Chapter 1 : Numerical Analysis

1.1 Error Analysis

Exact Solution = Approximate Solution + Mathematical Error
Approximate Solution = Numerical Solution + Arithmetic Error
Total Error = Mathematical Error + Arithmetic Error

x = $(-1)^s \cdot (0.d_1 d_2 \dots d_n \dots)_\beta \cdot \beta^e$
In n-digit fl(x), we either chop d_{n+1} or round up d_n.
Machine epsilon is largest ϵ such that fl(1+ϵ)=1

$|E_r(fl(x))| \leq 0.5 \cdot 10^{-n+1}$ for n-digit rounding Relative error in x_A should be $\leq \frac{1}{2}\beta^{-r+1}$ for r signficant digits.
Subtraction often leads to a loss in signifcance. To avoid the same we can try rationalisation or using Taylor's approximation.

Absolute error gets added in addition, relative gets added in multiplication. Except subtraction all operations have tolerable relative error growth.
Condition number conveys how well x_A approximates a function to x, is $|x \cdot f'(x)/f(x)|$. Well or ill conditioned is determined by comparing this number to a threshold. A computation is unstable iff at least has an unbounded condition number.

1.2 Iterative Methods

We tryna solve $A\mathbf{x} = \mathbf{b}$ for $A \in \mathcal{M}_n, b \in \mathbb{R}^n$ iteratively now coz direct methods are computationally expensive. Stationary methods, where x iterates as $\mathbf{x^{(k+1)}} = B\mathbf{x^{(k)}} + \mathbf{c}$.

1.2.1 Jacobi Method

To solve $A\mathbf{x} = b$, write $A = D - C$ where D is a diagonal matrix. Then we have $Dx = Cx + b$, now put a randomass $\mathbf{x^{(0)}}$ in the RHS, compute the corresponding LHS vector as $\mathbf{x^{(1)}}$ and proceed. If D is invertible then $\mathbf{x^{(k+1)}} = B\mathbf{x^{(k)}} + c$ where $B = D^{-1}C, c = D^{-1}b$. This equation is the **Jacobi method**.
For **Jacobi iterative sequence**, just write the linear equations sequentially and express x_1 in terms of others in equation 1, and so on. Now take your initial vector, put this in RHS and compute LHS same as before.
If $\mathbf{e^{(k)}} = \mathbf{x} - \mathbf{x^{(k)}}$ then $\mathbf{e^{(k+1)}} = B \cdot \mathbf{e^{(k)}}$, so Jacobi converges whenever $||B|| < 1$ (subnorm). A sufficient condition for the same is **diagonal dominance** of A (all mod sum).

1.2.2 Gauss Seidel Method

Identical to Jacobi, except in the further equations as we go, we take the updated value of $\mathbf{x^{(k+1)}}$ instead of $\mathbf{x^{(k)}}$. Also converges if A is diagonally dominant, no comments on the converse. GS

converges faster than Jacobi since $\max(\beta_i/1-\alpha_i) = \eta \leq \mu = \max(\alpha_i + \beta_i)$.
Residual error is $\mathbf{r} = \mathbf{b} - A\mathbf{x}^*$, where $\mathbf{x}^*$ is the computed solution. Then the error $\mathbf{e} = \mathbf{x} - \mathbf{x}^*$ can be obtained using the equation $A\mathbf{e} = \mathbf{r}$.
The residual error at k^{th} step $\mathbf{r^{(k)}} = \mathbf{b} - A\mathbf{x^{(k)}}$ can be used as a stopping condition subject to a threshold $\epsilon > 0$.

1.2.3 Convergence Theorem

Spectral radius of $A \in \mathcal{M}_n(\mathbb{C})$ is $\rho(A) = \max|\lambda_j|$ where λ_i are its eigenvalues.
$\rho(A)$ is the infimum over all subordinate matrix norms of A.
Any stationary method converges to a solution of $\mathbf{x} = B\mathbf{x} + \mathbf{c}$ iff $\rho(B) < 1$
In general $\mathbf{e^{(k+1)}} = B \cdot \mathbf{e^{(k)}}$ is the relation for any stationary method's error. Since the initial vector can be anything, $\mathbf{e^{(0)}}$ can be anything.

1.3 Eigenvalues

Eigenvalues of $A \in \mathcal{M}_n$ are the roots of the equation $|A - \lambda I| = 0$, need not be real.

1.3.1 Power Method

For the power method to work, we need A to be diagonalisable (its eigenspace should be $\mathbb{R}^n$) and it to have a unique dominant eigenvalue (geometric=algebraic). Also, the initial guess must have a non-zero component along the dominant eigenvector and it must singularise any A^k.
Using the hypotheses, we get

$$\lim_{k\to\infty} \frac{A^k\mathbf{x^{(0)}}}{\lambda_1^k} = c_1\mathbf{v_1};\ \lim_{k\to\infty} \frac{A^{k+1}\mathbf{x^{(0)}}}{A^k\mathbf{x^{(0)}}} = \lambda_1$$

In the method we compute $\mathbf{y^{(k)}} = A\mathbf{x^{(k-1)}}$, $\mu_k = ||\mathbf{y^{(k)}}||_\infty$ (entry with sign), and $\mathbf{x^{(k+1)}} = \mathbf{y^{(k)}}/\mu_k$, so that essentially max entry of $\mathbf{x^{(k)}}$ is exactly 1. Then $\mathbf{x^{(k)}}$ converges to the eigenvector and μ_k to λ_1.
In the event of multiple dominant eigenvalues, the sequence may not converge and in case the dominant component is zero, the sequence converges to the next dominant eigenvalue.

We can obtain the smallest (dominant) eigenvalue and the corresponding eigenvector of A (under analogous hypotheses) by applying power method on A^{-1}, since λ^{-1} would be an eigenvalue then. Instead of computing A^{-1} and then applying the power method, simply flip the intermediate equation to (LU) $A\mathbf{y^{(k+1)}} = \mathbf{x^{(k)}}$ and apply as always. We can further modulate this method to obtain any eigenvalue of A by taking $\nu \approx \lambda$ and applying inverse power method to $A - \nu I$, since its eigenvalue would be $\lambda - \nu$ and undoubtedly smallest.

1.3.2 Gerschgorin Theorem

Gerschgorin's Circle theorem helps us in localisation of eigenvalues of a matrix. We define Gerschgorin discs $D_k = \{z \in \mathbb{C} : |z - a_{kk}| \leq \rho_k\}$ where $\rho_k = \sum_{j\neq k} a_{kj}$. The theorem says that each eigenvalue must lie in one of these discs, and disjoint sections have exactly as many eigenvalues (as per algebraic multiplicity).

The theorem can be leveraged to determine if the unique dominant eigenvalue hypothesis of power method is satisfied, either in A or A^T since their eigenvalues are same.

1.4 Taylor hun

This chapter introduces key function approximation techniques—starting with Taylor series for local approximation, then moving to polynomial and piecewise polynomial interpolation for broader applicability and flexibility.

$f \in \mathcal{C}^n(I) \implies f$ is n-times continuously differentiable over $I \subset \mathbb{R}$
Taylor's polynomial to n terms: $T_n(x) = \sum_0^n f^{(k)}(a)/k! \cdot (x-a)^k$
Taylor's theorem: for some $\epsilon \in (a, x)$:

$$f(x) = T_n(x) + \frac{f^{(n+1)}(\epsilon)}{(n+1)!} \cdot (x-a)^{n+1}$$

We denote the remainder term as $R_n(x) = f^{(n+1)}(\epsilon)/(n+1)! \cdot (x-a)^{n+1}$
Proof proceed via repeated nested application of Rolle's Theorem
For $f \in \mathcal{C}^n(I) : R_n(x) \leq M_n \cdot |\beta - \alpha|^{n+1}/(n+1)!$ where $f^{(n+1)}(x) \leq M_n \ \forall x \in I$

Taylor Series is given by $f(x) = \sum_0^\infty f^{(k)}(a)/k! \cdot (x-a)^k$ which is valid for all $x \in I$, provided there exists an interval N_a around a such that $\exists M : f^{(k)}(x) \leq M^k \ \forall x \in N_a$

1.5 Interpolation

When we have data from only specific points in an experiment and need values at intermediate points, we use interpolation to construct an approximate function that matches the known data, enabling estimation without repeating the experiment or requiring the exact function form. Polynomial interpolation is the concept of fitting a polynomial to a given set of data.

Given $n+1$ nodal points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$ an interpolating polynomial p has **degree** $\leq n$ and satisfies $p(x_i) = y_i \ \forall i$.
Given $n+1$ nodes (meaning all distinct x_i) there exists a unique interpolating polynomial.
Proof proceeds by forming $n+1$ linear equations and showing invertibility of the van der Monde matrix (ill-conditioned) by virtue of linearly independent columns:

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}$$

1.5.1 Lagrange's

Corresponding to each node x_i we define a Lagrange polynomial l_i. These lagrange polies form a basis of the space $\mathcal{P}_n$ of all polynomials of degree $\leq n$.

$$l_k(x) = \prod_{i \neq k} \frac{x - x_i}{x_k - x_i}$$

The interpolating polynomial over $n+1$ nodes is $p_n(x) = \sum_0^n f(x_i) l_i(x)$
Lagrange polynomials are linearly computable but not scalable with n

1.5.2 Newton's

The coefficient of x^n in the polynomial $p_n(x)$ is denoted by $f[x_0, x_1, \ldots, x_n]$, and is called an n^{th} divided difference of f.
For $n+1$ nodes we can write the interpolating polynomial as:

$$p_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \ldots + f[x_0, x_1, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i)$$

Here the constants are given by:

$$f[x_0, x_1, \ldots, x_n] = \frac{f(x_n) - p_{n-1}(x_n)}{(x_n - x_0)(x_n - x_1) \ldots (x_n - x_{n-1})}$$

Divides differences are symmetric in their argument
The divided difference of $n+1$ nodes is also equal to the the quotient of the difference of divided difference of last n and first n nodes, by $x_n - x_0$
Thus these divided differences can be computed using a table

1.6 Error in Interpolating

In interpolating we have total error = mathematical error + arithmetic error
$ME_n(x) = f(x) - p_n(x); \quad AE_n(x) = p_n(x) - \tilde{p}_n(x)$

1.6.1 Mathematical Error

When $f \in \mathcal{C}^{n+1}[a, b]$, for each $x \in [a, b]$ there exists an $\epsilon_x \in (a, b)$ such that:

$$ME_n(x) = \frac{f^{(n+1)}(\epsilon_x)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

For nodes $x_0, x_1, \ldots, x_n, x \in [a, b]$ - $\exists \epsilon_x \in (a, b) : f[x_0, x_1, \ldots, x_n, x] = f^{(n+1)}(\epsilon_x)/(n+1)!$
For continuous f on $[a, b]$, infinity norm of f is $||f||_\infty = \max |f(x)|$ over $[a, b]$
For linear interpolation: $||ME_1(x)||_\infty \leq (x_1 - x_0)^2 \cdot ||f''||_\infty / 8$

1.6.2 Arithmetic Error

During actual computation we use $\text{fl}(f(x_i)) = f_i$ instead of actual $f(x_i)$
Consider the Lagrange form of interpolating polynomial:
$AE_n(x) = |p_n(x) - \tilde{p}_n(x)| = |\sum_0^n (f(x_i) - f_i) l_i(x)| \leq ||\epsilon||_\infty \sum_0^n ||l_i||_\infty$
Arithmetic error explodes as n grows, can be shown for evenly spaced nodes

1.6.3 Runge Phenomenon

Total Error: $TE_n(x) \leq |f(x) - p_n(x)| + |p_n(x) - \tilde{p}_n(x)| = ||f - p_n||_\infty + ||\epsilon||_\infty M_n$
Here $M_n = \sum_0^n ||l_i||_\infty$ grows exponentially with n
Runge phenomenon: for equally spaced nodes total error is low around the center but becomes particularly large near the edges of the interval, thus making $||f - p_n||_\infty$ large

Faber: For any set of n nodes in $[a, b]$, there exists a function f such that $||f - p_n||_\infty$ does not tend to zero as $n \to \infty$
For any continuous function f on $[a, b]$ there exists a set of nodes such that $||f - p_n||_\infty \to 0$ as $n \to \infty$, even for Runge function $f(x) = 1/1 + 25x^2$ over $[-1, 1]$
Chebyshev nodes $x_i = \cos[(2i + 1)\pi/2(n + 1)]$ serve this purpose for the Runge function

1.7 Piecewise Interpolation

Polynomial interpolation can be ineffective if the polynomial degree is too high, but by using fixed-degree polynomials over smaller intervals (piecewise interpolation), we achieve better accuracy in approximating the function.

1.7.1 Linear

Legit self explanatory, nothing notable. Just exists.
Lacks differentiability at nodes, unless lucky. Fixed by osculating interpolation.

1.7.2 Hermite

Osculating interpolation seeks a polynomial $h(x)$ such that $h^{(k)}(x_j) = f^{(k)}(x_j)\ \forall j$
Taylor's polynomial with degree n is osculating with a single node and $m_0 = n$
Given $n+1$ nodes x_i and points y_i, z_i, there exists a unique polynomial of degree $\leq 2n+1$ satisfying $h_{2n+1}(x_j) = y_j, h'_{2n+1}(x_j) = z_j$, expressible in terms of lagrange polynomials
Hermite polynomial is $h_{2n+1}(x) = \sum y_j(1 - 2(x - x_j)l'_j(x_j))l_j^2(x) + \sum z_j(x - x_j)l_j^2(x)$

1.7.3 Spline

A spline is a curve interpolating $n+1$ nodes on $[a, b]$ which is a polynomial of degree upto d between consecutive nodes and even its $(d-1)^{th}$ derivative is continuous on $[a, b]$
Cubic splines are obtained by first finding their 2nd derivatives which must be linear over every segment, making them a straight line over $[a, b]$. Then back-integration.

Root Finding

This chapter discusses methods for approximating the roots of nonlinear equations $f(x) = 0$, where an approximate root x is a value in $[a, b]$ such that $|r - x|$ is small and $f(x) \approx 0$, and introduces iterative methods to improve approximations of the root through initialisation and repeated refinement. Closed domain methods require an interval where at least one root is known to exist, whereas open domain commence arbitrarily.

1.8 Closed Domain (Bracketing) Methods

Closed domain methods begin with an interval $[a_0, b_0]$ that is known to contain at least one root of the given nonlinear equation, using IVT. These methods iteratively reduce the length of the interval while ensuring that at each step, at least one root remains within it.

1.8.1 Bisection Method

In bisection method we halve the length of the interval at each step by considering the sign of the midpoint and again applying IVT. We stop when some midpoint becomes a root or $|b_n - a_n|$ falls below a threshold.
For $f \in \mathcal{C}[a, b]$ with $f(a) \cdot f(b) < 0$, $\exists r \in [a, b] : f(r) = 0$ and $|x_n - r| < |b - a|/2^n$
Note that there isn't strict monotonic convergence: $|x_{n+1} - r| > |x_n - r|$ can hold

1.8.2 Regula-Falsi Method

In Regula Falsi we follow an identical procedure, except here we obtain x_n as the point where the line joining $(a, f(a))$ and $(b, f(b))$ cuts x-axis. This ensures the endpoint closer to the root is prioritised.
The intervals may not converge to length 0 but x_n certainly converges to the root.

1.9 Open Domain Methods

Open domain methods are iterative techniques that do not require a known interval containing a root, unlike closed domain methods. These methods allow arbitrary starting points to generate sequences, but the sequence may not always converge or be constructible.

1.9.1 Secant Method

In secant method we just scrap the initial requirement of domain for regula falsi method and the rest is identical. Absolute retarded shit bruh, fvck Baskar.
The iteration is $x_{n+1} = x_n - f(x_n)/m$ where m is the slope of the $x_{n-1} - x_n$ secant
All iterates mot not be defined and even if they are, the method may not converge

If $f \in \mathcal{C}^2(\mathbb{R})$ and r is a **simple root** of $f(x)$, *i.e.* $f'(r) \neq 0$, then $\exists \delta \in \mathbb{R}^+$ such that for

any $x_0, x_1 \in [r-\delta, r+\delta]$, secant method is well-defined, within $[r-\delta, r+\delta]$ and converges to r. Also we have the **order of convergence** ≈ 1.62:

$$\lim_{n\to\infty} \frac{|x_{n+1}-r|}{|x_n-r|^\alpha} = \left|\frac{f''(r)}{2f'(r)}\right|^{\frac{\alpha}{\alpha+1}} : \alpha = \frac{\sqrt{5}+1}{2}$$

For any n and $x_{n-1} \neq x_n \neq r$, we have $\xi_n, \zeta_n \in \mathbb{R}$ such that: (using MVT & Rolle's)
$2(x_{n+1}-r)f'(\xi_n) = (x_n-r)(x_{n-1}-r)f''(\zeta_n)$

1.9.2 Newton-Raphson Method

The secant method can be modified for **quadratic convergence** by replacing the secant slope with the tangent slope at x_n, resulting in the Newton-Raphson method.
The iteration is $x_{n+1} = x_n - f(x)/f'(x_n)$. Only x_0 needed to initiate.
The algorithm converges for $f \in \mathcal{C}^1(\mathbb{R})$ if $f'(r) \neq 0$ and x_0 is close to r

If $f \in \mathcal{C}^2(\mathbb{R})$ an r is a **simple root** of $f(x)$, *i.e.* $f'(r) \neq 0$, then $\exists \delta \in \mathbb{R}^+$ such that for any $x_0, x_1 \in [r-\delta, r+\delta]$, Newton-Raphson is well-defined, within $[r-\delta, r+\delta]$ and converges to r. Also we have the **order of convergence** $= 2$:

$$\lim_{n\to\infty} \frac{|x_{n+1}-r|}{|x_n-r|^2} = \left|\frac{f''(r)}{2f'(r)}\right|$$

If f is strictly increasing and strictly convex and has a root, then the root is unique and is attained by Newton-Raphson for any choice of $x_0 \in \mathbb{R}$

1.9.3 Fixed-Point Iteration

In fixed-point iteration, solving $f(x) = 0$ is reinterpreted as finding a fixed point of a function g, where a fixed point satisfies $x = g(x)$, with multiple possible choices of g.
The iteration is simply $x_{n+1} = g(x_n)$, for an initial guess x_0

A good iteration function g must have well-definedness and convergence for x_n, so it must have its range contained with the domain, making it a **self-map**.
To ensure convergence, we assume continuity and contractibility of g. A **contraction map** over $[a, b]$ is $h \in \mathcal{C}^2[a, b]$ such that $\exists\ K \in (0, 1) : |h'(x)| \leq K\ \forall x \in [a, b]$.

If $g \in \mathcal{C}^2[a, b]$ is a self-map and contraction map then $x = g(x)$ has a unique root $r \in [a, b]$ and fixed-point iteration converges for any $x_0 \in [a, b]$. For $\lambda = \max |g'(x)|$:

$$|x_n - r| \leq \frac{\lambda^n}{1-\lambda}|x_1 - x_0|; \quad \lim_{n\to\infty} \frac{r - x_{n+1}}{r - x_n} = g'(r)$$

Fixed-point iteration gives linear convergence, thus order is at least 1.

1.10 Quadrature Rules

We try to estimate the definite integral of a function by evaluating it at chosen points and combining appropriately. We interpolate the function to a polynomial then integrate.

This process yields $I(f) \approx I(p_n) = w_0 f(x_0) + w_1 f(x_1) + \ldots + w_n f(x_n)$ where x_i are quadrature points and w_i are weights.
When x_i are equally spaced we get the *Newton Cotes formula*

General Integral MVT: for continuous f and integrable g which retains sign over $[a, b]$, we have $\int_a^b f(x)g(x)dx = f(c)\int_a^b g(x)dx : c \in (a, b)$
The degree of precision of a quadrature formula is the largest positive integer n such that the formula is exact for all polynomials of degree $\leq n$.

1.10.1 Rectangle Rule

In case of degree 0, *i.e.* n=0, we have the rectangle and midpoint rules.
$I(p_0) = (b-a)f(x_0)$: $x_0 = a$ is rectangle rule and $x_0 = (a+b)/2$ is midpoint rule
Mathematical Error in **Rectangle Rule** $= f'(\eta)(b-a)^2/2$ for some $\eta \in (a, b)$

1.10.2 Trapezoidal Rule

When $n = 1$, we have $p_1(x) = f(x_0) + f[x_0, x_1](x - x_0)$ where $x_0 = a, x_1 = b$
Thus $I(f) \approx I_T(f) = (b-a) \cdot (f(a) + f(b))/2$
Mathematical Error in **Trapezoidal Rule** $= -f''(\eta)(b-a)^3/12$ for some $\eta \in (a, b)$
If we use multiple subintervals $[x_j, x_{j+1}]$ where $h = (b-a)/n$ and $x_j = a + jh$, then we get **Composite** Trapezoidal Rule: $I_T^n(f) = h \cdot [\frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{n-1}) + \frac{1}{2}f(x_n)]$

1.10.3 Simpson's Rule

For $n = 2$ we have $p_2(x) = \sum_0^2 f(x_i)l_i(x)$ where $x_0 = a, x_1 = (a+b)/2, x_2 = b$
Simpson's Rule: $I(f) \approx I_S(f) = \int_a^b p_2(x)dx = (b-a)/6 \cdot [f(a) + 4f((a+b)/2) + f(b)]$
Mathematical Error: $ME_S(f) = I(f) - I(p_2) = -f^{(4)}(\eta)(b-a)^5/2880$ for some $\eta \in (a, b)$
With $2n$ subintervals $[x_{2j}, x_{2j+2}]$ where $h = (b-a)/2n$ and $x_j = a + jh$, then **Composite** Simpson's Rule: $I_S^n(f) = h/3 \cdot [f(x_0) + f(x_{2n}) + 2\sum_{i=1}^{n-1} f(x_{2i}) + 4\sum_{i=0}^{n-1} f(x_{2i+1})]$

1.10.4 Gaussian Rules

The method of undetermined coefficients determines weights for integration rules simply by imposing the condition that the rule is exact for polynomials of degree $\leq n$.
Gaussian quadrature determines both the nodes and weights such that the integration rule is exact for polynomials of degree $\leq 2n + 1$, unlike fixed-node methods.

For $n = 0$, we have $\int_{-1}^1 f(x) \approx 2f(0)$
For $n = 1$, we get $\int_{-1}^1 f(x) \approx f(-1\sqrt{3}) + f(1/\sqrt{3})$
In general we solve the system of linear equations upto degree $2n + 1$
Mathematical Error in Gaussian Rule: $|ME_n(f)| \leq 2(b-a)\inf_{\deg q \leq 2n+1} ||f - q||_\infty$

1.11 Finite Differences

This section introduces numerical differentiation, focusing on approximating derivatives of a function at a given point. The first derivative can be approximated using difference quotients based on the definition of a derivative. For higher orders we explore.

1.11.1 Primitive Difference Approximations

Three formulae used nodes for computing first derivative: forward difference uses x to $x+mh$, backward difference uses $x-mh$ to x and central difference uses $x-mh$ through $x+mh$.
$D^+f(x) = [f(x+h) - f(x)]/h;\ D^-f(x) = [f(x) - f(x-h)]/h$
Central Derivative: $D^0f(x) = [f(x+h) - f(x-h)]/2h$
Forward mathematical error: $f'(x) - D^+f(x) = -h/2 \cdot f''(\eta)$ for some $\eta \in (x, x+h)$
Backward mathematical error: $f'(x) - D^-f(x) = h/2 \cdot f''(\eta)$ for some $\eta \in (x-h, x)$
Central mathematical error: $f'(x) - D^0f(x) = -h^2/6 \cdot f'''(\eta)$ for some $\eta \in (x-h, x+h)$
All proofs simply use Taylor expansion and orders are 1 and 2 respectively

1.11.2 Interpolation Approximation

To approximate $f'(x)$, we use $f'(x) \approx p_n'(x)$, where $p_n(x)$ is the interpolating polynomial for $f(x)$. Varying n and node placement gives different formulas.
For $n = 1 : p_1(x) = f(x_0) + f[x_0, x_1](x - x_0) \implies f'(x) \approx p_1'(x) = f[x_0, x_1]$

For $f \in \mathcal{C}^{n+2}[a,b]$ and p_n interpolating over $n+1$ nodes in $[a,b]$, we have the error:

$$f'(x) - p_n'(x) = \prod(x - x_i)\frac{f^{(n+2)}(\eta_x)}{(n+2)!} + \Big[\prod(x - x_i)\Big]'\frac{f^{(n+1)}(\xi_x)}{(n+1)!}$$

Forward difference on 3 nodes: $[-3f(x) + 4f(x+h) - f(x+2h)]/2h$ (order 2)

1.11.3 Undetermined Coefficients

The method of undetermined coefficients derives numerical differentiation formulas by expressing $f^{(k)}(x) \approx w_0f(x_0) + w_1f(x_1) + \cdots + w_nf(x_n)$, where the weights w_i are determined to ensure the formula is exact for polynomials up to degree n.

Central difference second derivative: $[f(x+h) - 2f(x) + f(x-h)]/h^2$ (order 2)
Mathematical Error: $f''(x) - D^{(2)}f(x) = -h^2/12 \cdot f^{(4)}(\xi)$ for some $\xi \in (x-h, x+h)$
Arithmetic Error: $D^{(2)}f(x) - \bar{D}^{(2)}f(x) = 4\epsilon_\infty/h^2$ where $\epsilon_\infty = \max\{AE(f(x_i))\}$

1.12 Initial Value Problems

IVP differential equation: $y' = f(x,y);\ y(x_0) = y_0, x_0 \in [a,b]$
Equivalent to $y(x) = y_0 + \int_{x_0}^{x} f(t, y(t))dt\ \ \forall x \in [a,b]$
We discretise the interval $[a,b]$ into n nodes as $x_j = x_0 + jh;\ x_0 = a, x_n = b$
Euler's method requires a small step size for good accuracy, making it inefficient.

Picard: Given a real planar domain D and $(x_0, y_0) \in D$, if there is a rectangle around (x_0, y_0) where f is Lipschitz continuous in y, then the IVP has a unique solution in some interval around x_0.
Existence Uniqueness: Lipschitz continuity in y also follows from the partial derivative of f wrt y being continuous (but not vice-versa)

1.12.1 Euler's Method

We know $y(x_0)$, can obtain $y(x_1)$ using forward difference formula for the derivative:
$y(x+h) = y(x) + hy'(x) = y(x) + h \cdot f(x, y(x))$
Thus we get the iterations as $y_{j+1} = y_j + h \cdot f(x_j, y_j)$ for Forward Euler method.
Similarly Backward Euler method is given by $y_{j+1} = y(x-h) = y_j - h \cdot f(x_j, y_j)$

Total Error in Euler Method = Mathematical Error + Arithmetic Error
Mathematical Error = Truncation Error + Propagated Error
Truncation error $T_j = h^2/2 \cdot y''(\xi_j)$, due to premature termination of Taylor series
Propagated error is $y(x_j) - y_j + h \cdot [f(x_j, y(x_j)) - f(x_j, y_j)]$

$$ME(y_{j+1}) = \left[1 + h \cdot \frac{\partial f(x_j, \eta_j)}{\partial y}\right] ME(y_j) + \frac{h^2}{2} \cdot y''(\xi_j); \quad \eta_j \in (y_j, y(x_j)), \xi_j \in (x_j, x_{j+1})$$

For $y \in \mathcal{C}^2[a,b]$, $\frac{\partial f(x,y)}{\partial y} < L, |y''(x)| < Y$: $\text{ME}(y_j) \leq hY/2L \cdot (e^{L(x_n - x_0)} - 1)$
Including arithmetic error for $y_j = \tilde{y}_j + \epsilon_j$, we get the total error $TE(y_j) = y(x_j) - \tilde{y}_j$:

$$|TE(y_j)| \leq \frac{1}{L}\left(\frac{hY}{2} + \frac{\epsilon_\infty}{h}\right)(e^{L(x_n - x_0)} - 1) + e^{L(x_n - x_0)}|\epsilon_0|$$

1.12.2 Runge Kutta Methods

Runge-Kutta methods improve accuracy without needing higher derivatives by evaluating $f(x, y)$ at selected points in each subinterval. The second-order method is derived here.
The Runge-Kutta method of order 2 is obtained by truncating the Taylor expansion of $y(x+h)$ after the quadratic term to get:

$$y(x+h) = y(x) + h[f(x, y(x))] + \frac{h^2}{2}\left[\frac{\partial f(x, y(x))}{\partial x} + f(x, y(x))\frac{\partial f(x, y(x))}{\partial y}\right] + O(h^3)$$

Writing the 2-dimensional Taylor expansion of $f(s,t)$ around (η, τ) we get (order 2)
$f(s,t) = f(\xi, \tau) + (s - \xi)\frac{\partial f(\xi,\tau)}{\partial s} + (t - \tau)\frac{\partial f(\xi,\tau)}{\partial t}$
Take $(\xi, \tau) = (x_j, y(x_j))$ to obtain the approximation:

$$y_{j+1} = y_j + \frac{h}{2}f(x_j, y_j) + \frac{h}{2}f(x_j + h,\ y_j + hf(x_j, y_j))$$

1.12.3 Modified Euler Methods

We can use Euler's method on the equivalent integral form of IVP and apply quadrature formula to $\int_{x_{j-1}}^{x_{j+1}} f(s,y)ds = f(x_j, y_j) \cdot (x_{j+1} - x_{j-1})$, which yields:

Euler's **Mid-point Method**: $y_{j+1} = y_{j-1} + 2h \cdot f(x_j, y_j)$

Implicit **Trapezoidal Method**: $y_{j+1} = y_j + h/2 \cdot (f(x_j, y_j) + f(x_{j+1}, y_{j+1}))$
To use implicit methods we can either solve the non-linear equation, or use the *predictor-corrector approach*: use forward Euler to find a dummy y_{j+1} for putting in RHS.
We say that the origin 0 is (asymptotically) stable if $y(x) \to 0$ as $x \to \infty$