CS375 Final Cheat Sheet

Ryan Scherbarth

November 2024

1 Final review

1.1 Big O Notation

1.2 Convergence

1.2.1 Algebraic Convergence

When error decreases as a power of h or x, e.x. $O(h^n)$.

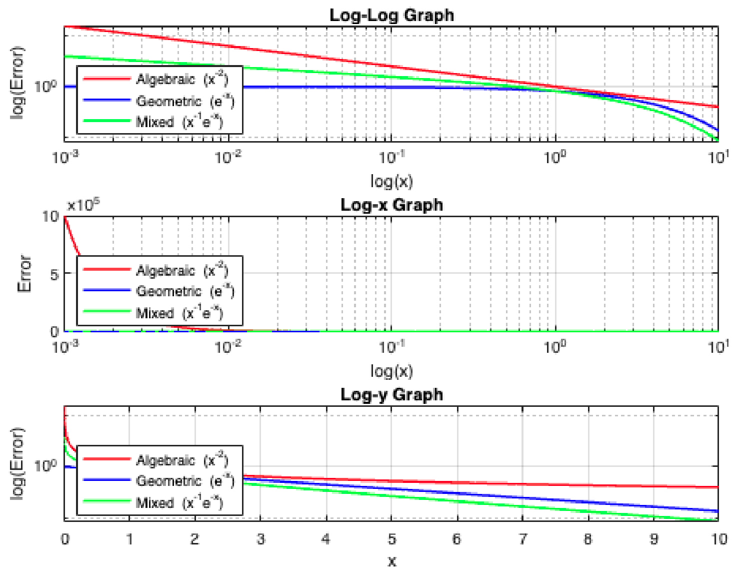
On a **log-log** graph, a straight line with slope n gives us algebraic convergence x^n.

1.2.2 Geometric Convergence

When error decreases exponentially, e.x. $O(e^{-x})$.

On a log-y plot, linear decay means we have a geometric convergence e^{-x}.

1.2.3 Identifying using Graphs



Algebraic (x^{-2}) is a straight line on **log-log** plot, with slope 2.
Geometric (e^{-x}) is a straight line on the **log-y** plot.
I don't really know when, if ever, we care about a **log-x** plot.

1.3 Error

$$\textbf{Absolute Error}: |x_{true} - x_{approx}|$$
$$\textbf{Relative Error}: \frac{|x_{true} - x_{approx}|}{|x_{true}|}$$
$$\textbf{Residual } (r): |f(x_n) - 0|$$
$$: b - A\overrightarrow{x}$$

Residual - How close we are to solving the problem - Tolerance on output, backward error.

$$\textbf{Absolute Tolerance}: |x_n - x_{n-1}| < \partial_x$$
$$\textbf{Relative Tolerance}: |\frac{x_n - x_{n-1}}{b-a}| < \partial_x$$

Where x_n is the current guess (midpoint of current bracket), and n_{n-1} is the previous guess (midpoint of prev. bracket).

1.4 Polynomial Interpolation

Polynomial Interpolation is the process of constructing a polynomial $P(x)$ of degree n, passing through $n+1$ given points, $(x_0, y_0), (x_1, y_1), \ldots$. We use PI to approximate the underlying function $f(x)$ using $P(x)$ s.t. $P(x_i) = f(x_i)$ for all i.

1.4.1 Newton Cotes Rules

Newton-Cotes integration rules use evenly spaced points on the interval $[a, b]$.

Name	n	Formula
Trapezoid	1	$\frac{(b-a)}{2}[f(a) + f(b)]$
Simpson's $\frac{1}{3}$	2	$\frac{(b-a)}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$
Simpson's $\frac{3}{8}$	3	$\frac{(b-a)}{8}[f(a) + 3f(a+h) + 3f(b-h) + f(b)]$
Boole's	4	$\frac{(b-a)}{90}[7f(a) + 32f(a+h) + 12f(\frac{a+b}{2}) + 32f(b-h) + 7f(b)]$

1.4.2 Techniques for finding a polynomial interpolate

1. **Monomial Basis**

   Monomial Basis is expressing the polynomial in terms of x^n powers. Like building a skyscraper from bottom up adding $x^2, x^3, \dots$ levels.

   $$P(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \dots$$

   To find the coefficients, c_n, form the system;

   $$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \dots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \dots \\ f(x_n) \end{bmatrix}$$

   Numerically unstable for large n, especially with equally spaced points, which leads to the introduction of **Runge's Phenomenon** (s-1.4.3)

   Advantages

   (a) Simple

   Disadvantages

   - Ill-conditioned for large n
   - prone to numeric instability

2. **Lagrange Basis**

   Lagrange Basis is expressing the polynomial using predetermined **basis functions** $\ell_i(x)$, which will be 0 everywhere except a single point.

   $$P(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x)$$
   $$\ell_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

   Advantages

   - Each $\ell_i(x)$ is zero at all x_j except x_i, which makes it easier to interpret
   - You don't need to solve a system of equations

   Disadvantages

   - Computationally expensive for repeated high-degree polynomials

3. **Newton Basis**

   Newton Basis is expressing the polynomial in a nested (divided difference) form.

$$
\begin{aligned}
P(x) &= c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + \cdots + c_n(x-x_0)\dots(x-x_{n-1}) \\
c_i &= \frac{f[x_i,\dots,x_{i+k}] - f[x_{i-1},\dots,x_{i+k-1}]}{x_i - x_{i-1}} && \text{divided differences} \\
f[x_i,\dots,x_j] &= \frac{f[x_{i+1},\dots,x_j] - f[x_i,\dots,x_{j-1}]}{x_j - x_i}
\end{aligned}
$$

   Advantages

   - Efficient to add or remove points
   - More numerically stable

   Disadvantages

   - More complex than Lagrange
   - Harder to interpret than monomial form

1.4.3 Polynomial interpolate Error Formulas

1. **Evenly Spaced Points**

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

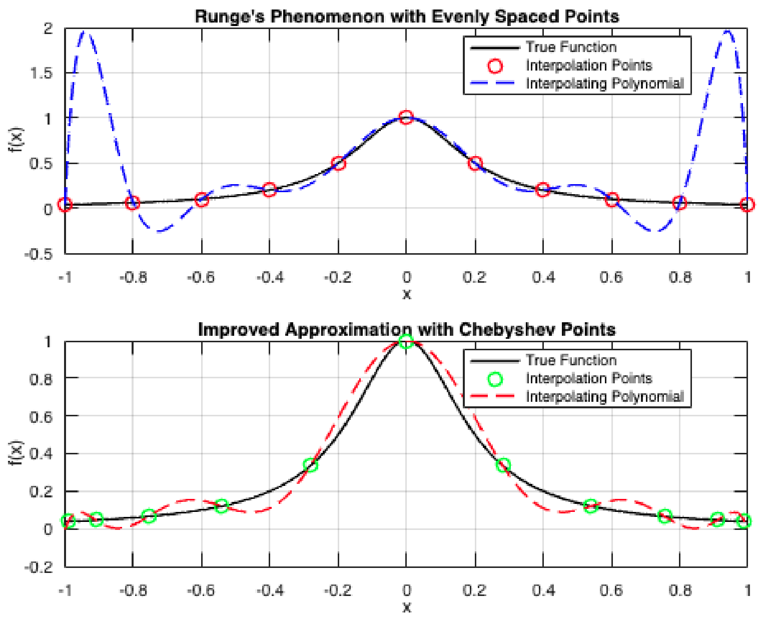
2. **Randomly distributed points**

For B:

- These can cause **Runge's Phenomenon**

1.4.4 Runge's Phenomenon

Occurs when solving for a polynomial interpolate by using evenly spaced points. We find Runge's Phenomenon when the estimate gets increasingly inaccurate as we approach the endpoints.



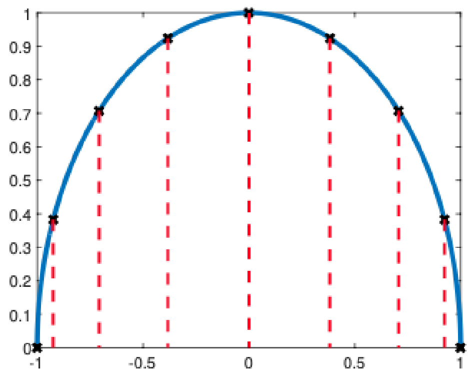
This is caused by the edge terms in (1.4.3) for evenly spaced points, the error for each position is given by $(x - x_i)$. Since we're starting at the center and working outwards, our error will increase as me move that way.

Can be combated using <u>Chebychev points</u>.

1.4.5 Chebychev points

points are defined by sed on cosine function. Often derived as projections of equally spaced points on a semicircle onto the x axis:

$$x_i = \cos\left(\pi \frac{2i+1}{2n+2}\right), \qquad i = 0, \ldots, n$$



1.5 Splines

A method of polynomial interpolation which divides a dataset into smaller polynomials. Splines mitigate the Runge's Phenomenon by using lower-degree polynomials on each sub-interval.
A **Cubic Spline** is a spline with a cubic polynomial.

$$S_i(x) = y_i + b_i(x - t_i) + c_i(x - t_i)^2 + d_i(x - t_i)^3$$

Match function values

$$S_i(t_i) = y_i$$
$$S_i(t_{i+1}) = y_{i+1}$$

Match first derivatives

$$S_i'(t_{i+1}) = S_{i+1}'(t_{i+1})$$

Match second derivatives

$$S_i''(t_{i+1}) = S_{i+1}''(t_{i+1})$$

We set the end points to be 0 for a natural cubic spline; $S_0''(t_0) = 0$, and $S_{n-1}''(t_n) = 0$.

Advantages

- Computationally efficient and stable
- splines avoid the oscillations near endpoints that show up with high degree polynomials

1.6 Least Squares Fitting

Least squares fitting is finding the best-fit solution to a system of equations that may not have an exact solution. We do so by minimizing the sum of squares of residuals.

$$\min_x ||Ax - b||_2^2$$

Where A is the coefficient matrix, b the observed values matrix, and x the unknowns vector.

1.6.1 Normal Equations

Derives the solution by solving

$$A^T Ax = A^T b$$

1. Solve $A^T A$
2. Solve $A^T b$
3. solve using a direct method (Gauss elim)

Normal Eqs are straightforward, but prone to ill-conditioning since we have to calculate $A^T A$.

1.6.2 QR Factorization

Decomposes A into A (orthogonal) and R (upper triangular), s.t.

$$\begin{aligned} Ax &= b \\ QRx &= b \end{aligned}$$

1. Find $Q^T b$
2. Simplify to $Rx = Q^T b$
3. Solve for x using back substitution

More numerically stable, especially when A has well-spaced singular values.

1.6.3 Singular Value Decomposition (SVD)

$$A = USV^T$$

Where U is a $m \times n$ orthogonal matrix, S is a $m \times n$ diagonal, with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$, and V is a $n \times n$ orthogonal matrix.

1. **Decompose** A
   Compute U, S, and V by finding the eigenvalues and eigenvectors of $A^T A$ for V, and AA^T for U.
2. **Transform**
   Substitute SVD into $\min ||Ax - b||_2^2$.
   $$\min ||Ax - b||_2^2 = \min ||U^T b - SV^T x||_2^2$$
3. **Solve**
   Let $c = U^T b$ and $y = V^T x$, we have the new residual
   $$r = ||c - Sy||_2^2$$
   For nonzero singular values, σ_i, set $y_i = \frac{c_i}{\sigma_i}$. For zero singular values, set $y_i = 0$.
4. **Back transform**
   $$x = Vy$$
   Then, the final value will be given by
   $$x = \sum_{i=1}^{r} \frac{c_i}{\sigma_i} v_i$$
   Where $c_i = u_i^T$ and v_i are the columns of V.

Advantages

- handles ill-conditioned matrices better than normal equations
- avoids forming $A^T A$, which makes it safer to avoid catastrophic cancellation.

1.6.4 Iterative Methods

Iterative Methods solve $A\overrightarrow{x} = \overrightarrow{b}$ **approximately** by iterating until a stopping condition is met.
Ideal for large, sparse matrices or when exact solutions are unnecessary

1. **Jacobi Method**

   Approximate A^{-1} with diagonal D

   $$x^{(k)} = D^{-1}(b - (L+U)x^{(k-1)})$$

   A **must be SPD or diagonally dominant to converge**

   Runs in $O(n^2)$ for a dense matrix, $O(mn)$ for a sparse matrix.

2. **Gauss-Seidel Method**

   Approximate A^{-1} with $Q = D - L$

   $$x^{(k)} = (D-L)^{-1}(b - Ux^{(k-1)})$$

   Similar to Jacobi, but applies the updated values immediately instead of waiting to the next iteration.

3. **Conjugate Gradients Method**

   Solve $A\overrightarrow{x} = \overrightarrow{b}$ for SPD A by minimizing $\phi(x) = \frac{1}{2}x^TAx - b^Tx$

   $$\begin{aligned} r_0 &= b - Ax_0 && \text{1. initialize } r \\ s_0 &= r_0 && \text{2. initialize } s \\ a_k &= \frac{r_k^T r_k}{s_k^T A s_k} && \text{3. Iterate} \\ x_{k+1} &= x_k + a_k s_k \\ r_{k+1} &= r_k - a_k A s_k \\ \beta_k &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\ s_{k+1} &= r_{k+1} + \beta_k s_k \end{aligned}$$

   Runs in $O(n^2)$ time per iteration.

4. **Successive Over-Relaxation (SOR)**

   Add a relaxation param $\omega > 1$ to Gauss-Seidel

   $$x^{(k)} = x^{(k-1)} + \omega(D-L)^{-1}r^{(k-1)}$$

   Generally, $\omega = 1$ is gauss-Seidel. $\omega > 1$ accelerates convergence, and $\omega < 1$ is **Successive Under-relaxation** - can be used for stability but doesn't converge quicker.

1.6.5 Overdetermined Systems

Systems with more equations than unknowns. Tall, Skinny Matrices.
An overdetermined system cannot have an exact solution. Your solution will always include at least 1 unknown.

1.6.6 Condition Numbers

$$cond(A) = \frac{\sigma_1}{\sigma_r} \qquad \text{Regular formula}$$

$$cond(A) = ||A||_2 \cdot ||A^{-1}||_2 \qquad \text{Norm-based method}$$

Where σ_1 is the largest singular values, and σ_r the smallest non-zero singular value.

A well-conditioned matrix is found when $cond(A)$ is small. This means that small perturbations in A or b result in small changes to the solution, x.

An ill-conditioned matrix is found when $cond(A)$ is large. Here, small perturbations in A or b result in large changes to the solution, x.

1.7 Finding Eigen Values and Eigenvectors

Eigenvalues, (λ), are used when solving ODEs. Derived from the coefficient matrix of a linear system, they represent solutions to the characteristic equation of the matrix.

Eigenvectors give directions in which the transformations described by the matrix act by scaling vectors.

1.7.1 Characteristic Polynomial

$$det(A - \lambda I)$$

Where λ is the eigenvalue, and I the identity matrix.
The rots of the characteristic polynomial are the eigenvalues.
Eigenvectors are found by solving the following for each λ.

$$(A - \lambda I)v = 0$$

1.7.2 Power Method

Iteratively compute eigenvectors and eigenvalues. We start with an initial (arbitrary) guess for the eigenvector associated with the largest eigenvalue. We then refine it using

$$x_{k+1} = Ax_k$$

After each step, we normalize the vector again to prevent overflow or underflow.

1.7.3 Inverse Power Method

Same as the **power method**, but finds the smallest eigenvalue. Just replace A with A^{-1}.

$$x_{k+1} = A^{-1}x_k$$

1.7.4 Singular Value Decomposition (SVD)

This is somewhere else idk where it should be

1.7.5 Diagonalization

For "nice" matrices (like a normal matrix), we find an orthogonal matrix V, and diagonal matrix D s.t.

$$A = VDV^T$$

where cols of V are eigenvectors and diagonal entries of D are eigenvalues.

1.8 Numeric Differentiation

1.8.1 Finite Differences

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \qquad \textbf{Forward Difference}$$
$$f'(x) \approx \frac{f(x) - f(x-h)}{h} \qquad \textbf{Backward Difference}$$
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{h} \qquad \textbf{Centered Difference}$$

Centered Difference has the best accuracy.

1.8.2 Taylor Series

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \cdots + \frac{h^n}{n!}f^{(n)}(x) + R_n$$
$$R_n = \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(c)$$

where h is the step size, $f'(x), \ldots$ are the derivatives, and R_n is the remainder representing the error when truncating.

Errors arise from the growing truncating error because of floating point arithmetic.

1.8.3 Numeric Differentiation Error

Forward and backward difference methods have an error of $O(h)$ since we accumulate error at each step in the R_n. Centered difference has an error of $O(h^2)$ since we calculate the forward and backward values each time.

1.8.4 Richardson Extrapolation

Richardson extrapolation is a method to improve accuracy in our numeric differentiation methods by combining approximations with different step sizes.

Just know that we make the problem more complex in exchange for more accuracy.

Basically just taking any of the above methods and making it more accurate by increasing the complexity (num steps etc.)

1.9 Integration Methods

1.9.1 Trapezoid Rule

When $n = 1$, we approximate using

$$\frac{(b-a)}{2}[f(a) + f(b)]$$

Costs $O(n)$ runtime. Each step takes a constant $O(1)$ runtime, so we're bounded by how many sub-intervals, n, we have.

Error

$$E_r = -\frac{(b-a)h^2}{12}f''(\epsilon)$$

Where ϵ is some unknown point in $[a, b]$, and $h = b - a$

Degree-1 Precision

1.9.2 Composite Trapezoid Rule

When $n > 1$, we have the **Composite Trapezoid Rule**;

$$\int_a^b f(x)dx \approx \frac{h}{2}[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)]$$

Where $h = \frac{b-a}{n}$ is the step size
$x_i = a + ih$ are the sub-interval points

(Essentially just breaking up into more sub-intervals to increase accuracy)

Costs $O(n)$. We evaluate $n + 1$ function values across the n sub-intervals.

1.9.3 Simpson's Rule (Simpson's $\frac{1}{3}$)

The standard **Simpson's Rule** applies to $n = 2$

$$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$$

Where $h = \frac{b-a}{n}$
$x_i = a + ih$

Costs $O(1)$ for any given sub-interval, since we just compute 3 values at each one; $f(a)$, $f(\frac{a+b}{2})$, and $f(b)$.

Error

$$E_r = -\frac{(b-a)h^4}{90}f^{(4)}(\epsilon)$$

Where $h = \frac{b-a}{2}$

Degree-3 Precision

1.9.4 Composite Simpson's Rule

For $n > 2$, we have the **Composite Simpson's Rule**

$$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a) + 4\sum_{\text{odd } i} f(x_i) + 2\sum_{\text{even } i} f(x_i) + f(b)]$$

(Essentially just breaking up into more sub-intervals to increase accuracy)
Costs $O(1)$ for any given sub-interval, since we just compute 3 values at each one; $f(a)$, $f(\frac{a+b}{2})$, and $f(b)$.

1.9.5 Simpson's $\frac{3}{8}$ Rule

Simpson's Rule, but with $n = 3$

$$\frac{(b-a)}{8}[f(a) + 3f(a+h) + 3f(b-h) + f(b)]$$

Costs $O(1)$ since we evaluate 4 function values at each step, in the same way as Composite Simpson's Rule.
Error

$$E_r = -\frac{(b-a)h^4}{80}f^{(4)}(\epsilon)$$

Where $h = \frac{b-a}{3}$

Degree-3 Precision

1.9.6 Boole's Rule

Uses Newton Cotes rules with $n = 4$, which gives a degree-4 fit;

$$\frac{(b-a)}{90}[7f(a) + 32f(a+h) + 12f(\frac{a+b}{2}) + 32f(b-h) + 7f(b)]$$

Runs in $O(1)$ time, since we just calculate 5 function values at each step.

Error

$$E_r = -\frac{2(b-a)h^6}{945}f^{(6)}(\epsilon)$$

Where $h = \frac{b-a}{4}$.

Degree-5 Precision

1.9.7 Gaussian Quadrature

Uses non-uniform points x_j and weights w_j;

$$\int_a^b f(x)dx \approx \sum_{j=1}^{n} w_j f(x_j)$$
$$w_j = \int_{-1}^{1} \prod_{k \neq j} \frac{x - x_k}{x_j - x_k} dx$$

Where w_j are calculated s.t. the formula is exact for polynomials up to degree $2n+1$.

Changing Interval $[a, b]$ to $[-1, 1]$

$$x(t) = \frac{b-a}{2} t + \frac{b+a}{2}$$
$$dx = \frac{b-a}{2} dt$$

Error
$2n+1$ precision for $n+1$ points.

1.9.8 Newton-Cotes Rules for Integration

Newton-Cotes integration rules use evenly spaced points on the interval $[a, b]$.

Name	n	Formula
Trapezoid	1	$\frac{(b-a)}{2}[f(a) + f(b)]$
Simpson's $\frac{1}{3}$	2	$\frac{(b-a)}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$
Simpson's $\frac{3}{8}$	3	$\frac{(b-a)}{8}[f(a) + 3f(a+h) + 3f(b-h) + f(b)]$
Boole's	4	$\frac{(b-a)}{90}[7f(a)32f(a+h)12f(\frac{a+b}{2}) + 32f(b-h) + 7f(b)]$

1.10 Normal Ordinary Differential Equations (ODEs)

1.10.1 Euler's Method

$$u_{i+1} = u_i + hf(t_i, u_i)$$

Where h is the step size.

Error

Error is $O(h^2)$ per step. We accumulate overall error as $O(h)$ since we take $\frac{1}{h}$ steps.

Stability

$$G_i = |1 - h\lambda|$$

For $u'(t) = -\lambda u(t)$. Stability is then $h\lambda < 2$.

1.10.2 Runge-Kutta Methods

1. **RK2a (Trapezoidal Rule)**

$$\begin{aligned} K_1 &= f(t_i, u_i) \\ K_2 &= f(t_i + h, u_i + hK_1) \\ u_i &= u_i + \frac{h}{2}(K_1 + K_2) \end{aligned}$$

   Local Error: $O(h^3)$
   Global Error: $O(h^2)$

   Stability

   Amplification factor and stability region are generally better than Euler.

2. **RK2b (Midpoint Rule)**

$$\begin{aligned} K_1 &= f(t_i, u_i) \\ K_2 &= f(t_i + \frac{h}{2}, u_i + \frac{h}{2}K_1) \\ u_i &= u_i + hK^2 \end{aligned}$$

   Local error: $O(h^3)$
   Global error: $O(h^2)$

   Stability

   More stability than RK2a, region of stability enclosed in curves derived from the quadratic solution.

3. **RK4 (Fourth-Order Method)**

$$\begin{aligned} K_1 &= f(t_i, u_i) \\ K_2 &= f(t_i + \frac{h}{2}, u_i + \frac{h}{2}K_1) \\ K_3 &= f(t_i + \frac{h}{2}, u_i + \frac{h}{2}K_2) \\ K_4 &= f(t_i + h, u_i + hK_3) \\ u_{i+1} &= u_i + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \end{aligned}$$

   Local error: $O(h^5)$
   Global error: $O(h^4)$

   Stability
   Highly stable, much larger stability region than the other methods. Amplification factor derived using higher-order terms.

1.10.3 Monte Carlo Integration

Does not require continuity or differentiability of the function. Also particularly effective for higher-dimensional integrations where deterministic methods struggle because of exponential growth in computation cost.

$$\int_D f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

where x_i are randomly generated samples.

Drawbacks

1. **Convergence Rate**

   We converge in $O(\frac{1}{\sqrt{n}})$ which is slower than our deterministic methods for low-dimensional methods.

   High Variance

   The method relies on randomness. The variance of the estimate can be both large and small for any size N, which is why it's better for large N when we expect it to take a long time anyway.

   Central Limit Theorem (CLT): guarantees that our error will converge as $N \to \infty$.

2 Midterm Review

2.1 Big O Notation

2.2 Errors

$$\begin{aligned} \textbf{Absolute Error} &: |x_{true} - x_{approx}| \\ \textbf{Relative Error} &: \frac{|x_{true} - x_{approx}|}{|x_{true}|} \\ \textbf{Residual } (r) &: |f(x_n) - 0| \\ &: b - A\overrightarrow{x} \end{aligned}$$

Residual - How close we are to solving the problem - Tolerance on output, backward error.

$$\begin{aligned} \textbf{Absolute Tolerance} &: |x_n - x_{n-1}| < \partial_x \\ \textbf{Relative Tolerance} &: |\frac{x_n - x_{n-1}}{b-a}| < \partial_x \end{aligned}$$

Where x_n is the current guess (midpoint of current bracket), and n_{n-1} is the previous guess (midpoint of prev. bracket).

2.2.1 Finite Precision

2.2.2 Machine Epsilon

The smallest value that can be added to 0 to get a non-zero result; $fl(1 + \epsilon_m) \neq 0$.

$$\begin{aligned} \textbf{Single Precision } \epsilon_m &= 2^{-23} \\ \textbf{Double Precision } \epsilon_m &= 2^{-52} \end{aligned}$$

2.3 Floating Point Arithmetic

$$\begin{aligned} fl(x) &= x(1+\epsilon) && \text{for } |\epsilon| < \epsilon_m \\ fl(x \bigcirc y) &= (x \bigcirc y)(1 + \epsilon_{\bigcirc}) && \text{for } +,\ -,\ *,\ \div \\ fl(x \bigcirc y) &= fl(y \bigcirc x) && \text{for } +,\ * \end{aligned}$$

Cancellation: Occurs when two numbers of very different magnitudes are added, or two of very close magnitudes are subtracted.

$$\begin{aligned} c &= a + b && \text{if } a >> b\text{, or if } b << a \\ c &= a - b && \text{if } a \approx b \end{aligned}$$

Catastrophic Cancellation: Large Cancellation caused by a single operation

2.3.1 Loss of precision

$$\beta^{-p} \leq 1 - \frac{y}{x} \leq \beta^{-q} \qquad x < y < 0$$

2.3.2 Loss of Signifigance

<u>**08-29-24 L04 Slide 14**</u>

2.4 Linear Solving

2.5 Symmetric Positive Definite (SPD)

- **non singular**
  If $x^T Ax > 0$, for all nonzero x then $\lambda > 0$ for all eigenvalues λ.
- **no pivoting**
  From the algorithm, dervice $|I_{k,j}| \leq \sqrt{a_{k,k}}$.
  Elements of L do not grow with respect to A.
- **Cholesky $O(\frac{n^3}{2})$ faster than LU $O(n^3)$**

2.6 Direct Methods

Direct methods solve $A\overrightarrow{x} = \overrightarrow{b}$ **exactly** (up to a given error, generally ϵ_m) by transforming the matrix systematically.
Ideal for small or moderate sized systems and/or when relatively perfect precision is needed.

2.6.1 Gaussian Elimination

Solves $A\overrightarrow{x} = \overrightarrow{b}$ by reducing A to upper triangular form U

1. Elim rows below pivot by subtracting multiple of the pivot row to 0 out non-pivots
2. back-substitute U to find $\overrightarrow{x}$

Cost: $O(n^3)$ to reduce A to U, and $O(n^2)$ for back substitution.

2.6.2 LU Factorization

Factorizes $A = LU$ where L is the lower triangular, and U the upper triangular.

1. Solve $L\overrightarrow{y} = \overrightarrow{b}$ via **Forward Substitution**
2. Solve $U\overrightarrow{x} = \overrightarrow{y}$ via **back-substitution**

Calculates L, U in $O(n^3)$, and solves for $\overrightarrow{b}$ in $O(n^2)$.

Ideal for when we need to solve $A\overrightarrow{x} = \overrightarrow{b}$ for multiple b, since after the first factorization step we can solve in $O(n^2)$ time for each successive run with the same matrix A.

2.6.3 Cholesky's Method

Only works for SPD Matricies

Factor $A = LL^T$ where L is lower triangular, and L **must have positive diag entries**.

1. Solve $L\overrightarrow{y} = \overrightarrow{b}$
2. Solve $L^T\overrightarrow{x} = \overrightarrow{y}$

Costs half the flops of LU, but still has a run time of $O(n^3)$ for factorization, $O(n^2)$ for solving.

2.6.4 LDL^T Factorization

Only works for Symmetric A

1. Factor $A = LDL^T$, where D is diag, L is lower triangular
2. Solve using L, D, L^T sequentially

similar cost to cholesky, but handles symmetric matricies.

2.7 Integration Techniques

Integration techniques **approximate definite integrals** $\int_a^b f(x)dx$. Best for evaluating integrals when exact solutions are difficult to compute in another way.