\underline{x}} \in \mathbb{R}^n$ be an approximation to the actual solution for $A\underline{x} = \underline{b}$.

The *residual vector* $\underline{r}$ for $\widehat{\underline{x}}$ is

$$\underline{r} = \underline{b} - A\widehat{\underline{x}}$$

In Jacobi/Gauss-Seidel methods a residual vector is associated with each calculation of an approximation component to the solution vector.

Aim: Generate a sequence of approximations that cause the associated residual vectors to converge rapidly to zero. Then methods involving equations of the form

$$x_i^{(k+1)} = x_i^{(k)} + \omega \frac{r_{ii}^{(k+1)}}{a_{ii}}$$

are called **relaxation methods** and for certain choices of positive ω lead to faster convergence.

For $0 < \omega < 1$, the procedures are called **under-relaxation methods** and used for convergence of systems which do not converge by Gauss-Seidel.

For $\omega > 1$, the procedures are called **over-relaxation methods** which are used to accelerate convergence of systems which are convergent by Gauss-Seidel.

Scheme: For Gauss-Seidel we have

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

for $(i = 1, ..., n)$.

How do we choose the value of ω?

Firstly we introduce a few relevant ideas.

Suppose the matrix A can be written in the form

$$A = D - L - U$$

where D is a matrix with the diagonal elements of A (zero everywhere else).

L is a strictly lower-triangular part of A.

U is strictly upper part of A

Then for the Jacobi method we have

$$T_j = D^{-1}(L+U)$$

and for Gauss-Seidel

$$T_g = (D-L)^{-1}U$$

Example: Consider

$$A = \begin{pmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}$$

So

$$D = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix} \rightarrow D^{-1} = \frac{1}{4}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$T_j = D^{-1}(L+U) = D^{-1}(D-A)$

$$= \begin{pmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix}\begin{pmatrix} 0 & -3 & 0 \\ -3 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{3}{4} & 0 \\ -\frac{3}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 \end{pmatrix}$$

Given an $n \times n$ matrix A the *spectral radius* $\rho(A)$ is defined as

$$\rho(A) = \max|\lambda_i| \quad i = 1, 2, \ ....., n$$

where λ is the eigenvalue of A.

Note: If $\lambda = \alpha + i\beta$ then $|\lambda| = \sqrt{\alpha^2 + \beta^2}$.

In response to the earlier question regarding the choice of ω, there is no complete answer. However there are a few well known results (theorems) which can be used in certain situations.

If A is a positive definite tridiagonal matrix then the optimal choice for ω is

$$\begin{aligned} \omega &= \frac{2}{1+\sqrt{1-\rho\left(T_g\right)}} \\ &= \frac{2}{1+\sqrt{1-\rho^2\left(T_j\right)}} \end{aligned}$$

With this choice of ω, $\rho\left(T_\omega\right) = \omega - 1$.

The earlier example gave

$$T_j = \begin{pmatrix} 0 & -\frac{3}{4} & 0 \\ -\frac{3}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 \end{pmatrix}$$

so we have

$$\begin{aligned} T_j - \lambda I &= \begin{pmatrix} -\lambda & -\frac{3}{4} & 0 \\ -\frac{3}{4} & -\lambda & \frac{1}{4} \\ 0 & \frac{1}{4} & -\lambda \end{pmatrix} \rightarrow \\ |T_j - \lambda I| &= -\lambda\left(\lambda^2 - \tfrac{5}{8}\right) \end{aligned}$$

Thus $\rho\left(T_j\right) = \sqrt{0.625}$ and

$$\begin{aligned} \omega_j &= \frac{2}{1+\sqrt{1-\rho^2\left(T_j\right)}} = \frac{2}{1+\sqrt{1-0.625}} \approx 1.24 \\ \omega_g &= \frac{2}{1+\sqrt{1-\rho\left(T_g\right)}} \end{aligned}$$

Another theorem states:

If A is positive definite and $0 < \omega < 2$, then the SOR method converges for any starting guess $\underline{x}^{(0)}$.

6 The Theory

Earlier the treatment was casual with more reliance on numerical examples. Here we use a more formal approach to discuss iterative techniques.

Recalling from linear algebra, a matrix $\mathbf{M}$ with entries $\mathbf{M}_{ij}$ is *strictly diagonally dominant* if

$$|\mathbf{M}_{ii}| > \sum_{j \neq i} |\mathbf{M}_{ij}|,$$

So in this case

$$|b_n| > |a_n| + |c_n|$$

and hence the scheme will converge to the solution of the original problem,

$$\lim_{k \to \infty} \widehat{v}_n^{(k)} = v_n$$

Suppose the matrix $\mathbf{M}$ (given by (1) can be written in the form

$$\mathbf{M} = L + D + U$$

where D is a matrix with the diagonal elements of $\mathbf{M}$ (zero everywhere else).

L is a strictly lower-triangular part of $\mathbf{M}$.

U is strictly upper part of $\mathbf{M}$

$$\underbrace{\begin{pmatrix} b_0 & c_0 & 0 & \cdots \\ a_1 & b_1 & c_1 & \\ 0 & a_2 & \ddots & \ddots \\ \vdots & & \ddots & \ddots \end{pmatrix}}_{\mathbf{M}} =$$

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 & \cdots \\ a_1 & 0 & 0 & \\ 0 & a_2 & 0 & \ddots \\ \vdots & & \ddots & \ddots \end{pmatrix}}_{\mathbf{A}} + \underbrace{\begin{pmatrix} b_0 & 0 & 0 & \cdots \\ 0 & b_1 & 0 & \\ 0 & 0 & b_2 & \ddots \\ \vdots & & \ddots & \ddots \end{pmatrix}}_{\mathbf{B}} + \underbrace{\begin{pmatrix} 0 & c_0 & 0 & \cdots \\ 0 & 0 & c_1 & \\ 0 & 0 & \ddots & \ddots \\ \vdots & & \ddots & \ddots \end{pmatrix}}_{\mathbf{C}}$$

The scheme now becomes

$$\widehat{\mathbf{v}}^{(k+1)} = \mathbf{B}^{-1}\mathbf{w} - \mathbf{B}^{-1}\left(\mathbf{A} + \mathbf{C}\right)\widehat{\mathbf{v}}^{(k)}.$$

$\mathbf{B}^{-1}\left(\mathbf{A} + \mathbf{C}\right)$ is another matrix which premultiplies the iteration vector, call this $\mathbf{IM}$, to write

$$\widehat{\mathbf{v}}^{(k+1)} = \mathbf{B}^{-1}\mathbf{w} - \mathbf{IM}\widehat{\mathbf{v}}^{(k)}$$

Given an $N \times N$ matrix $\mathbf{M}$ the *spectral radius* $\rho\left(\mathbf{M}\right)$ is defined as

$$\rho\left(\mathbf{M}\right) = \max\left|\lambda_i\right| \quad i = 1, 2, \ldots\ldots, n$$

where λ is the eigenvalue of $\mathbf{M}$.

Note: If $\lambda = \alpha + i\beta$ then $|\lambda| = \sqrt{\alpha^2 + \beta^2}$

Convergence of the Jacobi method is proved theoretically by showing that the spectral radius of the iteration matrix $\mathbf{IM}$ <1.

The Gauss Seidel Method

We can now refine the Jacobi method very simply by using the most up-to-date $\widehat{v}^{k+1}$ available to us, by writing

$$\widehat{v}_n^{(k+1)} = \frac{1}{b_n}\left(w_n - a_n\widehat{v}_{n-1}^{(k+1)} - c_n\widehat{v}_{n+1}^{k}\right).$$

Note that at iteration step $(k+1)$, we are using the most recent $\widehat{v}_{n-1}^{(k+1)}$, instead of $\widehat{v}_{n-1}^{(k)}$ which is used in Jacobi. Again, iteration convergence is guaranteed if the coefficient matrix is strictly diagonally dominant. Write the above as

$$b_n\widehat{v}_n^{(k+1)} + a_n\widehat{v}_{n-1}^{(k+1)} = w_n - c_n\widehat{v}_{n+1}^{k}$$

which as a matrix problem becomes

$$\left(\mathbf{A} + \mathbf{B}\right)\widehat{\mathbf{v}}^{(k+1)} = \mathbf{w} - \mathbf{C}\widehat{\mathbf{v}}^{(k)}.$$

Write

$$\mathbf{Q} = (\mathbf{A} + \mathbf{B})^{-1}, \ \mathbf{IM} = \mathbf{Q}^{-1}\mathbf{C}$$

so our iteration problem becomes

$$\widehat{\mathbf{v}}^{(k+1)} = \mathbf{Q}^{-1}\mathbf{w} - \mathbf{IM}\widehat{\mathbf{v}}^{(k)}.$$

As before convergence of this GS method is proved theoretically by showing that the spectral radius of the iteration matrix **IM** <1, but we will not be implementing in terms of the iteration matrix.

Successive-Over-Relaxation (SOR) Methods

SOR can be applied to either the Jacobi or the Gauss-Seidel method.

The sequence of approximate solutions $\widehat{v}_n^{(k)}$ such that $\lim_{k\to\infty} \widehat{v}_n^{(k)} \longrightarrow v_n$, can be speeded up. This is the essence of SOR. To see how this works, begin by writing

$$\widehat{v}_n^{(k+1)} = \widehat{v}_n^{(k)} + \left(\widehat{v}_n^{(k+1)} - \widehat{v}_n^{(k)}\right).$$

Of interest is the term $\left(\widehat{v}_n^{(k+1)} - \widehat{v}_n^{(k)}\right)$ which can be regarded as a correction to be added to $\widehat{v}_n^{(k)}$ in order for it to get closer to the exact solution $\widehat{v}_n^{(k)}$. Introduce ω to control the magnitude of this correction term

$$\widehat{x}_n^{(k+1)} = \widehat{v}_n^{(k)} + \omega\left(\widehat{v}_n^{(k+1)} - \widehat{v}_n^{(k)}\right).$$

So with careful choice of ω we hope that $\widehat{x}_n^{(k+1)}$ will be a better approximation to v_n than $\widehat{v}_n^{(k+1)}$.

SOR correction for **Jacobi**

$$\begin{aligned} \widehat{u}_n^{(k)} &= \frac{1}{b_n}\left(w_n - a_n\widehat{v}_{n-1}^{(k)} - c_n\widehat{v}_{n+1}^{k}\right) \\ \widehat{v}_n^{(k+1)} &= \widehat{v}_n^{(k)} + \omega_{\mathbf{J}}\left(\widehat{u}_n^{(k)} - \widehat{v}_n^{(k)}\right) \end{aligned}$$

SOR correction for **GS**

$$\begin{aligned} \widehat{u}_n^{(k)} &= \frac{1}{b_n}\left(w_n - a_n\widehat{v}_{n-1}^{(k+1)} - c_n\widehat{v}_{n+1}^{k}\right) \\ \widehat{v}_n^{(k+1)} &= \widehat{v}_n^{(k)} + \omega_{\mathbf{GS}}\left(\widehat{u}_n^{(k)} - \widehat{v}_n^{(k)}\right) \end{aligned}$$

Assignment for Module 3

September 2020

This is a numerical methods based computational project on the use of direct and indirect methods to solve a linear system. It should be completed in Python.

Consider the linear system

$$A\mathbf{x} = \mathbf{b} \text{ such that } A = \begin{bmatrix} 0 & 3 & -1 & 8 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 10 & -1 & 2 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 15 \\ 25 \\ -11 \\ 6 \end{bmatrix} \tag{*}$$

where $\mathbf{x} = (x_1, x_2, x_3, x_4)^{\mathrm{T}}$.

1. Write a program to test if the matrix A is **strictly diagonally dominant**.
2. LU **Decomposition**: Write a program to obtain a solution of $(*)$ using both methods of Doolittle and Crout. Your program in each case should list L and U, followed by the solution for $\mathbf{x}$
3. **Gauss-Seidel Method**: Write a program to solve $(*)$. Use an initial guess of $\mathbf{x}^{(0)} = (0, 0, 0, 0)$ and then repeat with $\mathbf{x}^{(0)} = (1, 1, 1, 1)$; examine if one converges quicker than the other with consistency to four decimal places. At each iteration step, the solution vector should be printed together with use of the l_∞ norm.
4. **SOR:** Using an acceleration factor of $\omega = 1.1$, repeat part 3. above for $\mathbf{x}^{(0)} = (0, 0, 0, 0)$.
5. Include a report which discusses your results, together with observations of accuracy and **computational efficiency**.

Note: project/course-work type assessments are graded differently to mathematical exercises. The mechanical process of assessing maths based work where one can score 100% is different to the way a project is marked. So e.g. a 80% score in such a study would be deemed outstanding while 90% is exceptional.

The instructions above if followed carefully can lead to 80%. The ability and willingness to experiment and demonstrate initiative, beyond this, can attain a higher score.