112001051
VM Sreeram

CS 5016: Computational Methods and Applications

Lecture 03
01/02/23

Linear Systems

In applied science, we often face equations like

$$Ax = b$$

where A is an $n \times n$ matrix whose elements are a_{ij}. x and b are column vectors of dimension n. Component wise, the above eqn can be rewritten as

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad \forall i \{1, 2, .., n\}$$

b can also be interpreted as a linear combination of the columns of matrix A, weighed by the vector x.

Iterative Solution Method

An iterative method for solution of the linear system results in a sequence of vectors $\{x^{(k)}, k \geq 0\}$ of $\mathbb{R}^n$ that converges to the exact solution x, i.e.,

$$\lim_{k \to \infty} x^{(k)} = x \qquad \text{for any initial vector } x^{(0)} \in \mathbb{R}^n$$

Constructing an Iterative Method

We can write A as $A = P - (P - A)$, where P is a suitable non-singular matrix.

Non-singular matrix is a square matrix with non-zero determinant. This property should be satisfied so that then only inverse of it exists.

$$A = P - (P - A)$$

$$\Rightarrow P = A - (A - P) \quad \Rightarrow Px = Ax - (A - P)x$$

$$\Rightarrow Px = b - (A - P)x \quad , \text{where } Ax = b \quad \text{—} (1)$$

Correspondingly for $k \geq 0$, we have

$$Px^{(k+1)} = b - (A - P)x^{k} \quad \text{—} (2)$$

$$(2) - (1) \Rightarrow \quad x^{(k+1)} - x = (I - P^{-1}A)(x^{(k)} - x)$$

where I is the identity matrix of suitable dimension.

Convergence

Let $e^{(k)} = x^{(k)} - x$ denote the error at step k.

If $(I - P^{-1}A)$ is symmetric and positive definite,

$$\| e^{(k+1)} \|_2 = \| (I - P^{-1}A) e^{(k)} \|_2 \leq \rho(I - P^{-1}A) \| e^{(k)} \|_2$$

where $\rho(.)$ is known as spectral radius (maximum radius of eigenvalues) :

$\rho(.) < 1 \Rightarrow$ Convergence

A symmetric matrix is a square matrix that is equal to the transpose of itself , i.e., $A^T = A$

A symmetric matrix is positive definite iff all its eigenvalues are positive.

The Jacobi method

If the diagonal entries of A are non-zero, we can set $P = D$, where D is the diagonal matrix containing the diag. entries of A. Then we get the foll. iteration

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)} \right) \qquad \forall i \in \{1, 2, \ldots, n\}$$

If the matrix A is strictly diagonally dominant by row, then Jacobi method converges.

A matrix is called diagonally dominated if $a_{ii} \geq \sum_{j\neq i} |a_{ij}| \ \forall i$ and is strictly diagonally dominated if it is diagonally dominated at $\exists i$ such that $a_{ii} > \sum_{j\neq i} |a_{ij}|$.

An informal proof : The Jacobi method converges for strictly diagonally dominant matrices because the iterations of the method produce a sequence of matrices with strictly smaller entries, until the soln is reached. This is due to the fact that in each iteration, the off-diag entries are divided by the diag. entries, which are always larger in magnitude.

The Gauss-Siedel method

Faster convergence hopefully could be achieved if the new (k+1) components already available are used.

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \quad \forall\, i \in \{1, 2, \ldots, n\}$$

There are no general results stating that the Gauss-Seidel method converges faster than the Jacobi method.

* Python's numpy.linalg rely on BLAS and LAPACK to provide efficient low-level implementations of standard linear algebra algorithms.

Interpolation

In several applications we may only know the val of a fn f at some given points $\{(x_i, y_i),\ i = 0, 1, 2, \ldots, n\}$. How do we determine f?

Approx. function $\tilde{f}$ such that

$$\tilde{f}(x_i) = y_i \quad \forall\, i \in \{0, 1, \ldots, n\}$$

Cubic fn passing through $(0,1)$, $(1,4)$, $(-1,0)$, $(2,15)$

Let $\tilde{f}(x)$ be

$$\tilde{f}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

On plugging in given points we get

$$a_0 + 0 + 0 + 0 = 1$$
$$a_0 + a_1 + a_2 + a_3 = 4$$
$$a_0 - a_1 + a_2 - a_3 = 0$$
$$a_0 + 2a_1 + 4a_2 + 8a_3 = 15$$

On solving, we get $a_0 = a_1 = a_2 = a_3 = 1$

$\therefore$ The function is $1 + x + x^2 + x^3$

Number of such functions $= 1$ $\because$ this is the only soln of sys of lin. eqns.

This is equivalent to solving for X

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 0 & -1 \\ 1 & 2 & 4 & 8 \end{bmatrix}_{n \times n}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} 1 \\ 4 \\ 0 \\ 15 \end{bmatrix}}_{B}$$

The complexity ~~is~~ to solve this by matrix inversion method is $O(n^3)$ because inverting $A_{n \times n}$ ~~#~~ takes $O(n^3)$ steps, and then latter computation $O(n^2)$. ~~$O(n^2)$ is computational~~

Different kinds of interpolation

- Polynomial interpolation

$$\tilde{f}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- trignometric interpolation

$$\tilde{f}(x) = a_{-M} e^{-iMx} + \cdots + a_0 + \cdots + a_M e^{iMx}$$

- rational interpolation

$$\tilde{f}(x) = \frac{a_0 + a_1 x + \cdots + a_k x^k}{b_0 + b_1 x + \cdots + b_n x^n}$$

Lagrangian Polynomial Interpolation

For $j \in \{0, 1, \ldots, n\}$, define

$$\psi_j(x) = \prod_{i=0, i\neq j}^{n} \frac{x - x_i}{x_j - x_i}$$

Note that

$$\psi_j(x_k) = \begin{cases} \prod_{i=0, i\neq k}^{n} \frac{x_j - x_i}{x_j - x_i} = 1 & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

Then, the required approximation is

$$\tilde{f}(x) = \sum_{j=0}^{n} y_j \psi_j(x)$$

Coefficients of the above polynomial can be computed in $O(n^2)$ time. This is because the calculation of $\tilde{f}$ involves $n+1$ $(= O(n))$ terms, ~~[illegible]~~ and each term can be calculated in $O(n)$ time. So, $O(n) \times O(n) = O(n^2)$.

Example : Function passes through $(0,1)$, $(1,4)$, $(-1,0)$, $(2,15)$.

$$\psi_1(x) = \frac{x^3 - 2x^2 - x + 2}{2}, \quad \psi_2(x) = \frac{-x^3 + x^2 + 2x}{2},$$

$$\psi_3(x) = \frac{-x^3 + 3x^2 - 2x}{6}, \quad \psi_4(x) = \frac{x^3 - x}{6}$$

and,

$$\tilde{f}(x) = \psi_1(x) + 4\psi_2(x) + 15\psi_4(x)$$

$$= x^3 + x^2 + x + 1$$

* Python's scipy.interpolate module spline functions & classes, 1D & multidimensional interpolation classes, etc.

Reservoir example

Let $P_1 = a$, $P_2 = b$, $P_3 = c$, $P_4 = d$ for simplicity

Since no info regarding pipe lengths were given, it is assumed that all lengths are equal.

at point

1. $Q_1 = Q_2 + Q_3 + Q_4 \Rightarrow 10 - a = a - b + a - d + a - c$ — (1)
2. $Q_2 = Q_{10} + Q_9 \Rightarrow a - b = b + b - d$ — (2)
3. $Q_4 = Q_6 + Q_5 \Rightarrow a - c = c + c - d$ — (3)
4. $Q_9 + Q_3 + Q_5 = Q_8 + Q_5 \Rightarrow b - d + a - d + c - d = d + d$ — (4)

Using eqns (1),(2),(3),(4), we solve for a, b, c, d.

From (2),

$$b = \frac{a+d}{3}$$

From (3),

$$c = \frac{a+d}{3}$$

Putting b, c in (4),

$$5d - a = b + c = \frac{2a+2d}{3} \Rightarrow 15d - 3a = 2a + 2d$$

$$\Rightarrow \boxed{13d = 5a}$$

Putting b, c in (1),

$$4a - d = 10 + b + c = 10 + \frac{2a+2d}{3} \Rightarrow 12a - 3d = 30 + 2a + 2d$$

$$\Rightarrow 10a = 30 + 5d$$

$$\Rightarrow \boxed{2a = 6 + d}$$

$\cancel{13a = 5(2a-6) = 10a - 30}$

$\cancel{\Rightarrow 3a =}$ $\cancel{13(2a-6) = 5a}$

$\cancel{26a - 26 = 5a}$

$\cancel{22a = 26}$

$$13(2a-6) = 5a$$

$$26a - 78 = 5a$$

$$a = \frac{78}{21} = \frac{26}{7}$$

$$d = \frac{5}{13} \times \frac{26}{7} = \frac{10}{7}$$

$$b = c = \frac{a+d}{3} = \frac{12}{7}$$

~~b+c~~