Root finding using Bisection & Newton-Raphson Method

For a linear equation in one-dimension $f(x) = 0$, we can find a desired root, or for a set of linear equation, "implicit function theorem" we can solve N equations with N unknowns simultaneously

$$f(\vec{x}) = \vec{0}.$$

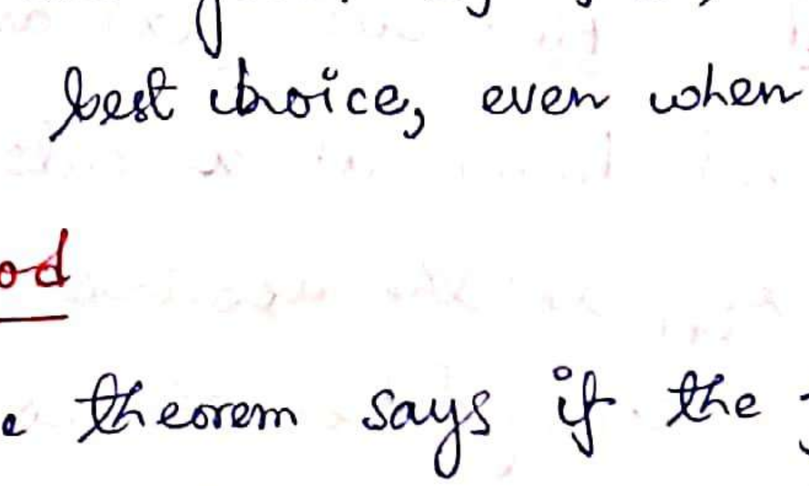
In one-dimension a root can be found by trapping ("bracketing") within an interval. In multidimension, its impossible to guarantee a root by this and if equations are non-linear, then equations may or may not have a root. In all dimensions, except in linear problems, root finding proceeds via iteration from some approximate trial solution towards convergence. Sign change of function is a precursor to hit a root, hence to choose the bracket, but multiple roots is a problem. For instance, if minima found is exactly zero, then you've found a "double root."

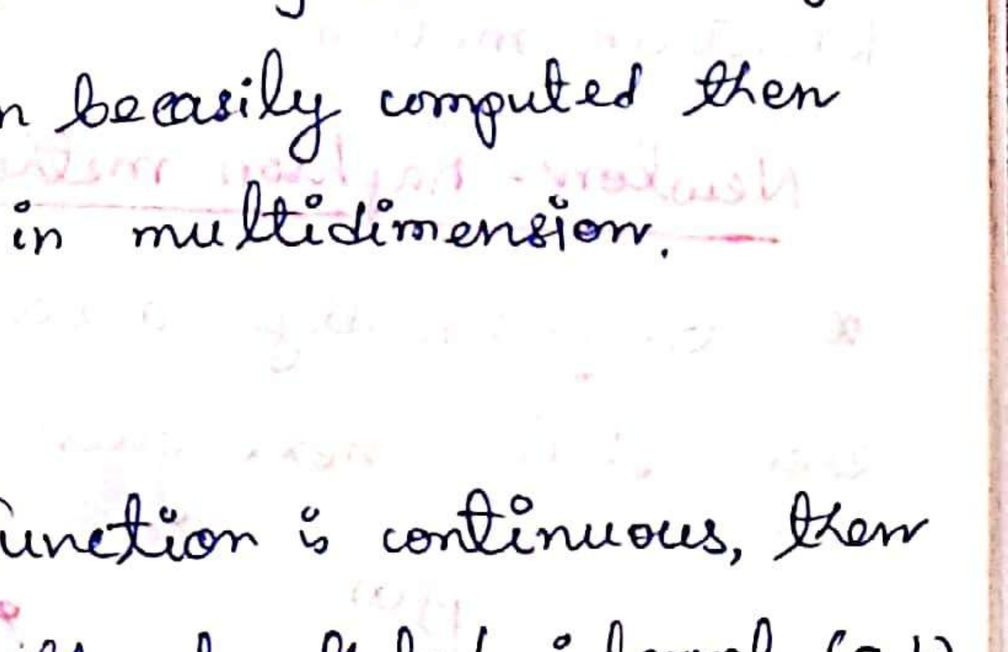
In 1D, Brent's method is best choice, when $f'(x)$ is hard to find. Ridders method is also good. If $f'(x)$ can be easily computed then Newton-Raphson is best choice, even when in multidimension.

Bisection Method

Intermediate value theorem says if the function is continuous, then at least one root must be there within the bracketed interval (a, b) with $f(a)$ & $f(b)$ have opposite signs.

For a bounded discontinuous function, say $f(x) = \frac{1}{x-c}$, a step discontinuity that crosses zero cannot be treated as a root. Even if Bisection method will converge to $x = c$, but $|f(x)| \to \infty$





To choice a bracket is a hard call, for example, for

$f(x) = 3x^2 + \frac{\ln[(\pi - x)^2]}{\pi^4} + 1$ is well behaved except at $x = \pi$ and dips below zero in the interval $x = \pi \pm 10^{-667}$.

In Bisection : (i) Evaluate $f(x)$ at interval's midpoint and examine its sign, (ii) Use the midpoint to replace whichever limit has the same sign. After each iteration, bracket containing the root decrease by a factor of two. If after n iterations, bracket size is ϵ_n, then in next iteration bracket size will be $\epsilon_{n+1} = \epsilon_n/2$.

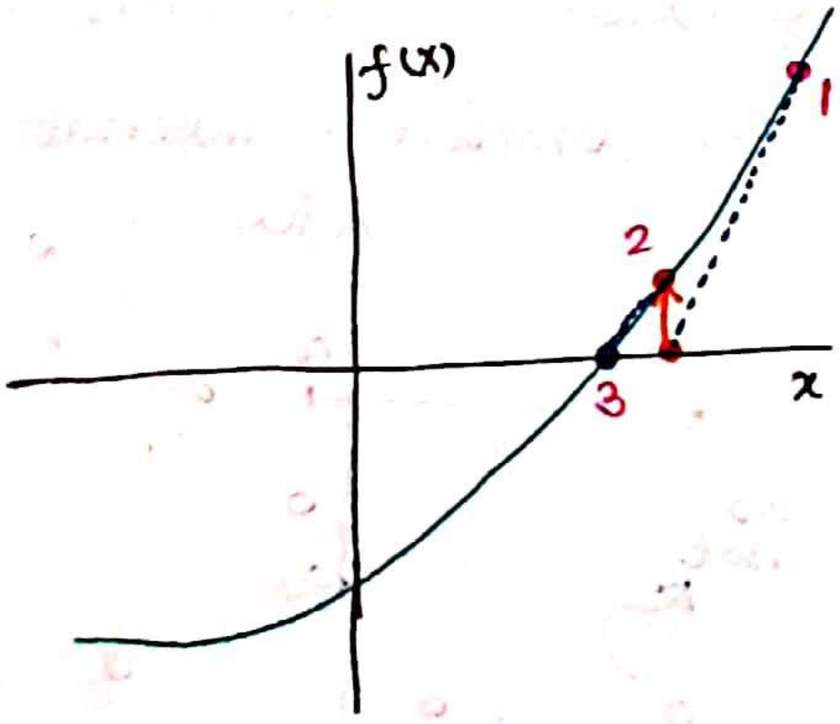
So # of iterations required to achieve given tolerance — linear convergence

$$n = \log_2 \frac{\epsilon_0}{\epsilon}, \quad \epsilon_0 = \text{initial size of bracket}, \ \epsilon = \text{tolerance}.$$

If more than 1 root, then Bisection will find only one. For a smooth function. Secant (or false position) method is faster than Bisection method.

Newton - Raphson method

Find out $f(x)$ and $f'(x)$ at arbitrary x. Geometrically, a tangent drawn at x cuts x-axis (abscissa) then setting next guess x_{i+1} to the abscissa. So NR-method extrapolates the local derivative to find the next estimate of the root. Algebra-ically, using Taylor series,



$$f(x+\delta) \approx f(x) + \delta f'(x) + \frac{\delta^2}{2!} f''(x) + \cdots$$

(nonlinear are unimportant)

So $f(x+\delta) = 0$ implies

$$\boxed{\delta = -\frac{f(x)}{f'(x)}} \quad \text{(Near the root)}$$

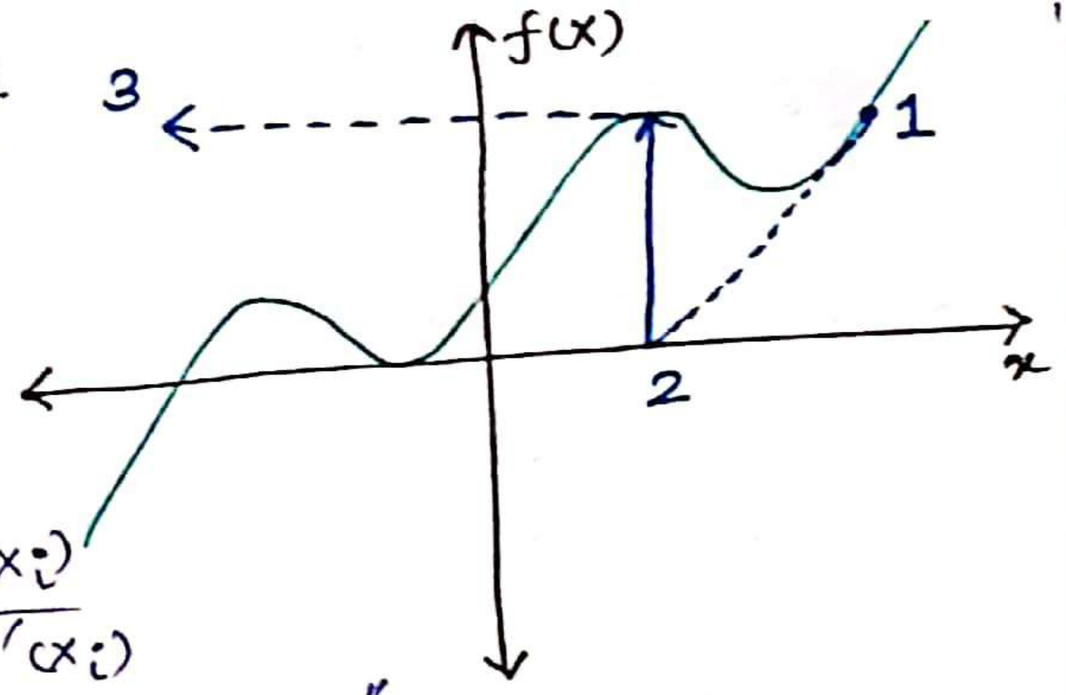
far from root $O(\delta^n)$, $n \geq 2$ are important, so NR method is inaccurate.

If there is a local extrema, then NR method fails because $f'(x) = 0$.

Newton-Raphson converges quadratically as,

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + \frac{\epsilon^2}{2!} f''(x) + \cdots$$

$$f'(x+\epsilon) = f'(x) + \epsilon f''(x) + \cdots$$



So from NR formula $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$

we can write, $\epsilon_{i+1} = \epsilon_i - \frac{f(x_i)}{f'(x_i)} = -\epsilon_i^2 \frac{f''(x)}{2f'(x)}$ by recurrence relation.

So near a root, number of significant digits doubles with each step.

Computation of $f'(x)$ numerically using finite-difference has its pros and cons.

Lagrange Interpolation

In polynomial interpolation, Lagrange polynomial is the polynomial of lowest degree, so that for a set of data points $(x_1, y_1 = f(x_1))$, $(x_2, y_2 = f(x_2))$, $\cdots$, $(x_n, y_n = f(x_n))$ with no two x_i are identical,

$$p(x) = \sum_{j=0}^{n} y_j \ell_j(x) \quad \text{(linear combination)}$$

where Lagrange basis polynomials $\ell_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}$

So interpolating polynomial $p(x)$ is

$$p(x) = \sum_{j=0}^{n} \frac{x - x_0}{x_j - x_0} \cdots \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} \cdots \frac{x - x_n}{x_j - x_n} y_j$$

$x_j \neq x_k$ for well-defined function.

Now $\ell_{j \neq i}(x_i) = \prod_{k \neq j} \frac{x_i - x_k}{x_j - x_k} = \frac{x_i - x_0}{x_j - x_0} \cdots \frac{x_i - x_i}{x_j - x_i} \cdots \frac{x_i - x_n}{x_n - x_k} = 0$ ($x_i - x_i = 0$)

$\ell_i(x_i) = \prod_{k \neq i} \frac{x_i - x_k}{x_i - x_k} = 1$. So all basis polynomials are zero except $x = x_i$; $\ell_i(x) = 1$ because it doesn't have $(x - x_i)$ term.

So $l_j(x_i) = \delta_{ij}$, so that $P(x_i) = \sum_{j=0}^{n} y_j \delta_{ij} = y_i$

Suppose we want to interpolate $f(x) = x^3$ within bracket $1 \le x \le 3$.

$x_0 = 1$, $f(x_0) = 1$
$x_1 = 2$, $f(x_1) = 8$
$x_2 = 3$, $f(x_2) = 27$

So $P(x) = 1 \frac{x-2}{1-2}\frac{x-3}{1-3} + 8\frac{x-1}{2-1}\frac{x-3}{2-3} + 27\frac{x-1}{3-1}\frac{x-2}{3-2}$

$= 6x^2 + 6 - 11x$.

Polynomial interpolation (Lagrange) is susceptible to Runge Phenomena of large oscillation. Commonly cubic spline or trigonometric interpolation is used.

truncation error of

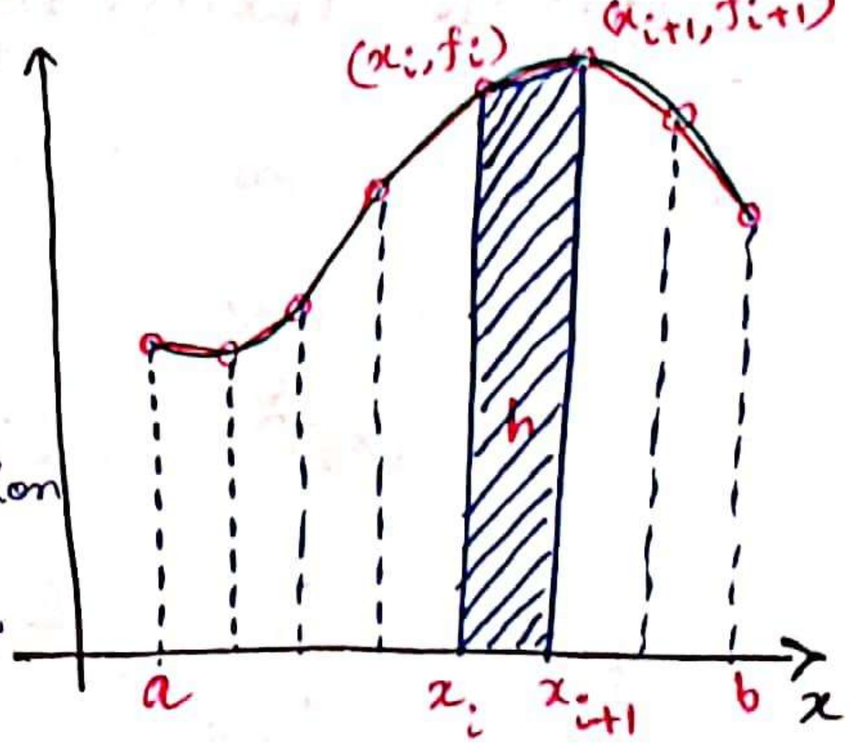
$$\int_a^b f(x)dx = \sum_{i=1}^{n} w_i f(x_i) + R_n$$

Integration of functions - Quadrature

Newton-Leibnitz formula

$$I = \int_a^b f(x)\,dx = F(b) - F(a),$$

$F(x) =$ primitive antiderivative. is often hard to find analytically. Numerical integration of bounded integral in 1D is called quadrature and in higher dimension, called cubature.



Geometrically $\int_a^b f(x)\,dx$ is the area within $y = f(x)$ and lines $x=a$, $x=b$ and x-axis. For uniform sampling within $[a,b]$, equidistant points are $x_i = a + (i-1)h$, $i = 1, 2, \ldots, n$. with $h = \frac{b-a}{n-1}$.

Trapezoidal Rule Replace the graph of the integrand by the polygonal line defined by a finite number of integrand values & then sum the formed trapezoidal area.

$$\int_a^b f(x)\,dx \approx \frac{h}{2}(f_1 + f_2) + \cdots + \frac{h}{2}(f_i + f_{i+1}) + \cdots + \frac{h}{2}(f_{n-1} + f_n)$$

$$\approx h\left[\frac{f_1 + f_n}{2} + \sum_{i=2}^{n-1} f_i\right]$$

$\lim_{h\to 0} \int_a^b f(x)\,dx$ is exact (Riemann integral)

Trapezoidal rule cumulates error $O(h^2)$. We'll use step-halving technique for adaptive control of the integration mesh, which i

(i) calculate integral for given h (n-node), (ii) compare result for $h/2$ ($2n-1$) node

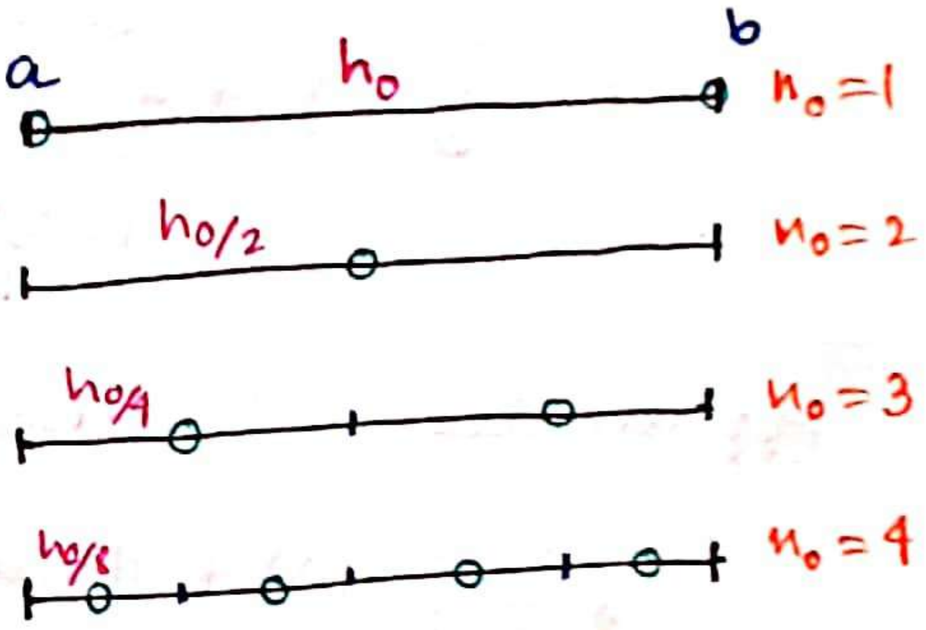
until relative difference drops under a tolerance.

First 3 approximations by trapezoidal rule

$$S_0 = \frac{h_0}{2}[f(a) + f(b)]\ ;\ h_0 = b-a$$

$$S_1 = \frac{h_0}{4}\left[f(a) + f(b) + 2f\left(a+\frac{h_0}{2}\right)\right]$$

$$S_2 = \frac{h_0}{8}\left[f(a) + f(b) + 2f\left(a+\frac{h_0}{4}\right) + 2f\left(a+\frac{h_0}{2}\right) + 2f\left(a+\frac{3h_0}{4}\right)\right]$$



Simpson's 1/3rd rule

It can be shown that Trapezoidal & Simpson's rule can be obtained from a reductionist approach to Newton-Cotes quadrature formulae that says $\int_a^b f(x)\,dx \approx (b-a)\sum_{i=1}^{n} H_i f_i$ where Cotes coefficient

$$H_i = \frac{\int_0^{n-1} \prod_{j\neq i}^{n} [q-(j-1)]\,dq}{(-1)^{n-i}(i-1)!\,(n-i)!\,(n-1)}\ ,\quad i=1,2,\ldots,n. \text{ with } \sum_{i=1}^{n} H_i = 1$$

and $H_i = H_{n-i+1}$.

For odd-number of mesh point $n=3$., $H_1 = \frac{1}{6}$, $H_2 = \frac{2}{3}$, $H_3 = \frac{1}{6}$.

and $b-a = x_3 - x_1 = 2h$, Simpson's formula is

$$\int_{x_1}^{x_3} f(x)\,dx \approx \frac{h}{3}(f_1 + 1f_2 + f_3).$$

Geometrically, we replace $y = f(x)$ by the parabola $y = P_2(x)$ with the Lagrange polynomial $P_2(x)$ defined by the 3 points (x_1, f_1), (x_2, f_2) and (x_3, f_3).

Similar to Trapezoidal rule, using additive property of integrals for subintervals, divide interval $[a,b]$ by odd number $n = 2m+1$

of equally spaced point $x_i = a + (i-1)h$, $i = 1, 2, \ldots, n$ with $h = \frac{b-a}{n-1} = \frac{b-a}{2m}$. So now Simpson's formula can be applied as

$$\int_a^b f(x)\,dx \approx \frac{h}{3}(f_1 + 4f_2 + f_3) + \frac{h}{3}(f_3 + 4f_4 + f_5) + \cdots + \frac{h}{3}(f_{n-4} + 4f_{n-3} + f_{n-2}) + \frac{h}{3}(f_{n-2} + 4f_{n-1} + f_n)$$

$$\approx \frac{h}{3}\left(f_1 + 2\sum_{i=2,4,\ldots}^{n-2} f_i + 4\sum_{i=3,5,\ldots}^{n-1} f_i + f_n\right).$$

($i = 3, 5, \ldots$ (odd); $i = 2, 4, \ldots$ (even))

Approximations in Simpson's rule,

$$S_1 = \frac{h_1}{3}\left[f(a) + 4f(a+h_1) + f(b)\right], \quad h_1 = \frac{h_0}{2} = \frac{b-a}{2}$$

$$S_2 = \frac{h_2}{3}\left[f(a) + 4f(a+h_2) + 2f(a+2h_2) + 4f(a+3h_2) + f(b)\right],$$

$$\left\{ h_2 = \frac{h_1}{2} = \frac{h_0}{4} \right\}$$

It's easy to derive, $S_1^{\text{simpson}} = \left.\frac{4S_1 - S_0}{3}\right|_{\text{trapezoidal}}$, $S_2^{\text{simpson}} = \left.\frac{4S_2 - S_1}{3}\right|_{\text{trapezoidal}}$

so that $S_k^{\text{simpson}} = \left.\frac{4S_k - S_{k-1}}{3}\right|_{\text{Trapezoidal}}$.

Simpson's method converges faster & it is $O(h^4)$ accurate.

Systems of linear Equations

Numerical methods for solving linear systems ⇒ (i) Direct methods (ii) Iterative methods. Direct method can solve a set of linear equations of order n at $\approx n^3$ floating point operations and very good solvers for small systems, but round off error is a serious problem as size of n increases, especially e.g. Cramer's rule. Examples are Gaussian and Gauss-Jordan Elimination, LU-factorization, Cholesky decomposition for SPD system.

Iterative methods, e.g. Jacobi and Gauss-Seidel iteration circumvents n^3. dependence & for a well-conditioned matrix converges to exact solution. Truncation error affects this class & can be controlled by tolerance.

Linear system $\boxed{\overline{\overline{A}}.\vec{x} = \vec{B}}$ where $\overline{\overline{A}} = [a_{ij}]_{nn}$, $\vec{x} = [x_i]_n$, $\vec{B} = [b_i]_n$

written explicitly,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \quad\quad \vdots \quad\quad & \quad\quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned} \qquad \boxed{a_{ii} \neq 0.}$$

else arrange so that this condition is met.

Inverting, $$\begin{aligned} x_1 &= t_1 + s_{12}x_2 + \cdots + s_{1n}x_n \\ x_2 &= s_{21}x_1 + t_2 + \cdots + s_{2n}x_n \\ \vdots \quad\quad \vdots \quad\quad & \quad\quad \vdots \\ x_n &= s_{n1}x_1 + s_{n2}x_2 + \cdots + t_n, \end{aligned}$$ where

$$\left.\begin{cases} s_{ii} = 0, & i = 1, 2, \cdots, n \\ s_{ij} = -a_{ij}/a_{ii}, & i \neq j,\ j = 1, 2, \cdots, n \\ t_i = b_i/a_{ii} \end{cases}\right\} \text{ or, } \boxed{\vec{x} = \overline{\overline{S}}.\vec{x} + \vec{t}}$$

where, $\overline{\overline{S}} = [s_{ij}]_{nn}$

$\vec{t} = [t_i]_n$.

We solve this reduced system by the method of successive approximations from initial approximation $\vec{x}^{(0)} = \vec{t}$, & using recurrence relation, the K^{th} order approximation based on the $(K-1)^{th}$ approximation

$$\vec{x}^{(K)} = \overline{\overline{S}}.\vec{x}^{(K-1)} + \vec{t}, \quad K = 1, 2, \cdots \quad \text{with} \quad \vec{x}_{exact} = \lim_{K\to\infty} \vec{x}^{(K)}$$

Explicitly the iterative procedure is

$$x_i^{(K)} = \sum_{j \neq i}^{n} s_{ij}\, x_j^{(K-1)} + t_i, \quad i = 1, 2, \cdots, n.$$

If absolute error is $\Delta_i^{(K)} = x_i^{(K)} - x_i^{(K-1)}$, $i = 1, 2, \cdots, n$, then Jacobi iteration takes the form,

Jacobi:

$$\begin{cases} \Delta_i^{(k)} = \sum_{j=1}^{n} s_{ij}\, x_j^{(k-1)} + t_i \\ x_i^{(k)} = x_i^{(k-1)} + \Delta_i^{(k)}, \quad i=1,2,\cdots,n \end{cases} \qquad \begin{cases} s_{ij} = -a_{ij}/a_{ii}, \; i,j = 1,2,\cdots n \\ t_i = b_i/a_{ii} \\ s_{ii} = -1 \end{cases}$$

In Gauss-Seidel method, Jacobi method is improved by using the most recently updated $x_i^{(k)}$ instead from the previous iteration $x_i^{(k-1)}$ before even completion of the iteration.

Gauss-Seidel:

$$\begin{cases} \Delta_i^{(k)} = \sum_{j=1}^{i-1} s_{ij} x_j^{(k)} + \sum_{j=i}^{n} s_{ij} x_j^{(k-1)} + t_i \\ x_i^{(k)} = x_i^{(k-1)} + \Delta_i^{(k)}, \quad i=1,2,\cdots,n \end{cases}$$

(updated)

For SPD matrices, Gauss-Seidel always converges irrespective of the initial approximation. Also these algorithms are convergent for following conditions, $\sum_{j=1}^{n} |s_{ij}| < 1, \quad i=1,2,\cdots,n$

$$\sum_{i=1}^{n} |s_{ij}| < 1, \quad j=1,2,\cdots,n$$

and $|a_{ii}| > \sum_{j\neq i} |a_{ij}|, \quad i=1,2,\cdots,n.$

"strictly diagonally dominant"

In practice, in most system $|a_{ii}| > \max_{j\neq i} |a_{ij}|, \quad i=1,2,\cdots,n$

Splitting Method : LU Decomposition:

Splitting methods are matrix iterative methods that split the coefficient matrix into two parts. For a linear system, $\bar{\bar{A}}\cdot\bar{x} = \bar{B}$

We split $A = M - N$ such that the linear system of form $Mx' = B'$ is easy to solve, or $x' = M^{-1}B'$ is easy to compute.

So it follows $Ax = B$ or $(M-N)x = B$ & $Mx - Nx = B$

& $x = M^{-1}Nx + M^{-1}b.$

So given an initial guess x^0, iteration is to compute the

Sequence of vectors $x^{K+1} = M^{-1}Nx^K + M^{-1}B$

Note that $M^{-1}N = M^{-1}(M-A) = I - M^{-1}A \leq I$ (always)

So the method works best when $M^{-1} \sim A^{-1}$ or $M = A$

This is the backbone of iterative methods.

Jacobi Iteration : Its called diagonal inversion as it involves inversion of diag(A) with $A = D - (D-A)$, so that iteration is

$$x^{K+1} = (I - D^{-1}A)x^K + D^{-1}B = x^K - D^{-1}Ax^K + D^{-1}B.$$

Example : $\begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 5 & 1 & 0 \\ 0 & 1 & 6 & 1 \\ 1 & 0 & 1 & 4 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 7 \\ 16 \\ 14 \end{pmatrix}$ has exact solution $x = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix}$

If we start from 1st Jacobi iteration for $x = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ with $D = I\begin{pmatrix} 1 \\ 5 \\ 6 \\ 4 \end{pmatrix}$

then in 19 iterations $(x^{K+1} - x^K < 10^{-6})$

$$x^1 = x^0 - D^{-1}Ax^0 + D^{-1}B = (0.25, 1.4, 2.6667, 3.5)^T$$
$$x^2 = x^1 - D^{-1}Ax^1 + D^{-1}B = (-0.1, 0.8167, 1.85, 2.7708)^T$$
$$x^3 = x^2 - D^{-1}Ax^2 + D^{-1}B = (0.0458, 1.05, 2.0688, 3.0625)^T$$
$$\vdots \quad \vdots \quad \vdots \quad \vdots$$
$$x^{19} = x^{18} - D^{-1}Ax^{18} + D^{-1}B = (0, 1, 2, 3)^T.$$

Gauss-Seidel Iteration : An obvious extension of Jacobi iteration is to invert the entire lower triangular part of $A = L - (L-A)$ so that iteration becomes

$$x^{K+1} = (I - L^{-1}A)x^K + L^{-1}B = x^K - L^{-1}Ax^K + L^{-1}B.$$

Another way is $A = L + U$ (note that this is different than LU decomposition where $A = LU$), then $(L+U)x = B$ gives

$$Lx^{K+1} = -Ux^K + B = (L-A)x^K + B, \text{ so that}$$
$$x^{K+1} = L^{-1}(L-A)x^K + L^{-1}B = (I - L^{-1}A)x^K + L^{-1}B = x^K - L^{-1}Ax^K + L^{-1}B$$

Either we compute L^{-1} and then multiply with $(L-A)x^K + B$, or solve the linear system $Lz = Ax$ using Gaussian elimination (forward substitution)

Previous Example $A = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 5 & 1 & 0 \\ 0 & 1 & 6 & 1 \\ 1 & 0 & 1 & 4 \end{pmatrix}$, $L = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 1 & 5 & 0 & 0 \\ 0 & 1 & 6 & 0 \\ 1 & 0 & 1 & 4 \end{pmatrix}$

$x^1 = (0.25, 1.35, 2.4417, 2.8271)^T$

$x^2 = (-0.0875, 0.9292, 2.0406, 3.0117)^T$

$x^3 = (0.0177, 0.9883, 2, 2.9956)^T$

$\vdots \quad \vdots \quad \vdots \quad \vdots$

$x^{11} = (0, 1, 2, 3)^T$

Total convergence in 11 iterations.

Gaussian Elimination

Suppose we have a set of linear equations $\bar{\bar{A}}\bar{x} = \bar{B}$ with

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix}$$

. A is a nonsingular matrix. Of interest are tridiagonal, symmetric positive (semi)-definite (SPD), triangular etc matrix. A square matrix is lower (upper) triangular if all elements above the main diagonal are zero.

for example $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 16 & 21 \\ 4 & 28 & 73 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 2 & 4 & 0 \\ 4 & 5 & 6 \end{pmatrix}}_{L} \times \underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}}_{U}$

Standard method to solve a linear tridiagonal system is using Gaussian Elimination: means first stage of eliminating all component of A below the main diagonal (or reducing tridiagonal A to row-triangular form) and second stage of backward solution (back solve).

To eliminate x_1, we do

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n - \frac{a_{21}}{a_{11}}(a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n) = 0$$

a_{11} is called the "pivot" element. To eliminate x_1 from all subsequent equations, we perform similar operations over all the subsequent rows to find the "Row Equivalent" form.

In 2nd round to eliminate x_2, a_{22} is the pivot. At the final step we have eliminated all the x_1 to x_{n-1} to find x_n. After x_n is found, back substitution is applied to find $x_{n-1}, x_{n-2}, \ldots, x_1$.

Symbolically the elimination steps are

$$a_{22}^{(1\text{st round})} = a_{22} - \frac{a_{21}}{a_{11}} a_{12} \quad \ldots$$

$$a_{23}^{(1\text{st round})} = a_{23} - \frac{a_{21}}{a_{11}} a_{13} \quad \ldots$$

$$a_{33}^{(2\text{nd round})} = a_{33}^{(1\text{st round})} - \frac{a_{32}^{(1\text{st round})}}{a_{22}^{(1\text{st round})}} a_{23}^{(1\text{st round})} \quad \ldots$$

$$a_{31}^{(2\text{nd round})} = a_{31}^{(1\text{st round})} - \frac{a_{32}^{(1\text{st round})}}{a_{22}^{(1\text{st round})}} a_{21}^{(1\text{st round})} \quad \ldots$$

$$b_2^{(1\text{st round})} = b_2 - \frac{a_{21}}{a_{11}} b_1 \quad \ldots$$

$$b_3^{(2\text{nd round})} = b_3^{(1\text{st round})} - \frac{a_{32}^{(1\text{st round})}}{a_{22}^{(1\text{st round})}} b_2^{(1\text{st round})} \quad \ldots$$

In general,

$$a_{ij}^{(K)} = a_{ij}^{(K-1)} - \frac{a_{iK}^{(K-1)}}{a_{KK}^{(K-1)}} a_{Kj}^{(K-1)}$$ elimination steps

$$b_i^{(K)} = b_i^{(K-1)} - \frac{a_{iK}^{(K-1)}}{a_{KK}^{(K-1)}} b_K^{(K-1)}$$

back substitution steps

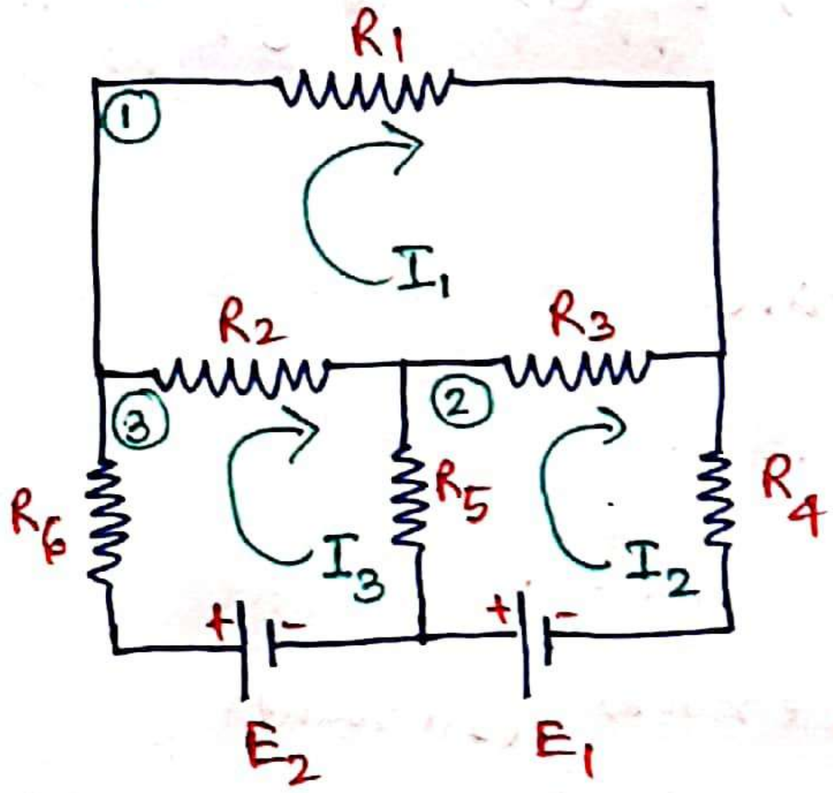
$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

$$x_{n-1} = \frac{b_{n-1}^{(n-1)} - a_{n-1,n}^{(n-1)} x_n}{a_{n-1,n-1}^{(n-1)}} \quad \ldots$$

or $$x_i = \left[b_i^{(n-1)} - \sum_{j=i+1}^{n} a_{ij}^{(n-1)} x_j \right] / a_{ii}^{(n-1)}, \quad i = (n-1), (n-2), \ldots, 1$$

For an ill-conditioned matrix, Gaussian Elimination is adversely affected by rounding error. Also for a large sparse matrix, Gaussian Elimination is worse because L & U is dense. This lead to iterative refinement / improvement of the algorithm.

Solution of Mesh Equations of Electric circuits.



$$① \quad R_1 I_1 + R_2 (I_1 - I_2) + R_3 (I_1 - I_2) = 0$$

$$② \quad R_4 I_2 + R_3 (I_2 - I_1) + R_5 (I_2 - I_3) = E_1$$

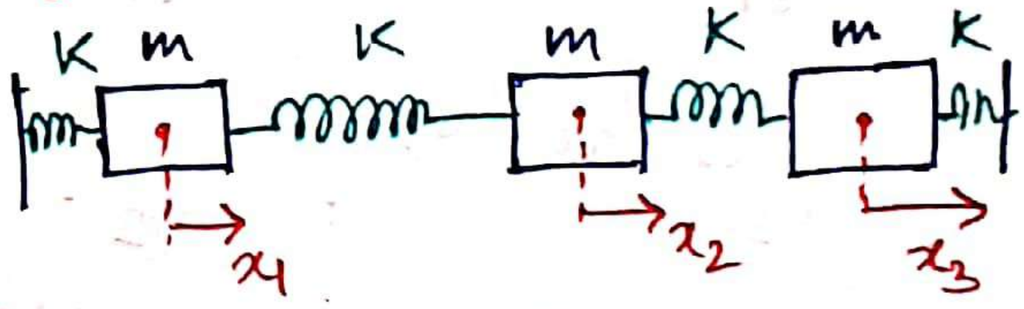
$$③ \quad R_6 I_3 + R_5 (I_3 - I_2) + R_2 (I_3 - I_1) = E_2$$

$R_1 = R_2 = R_3 = 2\Omega$ $\quad \mathcal{E}_1 = \mathcal{E}_2 = 5V$

$R_4 = R_5 = R_6 = 3\Omega$,

Solution, $I_1 = 0.86$, $I_2 = 1.29$, $I_3 = 1.29$.

Solution of Coupled Spring-mass System



$$m\ddot{x}_1 = -Kx_1 + K(x_2 - x_1)$$

$$m\ddot{x}_2 = -K(x_2 - x_1) + K(x_3 - x_2)$$

$$m\ddot{x}_3 = -K(x_3 - x_2) - Kx_3$$

Normal mode solution $\quad x_1(t) = x_1 \cos(\omega t - \phi)$

$$x_2(t) = x_2 \cos(\omega t - \phi)$$

$$x_3(t) = x_3 \cos(\omega t - \phi)$$

Subsitituting, $\quad -m\omega^2 x_1 = -2Kx_1 + Kx_2$

$$-m\omega^2 x_2 = -Kx_1 - 2Kx_2 + Kx_3$$

$$-m\omega^2 x_3 = Kx_2 - 2Kx_3$$

$$\propto \begin{pmatrix} m\omega^2/K - 2 & 1 & 0 \\ 1 & m\omega^2/K - 2 & 1 \\ 0 & 1 & m\omega^2/K - 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

This can be rewritten as $AX = \lambda x$ where

$$\underbrace{\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}}_{\text{eigenvalue}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \frac{m\omega^2}{K} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

λ

eigenfrequency $\omega^2 = \frac{K}{m}\lambda = (K/m\,\lambda_1, K/m\,\lambda_2, K/m\,\lambda_3)$

$$\omega = \sqrt{\frac{2K}{m}\left(1 - \frac{1}{\sqrt{2}}\right)}, \sqrt{\frac{2K}{m}}, \sqrt{\frac{2K}{m}\left(1 + \frac{1}{\sqrt{2}}\right)}$$

Spatial Discretization → finite difference Method (FDM)

finite difference method imply approximating the ~~defi~~ derivatives in an ordinary differential equation (ODE) $\frac{dy}{dx} = f'(x)$ using proper boundary condition by finite difference schemes (say, Taylor series) defined on a discrete set of grid (or mesh) points.

To approximate 1st derivative, Taylor expand $f(x \pm h)$ in following way.

$$f_i(\cdots, x_j - h, \cdots) = f_i(\cdots, x_j, \cdots) - h\frac{\partial f_i}{\partial x_j} + \frac{h^2}{2!}\frac{\partial^2 f_i}{\partial x_j^2} + O(h^3)$$

$$f_i(\cdots, x_j + h, \cdots) = f_i(\cdots, x_j, \cdots) + h\frac{\partial f_i}{\partial x_j} + \frac{h^2}{2!}\frac{\partial^2 f_i}{\partial x_j^2} + O(h^3)$$

So, $\frac{\partial f_i}{\partial x_j} = \frac{f_i(\cdots, x_j, \cdots) - f_i(\cdots, x_j - h, \cdots)}{h} + O(h)$ (Backward)

$$= \frac{f_i(\cdots, x_j + h, \cdots) - f_i(\cdots, x_j, \cdots)}{h} + O(h)$$ (Forward)

Also even if $O(h^2) \neq 0$, then also substraction cancels $O(h^2)$ term

$$J_{ij} = \frac{\partial f_i}{\partial x_j} = \frac{f_i(\cdots, x_j + h, \cdots) - f_i(\cdots, x_j - h, \cdots)}{2h} + O(h^2)$$ (Central)

↳ Jacobian

Definition of Linear FD operators

Operator	Symbol	Difference Representation
Forward Difference	Δ	$\Delta f_j = f_{j+h} - f_j$
Backward Difference	∇	$\nabla f_j = f_j - f_{j-h}$
Central Difference	δ	$\delta f_j = f_{j+h/2} - f_{j-h/2}$
Shift	E	$E f_j = f_{j+h}$
Average	μ	$\mu f_j = \frac{f_{j+h/2} + f_{j-h/2}}{2}$
Differentiation	D	$D f_j = \frac{df_j}{dx} = f_{j,x}$

Now, $f_{j+h} = f_j + h f_{j,x} + \frac{h^2}{2!} f_{j,xx} + \cdots$

or $Ef_j = \left[1 + hD + \frac{h^2D^2}{2!} + \cdots\right] f_j = e^{hD} f_j$

or $E = e^{hD}$. or $hD = \ln E = \ln(1+\Delta) = \Delta - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} - \cdots$

$= -\ln(1-\nabla) = \nabla + \frac{\nabla^2}{2} + \frac{\nabla^3}{3} + \cdots$

Also, $\delta = 2\sinh\left(\frac{hD}{2}\right)$ or $hD = 2\sinh^{-1}\left(\frac{\delta}{2}\right) = \delta - \frac{1}{2^2 3!}\delta^3 + \frac{3^2}{2^4 5!}\delta^5 - \cdots$

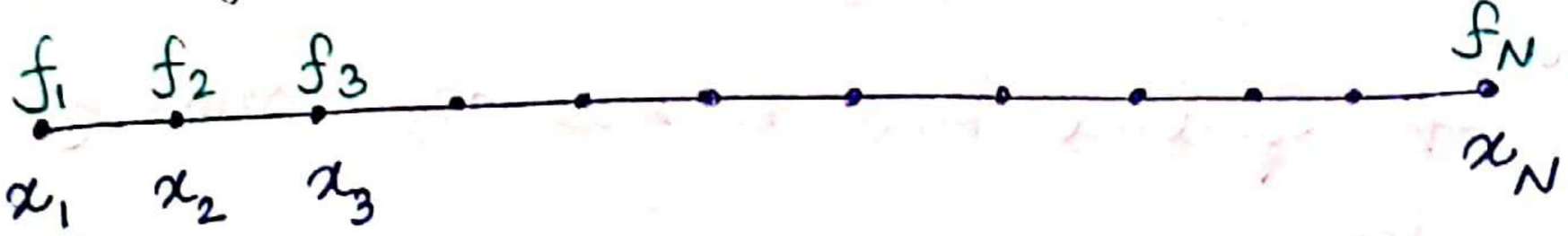
Using these, we can formulate various FD operator form of FDM.

$$hf_{j,x} = \begin{cases} \left(\Delta - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} - \cdots\right) f_j \\ \left(\nabla + \frac{\nabla^2}{2} + \frac{\nabla^3}{3} + \cdots\right) f_j \\ \left(\mu\delta - \frac{1}{3!}\mu\delta^3 + \frac{1}{30}\mu\delta^5 \cdots\right) f_j \end{cases}, \quad h^2 f_{j,xx} = \begin{cases} \left(\Delta^2 - \Delta^3 + \frac{11}{12}\Delta^4 - \cdots\right) f_j \\ \left(\nabla^2 + \nabla^3 + \frac{11}{12}\nabla^4 + \cdots\right) f_j \\ \left(\delta^2 - \frac{\delta^4}{12} + \frac{\delta^6}{90} + \cdots\right) f_j \end{cases}$$

By suitably truncating the infinite series, approximate formulas can be obtained. In computational fluid dynamics course, you'll learn while treating advection term, third order upwind finite difference scheme removes "artificial diffusion"

Differentiation Matrices

Given a set of grid points $\{x_j\}$ and corresponding function values $\{f_i(x_j)\}$, how can we approximate $f_i'(x_j)$. Consider a uniform grid $\{x_1, \cdots, x_N\}$ with $x_{j+1} - x_j = h$ for each j and a set of corresponding data values $\{f_1, \cdots, f_N\}$



Let w_j denote the approximation to $f'(x_j)$. Using Taylor series expansion we found that $w_j = \dfrac{f_{j+h} - f_{j-h}}{2h}$. For simple periodic problem $f_0 = f_N$ and $f_1 = f_{N+1}$. Then

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \frac{1}{h} \begin{pmatrix} 0 & \frac{1}{2} & & & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 0 & \frac{1}{2} \\ \frac{1}{2} & & & -\frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}$$

Toeplitz matrix + Circulent matrix ($a_{ij} \sim i-j$, $a_{ij} \sim (i-j) \pmod N$) + Tridiagonal matrix

An alternate way to reach is via local interpolation and differentiation. For $j = 1, 2, \cdots, N$. Let P_j be the unique polynomial of degree ≤ 2 with $P_j(x_{j-h}) = f_{j-h}$, $P_j(x_j) = f_j$ and $P_j(x_{j+h}) = f_{j+h}$. Set $w_j = P_j'(x_j)$.

$$\text{So } P_j = f_{j-h}\, a_{-h}(x) + f_j\, a_0(x) + f_{j+h}\, a_h(x)$$

$$\text{where } a_{-h}(x) = \frac{(x - x_j)(x - x_{j+h})}{2h^2}, \quad a_h(x) = \frac{(x - x_{j-h})(x - x_j)}{2h^2}$$

$$a_0(x) = -\frac{(x - x_{j-h})(x - x_{j+h})}{h^2}$$

Differentiating & evaluating at $x = x_j$ gives $w_j = \frac{f_{j+h} - f_{j-h}}{2h}$.

We can generalize this to higher orders by taking p_j of degree $\leq n$, so that for $n \leq 4$, we have a "pentadiagonal circulant matrix". By choosing a suitable polynomial of suitable degree, differentiation matrix becomes dense rather than sparse → Spectral Methods that gives order of accuracy $O(h^{12})$. Usage : High precision simulation of rocket/submarine velocity, rheochaos etc.

Gregory-Newton Interpolation & divided differences

Although cumbersome, but Lagrange interpolant polynomial is not very accurate because to add a new point (node), all $L_i^{(n)}$ functions have to be recomputed as its difficult to write

$p_{n+1} = f(L_i^{(n)}, p_n)$. This is avoided in G-N interpolation by

$p_{n+1}(x) = p_n(x) + a_{n+1}\, w_n(x)$ where

$p_{n+1}(x) =$ interpolating polynomial on x_i, $i = 0, 1, 2, \ldots, n, n+1$

$p_{n+1}(x) =$ " " x_i, $i = 0, 1, 2, \ldots, n$.

$p_n(x) =$ "

$$\boxed{w_n(x) = \prod_{i=0}^{n} (x - x_i)}, \quad \boxed{p_0(x) = a_0 = f(x_0)} \quad \boxed{a_{n+1} = \frac{f(x_{n+1}) - p_n(x_{n+1})}{w_n(x_{n+1})}}$$

$$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\cdots(x - x_{n-1})$$

" $\boxed{p_n(x) = \sum_{k=0}^{n} a_k\, w_{k-1}(x)}$ a_k are called "divided differences".

Example We construct the quadratic polynomial that interpolates $\sin x$ within $[0, \pi]$ equally spaced nodes.

So $x_i = 0, \pi/2, \pi$, $y_i = f(x) = 0, 1, 0$. So,

$a_0 = 0 \Rightarrow P_0(x) = 0.$

$$a_1 = \frac{\sin \pi/2 - P_0(\pi/2)}{\pi/2 - 0} = \frac{1-0}{\pi/2} = \frac{2}{\pi} \Rightarrow P_1(x) = \frac{2x}{\pi}$$

$$a_2 = \frac{\sin \pi - P_1(\pi)}{(\pi - 0)(\pi - \pi/2)} = \frac{0-2}{\pi^2/2} = -\frac{4}{\pi^2} \Rightarrow P_2(x) = \frac{2x}{\pi} - \frac{4x}{\pi^2}(x - \pi/2)..$$