Linear Systems

Matrix Operations and Gauss Elimination

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.

References

- S. Chapra and R. Canale, Numerical Methods for Engineers
- A. Greenbaum and T. Chartier, Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms
- M. Heath, Scientific Computing: An Introductory Survey
- C. Moler, Numerical Computing with MATLAB
- wikipedia.org

Linear Systems of Equation

- We saw how to solve a single non-linear equation in one variable $f(x) = 0$
- We now consider solving a *system* of *linear* equations
- For example consider a system of 2 equations in 2 unknowns x_1 and x_2 looks like:

$$\begin{aligned} 2x_1 + x_2 &= 1 \\ x_1 + x_2 &= 0 \end{aligned}$$

  which has the solution $x_1 = 1$, and $x_2 = -1$.
- In matrix form, we can write this system as:

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Linear Systems of Equation

A general system of m equations in n unknowns x_i:

$$\begin{aligned}
a_{11}x_1 + \;& a_{12}x_2 + \cdots + \; a_{1n}x_n = b_1 \\
a_{21}x_1 + \;& a_{22}x_2 + \cdots + \; a_{2n}x_n = b_2 \\
\vdots \quad\;\; & \;\;\vdots \qquad\qquad\quad \vdots \qquad \vdots \\
a_{m1}x_1 + \;& a_{m2}x_2 + \cdots + a_{mn}x_n = b_m.
\end{aligned}$$

In matrix notation: $\mathbf{A}\mathbf{x} = \mathbf{b}$,

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ & \ddots & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

Around the middle of the last century, it was thought that the largest systems we could numerically solve were of the order 50×50 (roundoff)

Linear Systems

- They are ubiquitous in physical and mathematical settings
- They arise naturally in the study of *systems* of nonlinear equations, optimization, ordinary and partial differential equations
- Hence, we want to spend a fair amount of effort to solve them accurately and efficiently
- Can be numerically solved using:
  - direct methods:
    - (i) Gauss elimination
    - (ii) LU decomposition
  - iterative methods:
    - (i) Jacobi
    - (ii) Gauss-Siedel
    - (iii) successive over-relaxation

Matrix Operations

- Addition and subtraction is only defined for matrices or vectors of the same size

$$\mathbf{A}_{m\times n} \pm \mathbf{B}_{m\times n} = \mathbf{C}_{m\times n},$$

  where $c_{ij} = a_{ij} \pm b_{ij}$
- This is an element-by-element operation
- For example,

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

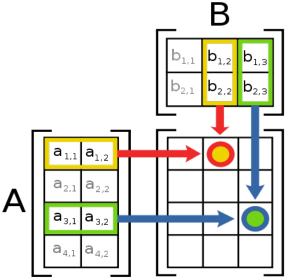
Matrix Operations

Two matrices can only be multiplied if the number of rows and columns match

$$\mathbf{A}_{m\times n} \times \mathbf{B}_{n\times p} = \mathbf{C}_{m\times p},$$

where $c_{ij} = \sum_{k=1}^{m} a_{ik}b_{kj}$

Schematically,



Matrix Operations

Illustration:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 14 & 14 \\ 32 & 32 & 32 \\ 50 & 50 & 50 \end{bmatrix}$$

Questions: True or False?

- If $\mathbf{C} = \mathbf{AB}$, then $\mathbf{C} = \mathbf{BA}$.
- If $\mathbf{C} = \mathbf{A} + \mathbf{B}$, then $\mathbf{C} = \mathbf{B} + \mathbf{A}$.

Matrix multiplication is not *commutative*, while matrix addition is.

Matrix Operations

The transpose of the matrix is obtained by flipping its rows and columns

$$[\mathbf{A}^{\mathrm{T}}]_{ij} = [\mathbf{A}]_{ji}$$

Example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}.$$

Definition: A matrix $\mathbf{M}$, which has $\mathbf{M}^T = \mathbf{M}$ is called a symmetric matrix. For example,

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 9 \end{bmatrix}.$$

Solving Linear Systems

- Given a linear system of m equations in m unknowns, $\mathbf{Ax} = \mathbf{b}$, we will learn how to solve for $\mathbf{x}$
- How sensitive is the solution to noise in $\mathbf{A}$ and $\mathbf{b}$, and round-off errors?
- You may have learned that the solution can be obtained from the inverse as $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. From a computational standpoint, this is a terrible way to solve for $\mathbf{x}$.
- Why? Consider a simple "1D" example $7x = 21$. This can be solved as $x = 21/7 = 3$.
- Alternatively, $x = 7^{-1} \times 21 = 0.142857 \times 21 = 2.99997$. Requires more work, and produces an inferior result.

Gauss Elimination

- Given a linear system of equations $\mathbf{Ax} = \mathbf{b}$, Gauss elimination forms an augmented matrix $[\mathbf{A} \mid \mathbf{b}]$ and proceeds in two phases
  - Forward Elimination: The unknowns are "eliminated" by elementary row operations (add/subtract/scalar-multiply equations) to form an "upper-triangular" system
  - Back Substitution: Starting from x_m use the upper-triangular system to solve for x_{m-1} and so on, all the way to x_1.
- We consider the naive or basic form of Gauss elimination first, and then consider complexities and modifications to handle them

Forward Elimination: Basics

- Consider the first two equations of the system $\mathbf{Ax} = \mathbf{b}$, and call them $\mathbf{E}_1$ and $\mathbf{E}_2$, respectively.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m &= b_2
\end{aligned}
$$

- Consider what happens if I perform the following operation: $\mathbf{E}_2' = \mathbf{E}_2 - (a_{21}/a_{11})\mathbf{E}_1$
- I get:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &= b_1 \\
+ a_{22}'x_2 + \cdots + a_{2m}'x_m &= b_2'
\end{aligned}
$$

where $a_{2j}' = a_{2j} - (a_{21}/a_{11})a_{1j}$, etc.

Forward Elimination: Basics

- In this operation, $\mathbf{E}_1$ is called the *pivot equation*, and a_{11} the *pivot element*
- We can repeat this process to zero out all the elements under a_{11}
- For example, the row i, $a'_{ij} = a_{ij} - (a_{i1}/a_{11})a_{1j}$
- Note that you have to perform the same operations on **b**
- Once we are through the first column, we can consider the smaller sub-system

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \cdots & b_1 \\ & a'_{22} & a'_{23} & \cdots & b'_2 \\ & a'_{32} & a'_{33} & \cdots & b'_3 \\ & & \cdots & & . \\ & a'_{m2} & a'_{m3} & \cdots & b'_m \end{array}\right]$$

Forward Elimination: Basics

Rinse; Repeat; Matlab-like pseudocode for a $m \times m$ matrix:

```
for i = 1:n-1  % run through all the n rows
  for j = i + 1:n % all rows below pivot

    factor = A(j,i)/A(i,i)

    for k = i + 1:n  % elements in current row
      A(j,k) = A(j,k) - A(i,k) * factor
    endfor

    b(j) = b(j) - factor * b(i)

  endfor
endfor
```

Back Substitution

- Once we have an upper triangular system, we can compute $\mathbf{x}$ using back-substitution
- Consider a $m \times m$ system $\mathbf{U}\mathbf{x} = \mathbf{b}$, where $\mathbf{U}$ is upper-triangular

$$\begin{array}{ccccccc} u_{1,1}x_1 & + & u_{1,2}x_2 & + \cdots + & u_{1,m}x_m & = & b_1 \\ & & u_{2,2}x_2 & + \cdots + & u_{2,m}x_m & = & b_2 \\ & & & \ddots & \vdots & & \vdots \\ & & & & u_{m,m}x_m & = & b_m \end{array}$$

- We can easily compute the x_i working from the last equation back towards the first.

Back Substitution

- To get x_m, we can use the formula

$$x_m = \frac{b_m}{u_{m,m}}$$

- Working backwards, for $i = m-1, m-2, ..., 1$

$$x_i = \frac{b_i - \sum_{j=i+1}^{m} u_{i,j} x_j}{u_{i,i}}$$

- It is straightforward to write down the algorithm for these steps.

Back Substitution

```
x(n) = b(n)/U(n,n) % Get last element

% Start from second-last row
for i = n-1:-1:1
  sum = b(i)

  % Compute the sum
  for j = i + 1: n
    sum = sum - U(i,j) * x(j)
  endfor

  x(i) = sum/U(i,i)

endfor
```

Number of Operations

Let us first count the number of:

- addition/subtraction, and
- multiplication/division steps

in forward elimination of a $m \times m$ matrix

It is useful to reference the pseudo-code we wrote earlier.

Loop i	**Loop** j	**add/sub**	**mul/div**
1	2 to m	$(m-1)m$	$(m-1)(m+1)$
2	3 to m	$(m-2)(m-1)$	$(m-2)(m)$
$\vdots$			
k	$k+1$ to m	$(m-k)(m-k+1)$	$(m-k)(m-k+2)$
$\vdots$			
$m-1$	m to m	$(1)(2)$	$(1)(3)$

Number of Operations: Addition/Subtraction

- The total number of add/sub is:

$$\sum_{k=1}^{m-1}(m-k)(m-k+1) = \sum_{k=1}^{m-1}\left(m(m+1) - (2m+1)k + k^2\right)$$

- This equals

$$m(m+1)\sum_{k=1}^{m-1} 1 - (2m+1)\sum_{k=1}^{m-1} k + \sum_{k=1}^{m-1} k^2$$

- Using formulae for sum of integers and squares of integers this equals

$$\boxed{\frac{m^3}{3} + \mathcal{O}(m)}$$

Number of Operations

- Similarly it can be shown that the total number of multiply/divide steps are

$$\boxed{\frac{m^3}{3} + \mathcal{O}(m^2)}$$

- Adding both the operations gives us the cost for forward elimination as:

$$\boxed{\frac{2m^3}{3} + \mathcal{O}(m^2)}$$

Number of Operations: Back Substitution

- From the algorithm, for a particular i, the number of add/sub is $m - i$ and the number of mult/div is $m - i + 1$
- This implies that the total number of operations are:

$$
\begin{aligned}
&= \sum_{i=1}^{m-1} (m - i) + \sum_{i=1}^{m-1} (m - i + 1) \\
&\frac{m(m-1)}{2} + \frac{m(m+1)}{2} \\
&\frac{m^2}{2} + \mathcal{O}(m) + \frac{m^2}{2} + \mathcal{O}(m) \\
&\boxed{m^2 + \mathcal{O}(m)}
\end{aligned}
$$

Number of Operations

- Thus the total number for Gauss elimination including the cost of elimination and substitution is

$$\underbrace{\frac{2m^3}{3} + \mathcal{O}(m^2)}_{\text{elimination}} + \underbrace{m^2 + \mathcal{O}(m)}_{\text{substitution}} = \frac{2m^3}{3} + \mathcal{O}(m^2)$$

- As m increases beyond 100, the fraction of computation spent in elimination is nearly 99%.
- For every order of magnitude increase in m, the computational cost increases by three orders of magnitude

Pitfalls and Workarounds

- So far we looked at naive Gauss elimination
- There are two common problems with this implementation, which can be fixed without increasing the asymptotic cost
- *Division by Zero*: During elimination or substitution we could have a pivot that is zero or nearly zero. Fix: partial pivoting
- *Some forms of ill-conditioning*: Round-off errors can overwhelm solution.* Fix: scaling.

*We will look at this issue in greater detail again later.

Partial Pivoting: Motivating Example

- Consider a system with the augmented matrix:

$$[\mathbf{A}|\mathbf{b}] = \left[\begin{array}{ccc|c} 1 & -1 & 2 & 8 \\ 0 & 0 & -1 & -11 \\ 0 & 2 & -1 & -3 \end{array}\right]$$

- Notice that the pivot element for the second row is 0, and we cannot proceed with naive GE
- However, if we interchanged rows 2 and 3 (no change in solution), we get:

$$[\mathbf{A}|\mathbf{b}] = \left[\begin{array}{ccc|c} 1 & -1 & 2 & 8 \\ 0 & 2 & -1 & -3 \\ 0 & 0 & -1 & -11 \end{array}\right]$$

Partial Pivoting

- Partial pivoting allows us to look for row exchanges opportunistically
- A partial pivoting strategy looks down a column from the (i, i) entry and below to find the entry with the largest magnitude and then exchanges those two rows.
- It is called partial, because there is also a full pivoting strategy that looks for the largest element in the block and calls for row and column exchanges
- Usually only partial pivoting is carried out.
- It is easy to see how it would have fixed the problem in the previous example

Is Partial Pivoting Enough?

- Consider the linear system

$$0.001x_1 + 1.0x_2 = 1.00$$
$$1.00x_1 + 1.00x_2 = 2.00$$

- The exact solution is $x_1 = 1000/999$ and $x_2 = 998/999$.
- Assume we carry out naive GE on a machine with 2-digit arithmetic.
- That is, we store all numbers as $\pm 0.d_1 d_2 \times 10^{\beta}$.
- To eliminate x_1 from the second equation we multiply the first by -1000 and add to get the coefficient of x_2 as $-1000 + 1 = -999$ which is just $-1000 = -0.10 \times 10^4$ in our finite precision machine.

Is Partial Pivoting Enough?

- Similarly, the right hand side is just $2 - 1000 = -998$ which is also -0.10×10^4 on our machine.
- Thus $x_2 = 1.0$. Substituting back into the first equation gives $x_1 = 0$ (not good!)
- So you think partial pivoting might help? It would interchange the equations as:

$$1.00x_1 + 1.00x_2 = 2.00$$
$$0.001x_1 + 1.0x_2 = 1.00$$

- If we eliminate x_1 now, we get $x_2 = 1$, and substituting back in the first equation we get $x_1 = 1$
- Halellujah!

Is Partial Pivoting Enough?

- Let's tweak the same problem a little bit, by multiplying the original eqn.1 by 10,000

$$10x_1 + 10000x_2 = 10000$$
$$1.00x_1 + 1.00x_2 = 2.00$$

- The exact solution is the same as before: $x_1 = 1000/999$ and $x_2 = 998/999$.
- Partial pivoting says we don't have to exchange equations, which leads us to $x_2 = 1$, and $x_1 = 0$
- We are back in trouble!
- The problem is that the matrix is not properly scaled (entries vary widely).

Row Scaling + Partial Pivoting

- In row scaling, we define scale factors for each of the m rows as

$$s_i = \sum_{j=1}^{m} |a_{ij}|$$

- In our example $s_1 = 10,010$ and $s_2 = 2$
- Then instead of finding the largest entry in the column we find the largest entry scaled by the corresponding s_i
- So in our example we have $10/10,010$ and $1/2$, so we should interchange rows and thus our problem is solved accurately.