Solving linear systems

The problem: given $\underline{b} \in \mathbb{R}^n$, and $A \in \mathbb{R}^n \times \mathbb{R}^n$, we look for $\underline{x} \in \mathbb{R}^n$ solution of

$$A\underline{x} = \underline{b} \tag{1}$$

Problem (1) has a unique solution *if and only if* the matrix A is non-singular (or invertible), i.e., $\exists A^{-1}$ such that $A^{-1}A = AA^{-1} = I$; necessary and sufficient condition for A being invertible is that $det(A) \neq 0$. Then the solution $\underline{x}$ is formally given by $\underline{x} = A^{-1}\underline{b}$.

Beware: never invert a matrix unless really necessary, due to the costs, as we shall see later on. (Solving a system with a general full matrix is also expensive, but not nearly as expensive as matrix inversion).

Some example of linear systems

The simplest systems to deal with are diagonal systems:

$$D\underline{x} = \underline{b}$$

$$D = \begin{bmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \cdots & 0 \\ 0 & \cdots & \ddots & 0 \\ 0 & \cdots & & d_{nn} \end{bmatrix} \longrightarrow \; x_i = \frac{b_i}{d_{ii}} \quad i = 1, 2, \cdots, n$$

The cost in terms of number of operations is negligible, given by n division.

Lower Triangular matrices

Triangular matrices are also easy to handle. If $A = L$ is lower triangular, the system can be solved "forward":

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \longrightarrow \begin{cases} x_1 = \dfrac{b_1}{l_{11}} \\ x_2 = \dfrac{b_2 - l_{21}x_1}{l_{22}} \\ \vdots \\ x_n = \dfrac{b_n - \sum_{j=1}^{n-1} l_{nj}x_j}{l_{nn}} \end{cases}$$

Counting the operations: for x_1 we have 1 product and 0 sums; for x_2 2 products and 1 sum, ..., and for x_n n products and $n-1$ sums, that is, $1 + 2 + \cdot + n = \dfrac{n(n+1)}{2}$ products, plus $1 + 2 + \cdots + n - 1 = \dfrac{(n-1)n}{2}$ sums, for a total number of operations $= n^2$.

Forward Substitution

What follows is an algorithm, for solving $L\underline{x} = \underline{b}$, with L lower triangular.

forward substitution

Input: $L \in \mathbb{R}^{n\times n}$, lower t., and $b \in \mathbb{R}^n$

for $i = 1, \dots, n$

  for $j = 1, \dots, i-1$

    $b_i = b_i - l_{i,j} b_j$

  end

  $b_i = \frac{b_i}{l_{ii}}$

end

the solution is in $\underline{b}$, i.e., $\underline{x} \leftarrow \underline{b}$

Homework

Write the above algorithm in MATLAB

A special case of lower triangular matrix (useful later...)

Assume L has only 1 on the diagonal, e.g., $l_{ii} = 1$:

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \longrightarrow \begin{cases} x_1 = b_1 \\ x_2 = b_2 - l_{21}x_1 \\ \vdots \\ x_n = b_n - \displaystyle\sum_{j=1}^{n-1} l_{nj}x_j \end{cases}$$

Then the following two algorithms, for solving $L\underline{x} = \underline{b}$ are equivalent:

forward substitution as above

Input: $L \in \mathbb{R}^{n\times n}$, lower t., and $b \in \mathbb{R}^n$
for $i = 1, \ldots, n$
    for $j = 1, \ldots, i-1$
        $b_i = b_i - l_{i,j}b_j$
    end
end
the solution is in $\underline{b}$, i.e., $\underline{x} \leftarrow \underline{b}$

equivalent to forward substitution

Input: $L \in \mathbb{R}^{n\times n}$, lower t., and $b \in \mathbb{R}^n$
for $k = 1, \ldots, n-1$
    for $i = k+1, \ldots, n$
        $b_i = b_i - l_{i,k}b_k$
    end
end
the solution is in $\underline{b}$, i.e., $\underline{x} \leftarrow \underline{b}$

Upper Triangular matrices

Upper triangular systems (if $A = U$ is upper triangular) are also easy to deal with, and can be solved "backward" with the same costs as lower triangular systems:

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & & \cdots & u_{nn} \end{bmatrix} \longrightarrow \begin{cases} x_n = \dfrac{b_n}{u_{nn}} \\ x_{n-1} = \dfrac{b_{n-1} - u_{n-1n}x_n}{u_{n-1n-1}} \\ \vdots \\ x_1 = \dfrac{b_1 - \sum_{j=2}^{n} u_{1j}x_j}{u_{11}} \end{cases}$$

Backward Substitution

What follows is an algorithm, for solving $U\underline{x} = \underline{b}$, with U upper triangular.

backward substitution

Input: $L \in \mathbb{R}^{n \times n}$, upper t., and $b \in \mathbb{R}^n$

for $i = n, \ldots, 1$

  for $j = i+1, \ldots, n$

    $b_i = b_i - u_{i,j} b_j$

  end

  $b_i = \frac{b_i}{u_{ii}}$

end

the solution is in $\underline{b}$, i.e., $\underline{x} \leftarrow \underline{b}$

Homework

Write the above algorithm in MATLAB

Costs for solving triangular systems

We saw that for solving the system we perform n^2 operations.

To have an idea of the time necessary to solve a triangular system, suppose that the number of equations is $n = 100.000$, and the computer performance is a TERAFLOP $= 10^{12} FLOPS$ (floating-point operations per second); the time in seconds is given by

$$t = \frac{\#operations}{\#flops} = \frac{10^{10} ops}{10^{12} flops} = \frac{1}{100} sec.$$

(pretty quick)

The next target of High Performance Computing is EXAFLOP $= 10^{18} FLOPS$