CHAPTER 2 Numerical methods for linear Algebra

- There are a variety of linear algebra problems one may encounter, for example
  - solutions of well-posed linear systems
    $Ax = b$ A square matrix
  - approximate solutions of other linear systems
    $Ax = b$ A non-square
    (cf. least-square fitting pbs)
  - eigenvalue/eigenvector problems
    $Ax = \lambda x$

- Moreover those can occur/arise as exact problems, or as a result of the discretization of a continuous problem (cf PDES/ODES) pb.

⇒ LEARNING STABLE, ACCURATE, FAST & EFFICIENT ALGORITHMS FOR LA WILL BE A FUNDAMENTAL RESOURCE FOR ITS OWN SAKE, AS WELL AS FOR THE SAKE OF STUDYING ODES & PDES.

The aim will be for you to develop a set of useful routines, as well as to learn how to use the LAPACK routines, for solving a wide range of LA problems.

I Matrix Equation $Ax = B$; direct methods

The goal of this section is to learn how to solve $Ax = B$ with A a square matrix.

① Gaussian Elimination

The standard algorithm learned in basic LA classes is the Gaussian elimination: given the matrix & vector

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & & & \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} L_1 \\ \vdots \\ L_n \end{pmatrix} \qquad B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

lines L_1 to L_n

Step 1: transform the matrix into upper-triangular form & modify B accordingly

$$\text{start}\begin{pmatrix} x&x&x&x&x\\ x&x&x&x&x\\ x&x&x&x&x\\ x&x&x&x&x\\ x&x&x&x&x \end{pmatrix} \to \begin{pmatrix} x&x&x&x&x\\ 0&x&x&x&x\\ 0&x&x&x&x\\ 0&x&x&x&x\\ 0&x&x&x&x \end{pmatrix} \to \begin{pmatrix} x&x&x&x&x\\ 0&x&x&x&x\\ 0&0&x&x&x\\ 0&0&x&x&x\\ 0&0&x&x&x \end{pmatrix}$$

$$\rightsquigarrow \begin{pmatrix} x&x&x&x&x\\ 0&x&x&x&x\\ 0&0&x&x&x\\ 0&0&0&x&x\\ 0&0&0&0&x \end{pmatrix}\text{final}$$

$\Rightarrow$ Algorithm is

for $j = 1$ to $n-1$ ← sweep over columns

  for $i = j+1$ to n ← sweep over lines

    $L_i = L_i - \frac{a_{ij} L_j}{a_{jj}}$ ← zeros out coefficient a_{ij}

    $b_i = b_i - \frac{a_{ij} L_j}{a_{jj}}$ ← carries out same in RHS.

Step 2 : Backsubstitute for x :

for $i = n$ to 1

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

Note : • in this formulation, it is very common to return x into the input b-vector so that the lines

for $i = n$ to 1

$$b_i = \frac{1}{a_{ii}}\left[b_i - \sum_{j=i+1}^{n} a_{ij} b_j \right]$$

works just as well

• It is quite obvious that if one of the diagonal elements is null, then the algorithm fails at Step 1. This is easily prevented by checking for these elements ahead of time, and swapping lines as necessary (with equiv. operation on the B-vector).

e.g. if at some step we have

$$\begin{pmatrix} x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & x & x & x & x \end{pmatrix}$$

problem as we are about to manipulate this column

→ Simply switch line 3 with another line below since switching the order of lines within a linear system (matrix + RHS) doesn't change the problem.

• If fact, we may want to switch lines even if the diagonal element is not exactly 0, but very small, to avoid overflow problems.

② Gaussian elimination with implicit pivoting

In fact, having small diagonal elements is not only a problem for overflow, but also for truncation errors → $\frac{1}{\text{small}}$ = large, so large truncation errors can occur, resulting in unstable algorithms.

Extreme example:
$$\begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

→ the lower line would be modified by the Gaussian Elimination algorithm to $L_2 = L_2 - \frac{1 . L_1}{10^{-20}}$

$$\begin{pmatrix} 10^{-20} & 1 \\ 0 & 1-10^{20} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 2-10^{20} \end{pmatrix}$$

in FP arithmetic ⇒
$$\begin{pmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ -10^{20} \end{pmatrix}$$

so then
$$y \simeq \frac{-10^{20}}{-10^{20}} = 1$$
$$x = \frac{(1-1)}{10^{-20}} \simeq 0$$

But the real answer is
$$\begin{cases} y = \dfrac{1-2\,10^{-20}}{1-10^{-20}} \simeq 1 \\[2ex] x = \dfrac{1}{1-10^{-20}} \simeq 1 \end{cases}$$

Meanwhile, if we had switched the lines of the original matrix around:

$$\begin{pmatrix} 1 & 1 \\ 10^{-20} & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

the FP arithmetic with Gaussian elimination yields

$$\begin{pmatrix} 1 & 1 \\ 0 & 1-10^{-20} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1-2\,10^{-20} \end{pmatrix}$$

$\Rightarrow$ in FP

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$\Rightarrow \begin{cases} y = 1 \\ x = 1 \end{cases}$$ as the true result is too.

<u>Conclusion</u>

$\Rightarrow$ Keeping <u>large</u> diagonal elements helps. This is the idea of <u>pivoting</u>

<u>Gaussian elimination with partial pivoting</u> is similar to Gaussian Elimination, but at each step k, we first search for the largest <u>pivot</u>, (i.e. the largest coefficient a_{ik} for $k \le i \le N$), then switch the line containing the diagonal with the one containing the pivot, and then proceed.

$$\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{pmatrix}$$

$\Rightarrow$ At step k, seek largest coefficient $\{a_{ik}\}_{i \ge k}$

$$\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{pmatrix}$$

switch line with pivot with line k, then continue on. (<u>note</u>: of course the RHS must be also modified).

An oddity: The system doesn't change if one of the equations is multiplied by a large number, but partial pivoting is then guaranteed to select this equation for a pivot early on

$\Rightarrow$ So consider **implicit pivoting** where each equation is first scaled by its largest coefficient, then the Gaussian Elimination is performed on the scaled matrix ... or not quite

Algorithm Given a matrix $a(i,j)$ and RHS $b(i)$ (dimensions $n \times n$ and n)

① For $i = 1$ to n
- Find the largest value $|a(i,j)|_{j=1..n}$ and store it as scale(i)

② For $j = 1$ to n
- Find $p = \max \left| \frac{a(k,j)}{\text{scale}(k)} \right|_{k=j \ldots n}$
  (i.e. find the largest scaled pivot)
- Swap line containing p with line j (incl. RHS)
- For $i = j+1, n$
  - $L_i = L_i - \frac{a_{ij} L_j}{a_{jj}}$ (including RHS)

③ Backsubstitute.

$\Rightarrow$ This performs partial implicit pivoting of the matrix A & RHS B to solve the system $Ax = B$

Note **Full** pivoting also exists.

③ Gauss-Jordan elimination & calculation of the inverse

GE yields an upper-triangular matrix, which must then be back-substituted. Imagine instead the following algorithm.

GJ Algorithm

① For $i = 1, n$

$$\text{scale}(i) = \max |a(i,j)|_{j=1,n}$$

② For $j = 1, n$

- $p = \max \left| \dfrac{a(k,j)}{\text{scale}(k)} \right| \quad k = j, n$
- swap line with p with line j (including RHS)
- divide (new) line j by p (including RHS)
- For $i = 1, n$ $(i \neq j)$

  $L_i = L_i - a_{ij} L_j$ (including RHS)

This effectively eliminates all off-diagonal components and transforms A to the unit matrix:

$$\begin{pmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{pmatrix} \rightarrow \begin{pmatrix} 1 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & x & x \\ 0 & 1 & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{pmatrix}$$

$$\cdots \rightsquigarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and the RHS b to the solution of the linear system.

(Mathematically, the sequence of linear ops performed upon A & B corresponds to multiplying both sides by A^{-1}).

- Note now that the same operations can easily be performed simultaneously on <u>many</u> RHS vectors.

$$AX = \begin{pmatrix} | & | & & | \\ b_1 & b_2 & \cdots & b_m \\ | & | & & | \end{pmatrix}$$

$$\hookrightarrow \quad A^{-1}AX = IX = \begin{pmatrix} | & | & | & & | \\ A^{-1}b_1 & A^{-1}b_2 & A^{-1}b_3 & \cdots & A^{-1}b_m \\ | & | & | & & | \end{pmatrix}$$

- So if the collection of vectors $(b_1, \ldots b_n)$ is actually the identity matrix initially, then upon exit B becomes the inverse of A.

$$AX = I$$

$$\Rightarrow \quad A^{-1}AX = A^{-1}I = A^{-1}$$

$$\Rightarrow \quad X = A^{-1}$$

So to obtain the inverse matrix of A, a sturdy method consists in performing GJ elimination with $B = I$.

<u>Homework</u>: Write a program to calculate the inverse of an $n \times n$ matrix using GJ eliminatio
The matrix should be read from a file.
and the result should be written to a file