

Numerical Methods

Dr Richard J van Arkel

Content

- Numerical Calculus & Interpolation: 8 lectures
- Numerical Solutions to ODEs: 7 lectures
- Transforms: 3 lectures (depending on timetable next term)

Contents

Chapter 2: Numerical Integration Introduction	73
Chapter 2: Numerical Integration Convergence	80
Chapter 2: Numerical Interpolation Lagrange & Error Formula	97
Chapter 2: Numerical Interpolation Newton & Crude Error Analyses	118
Chapter 2: Numerical Interpolation Splines & 2D Matrix Methods	137
Chapter 2: Numerical Interpolation Interpolating Triangulated 2D Domains	157
Chapter 2: Numerical Integration More Accurate Methods (Simpsons & Gauss)	174
Chapter 2: Numerical Differentiation Methods and Errors	197
Chapter 3: Numerical ODEs Forward Euler, Heun and RK4	218
Chapter 3: Numerical ODEs Multistep Methods & Convergence	237
Chapter 3: Numerical ODEs Absolute Stability	256
Chapter 3: Numerical ODEs Systems of ODEs and Stiffness	269
Chapter 3: Numerical ODEs Higher Order & Oscillating ODEs	282
Chapter 3: Numerical ODEs Non-Linear ODEs	294
Chapter 3: Numerical ODEs Boundary Value Problems	301
Chapter 4: Transforms Fourier Transform and Its Properties	319
Chapter 4: Transforms Fourier Transform Pairs & Examples	334
Chapter 4: Transforms Laplace Transform	344

Help and Support

Please ask questions

- Rare to have a unique question so please do ask mid-lecture

One-to-one feedback and support

- Drop in sessions (Thursday, 1pm) – CAGB 652.
- During tutorials (Wednesday 9am).
- Blackboard forum – I will check on lecture days.
 - Will sometimes answer in the lecture, sometimes via the channel, depending on the question.
- Note: full worked solutions for all tutorial questions are made available immediately.

Useful Resources

I personally like (and used when I was an UG student):

- Erwin Kreyszig '*WIE Advanced Engineering Mathematics*'
- Wolfram Alpha
- Wikipedia

I know others who like:

- Teukolsky, Vetterling & Flannery '*Numerical Recipes - The Art of Scientific Computing*'
- Randall J. LeVeque '*Finite Difference Methods for Ordinary and Partial Differential Equations*'

You may also like:

- Glyn James '*Advanced modern engineering mathematics*'
- Something else entirely – understanding is personal, if you find someone who's writing style you like, stick with them.

Examinable Content

1) Is *this* going to be in the exam?

- I reserve the right to examine absolutely anything covered in lectures.
- I will examine understanding, not memory; formulae will be provided, explanations of notation will not be.
- The tutorial questions are devised to help you develop your understanding.
- A slide stating “Students should be able to” precedes each topic I lecture on.

2) Is *this* going to be in the progress test?

- Linda’s content and anything I cover up to and including the content on ODE Boundary Value Problems (likely meaning up to and including the lecture on Friday 1st Dec) could be included in the progress test.

The General Theme for Numerical Methods

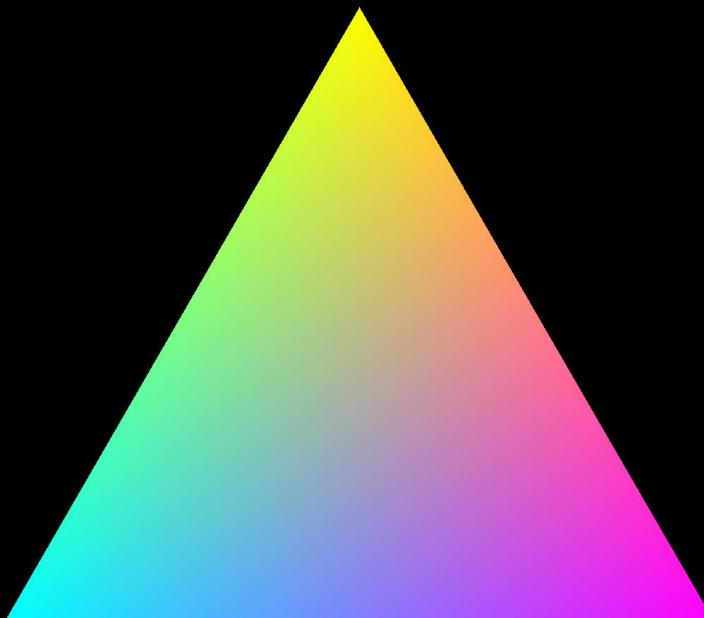
- Computers are AMAZING and better than us in everyway for repetitive tasks
 - Addition (and subtraction)
 - Multiplication (and division)
 - “*They eat for loops for breakfast*”
- However, they are also STUPID* and lack ability to problem solve and interpret
 - Computers can only add/multiply/loop so you need to translate maths for them
 - *maybe AI isn’t stupid – the terminator is coming?
- Numerical methods are techniques to simplify algebra to addition-multiplication-looping algorithms so that they can be solved by an amazing-stupid computer.
- A lot of the skill is in:
 - being able to select a method from many different options all of which could work but have pros and cons (*eerrr what now? When did maths start to sound like design & manufacture...?*)
 - being able to manipulate expressions that may look horrible, but are just addition/multiplication with some looping/iteration

Things I like about Numerical Methods

- It's learning how to breakdown complex problems into simpler steps
 - Problem solving ability, and the way of thinking are transferable skills
- It's learning how to get a computer to do the work
- We can embrace our inner nerdiness together twice a week
 - We all know deep down that when applying to university we wrote something to the effect of "*I really enjoy maths*" on our personal statements...
- Is there anything better in life than settling down to some wholesome mathematics?

My LecFlixTM Boxset Trailer:

$$\ddot{\theta}_1 = \frac{m_2 g \sin \theta_2 \cos(\theta_1 - \theta_2) - m_2 \sin(\theta_1 - \theta_2) (l_1 \dot{\theta}_1^2 \cos(\theta_1 - \theta_2) + l_2 \dot{\theta}_2^2) - (m_1 + m_2) g \sin \theta_1}{l_1(m_1 + m_2 \sin^2(\theta_1 - \theta_2))}$$
$$\ddot{\theta}_2 = \frac{(m_1 + m_2)(l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + g \sin \theta_1 \cos(\theta_1 - \theta_2) - g \sin \theta_2) + m_2 l_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) \cos(\theta_1 - \theta_2)}{l_2(m_1 + m_2 \sin^2(\theta_1 - \theta_2))}$$



Things to be aware of throughout

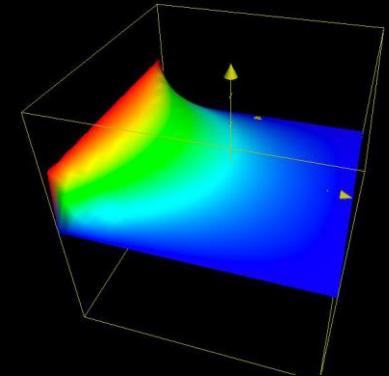
Computers are binary and have finite precision.

- Python and MatLab default storage is a 64-bit floating point number
- IEEE Std 754 defines this as $(-1)^s \times 2^e \times m$
 - 1 bit of storage is used for the sign ($s = 0 \rightarrow +ve, s = 1 \rightarrow -ve$)
 - 11 bits of storage are used for the exponent, e , between -1022 and 1023
 - Note $2^{11} - 1 = 2047$ meaning that the exponent range could be from -1023 to 1024
 - However 2^{-1023} and 2^{1024} are used for Nan and to track underflow/overflow (when a number is too small/large for the computer). For many programs $2^{1024} = \infty$
 - 52 bits are used for the significant digits. The achieved precision is 53 bit as the numbers are commonly normalised with the first digit assumed.

Things to be aware of throughout

Practical implications of finite calculation storage in computers are:

- Numbers are stored to ~ 16 significant digits: $\log_{10}(2^{53})$
- The smallest/largest possible number is $\sim 10^{\pm 308}$ ($1.999 \dots \times 2^{1023}$)
 - E.g. Next term, this will affect a plot you do with Monica.
- Decimals you think are exact, are not exact in the computer. E.g. consider 0.1
 - In base 10 (our world): this is 1×10^{-1} and is exact
 - In floating point binary, it has to be calculated with exponents of 2
 - E.g., if we used 2^0 for our significant digits: $2^0 \times 2^{-N} = 1 \times 2^{-N}$. But $2^{-3} = 0.125$ and $2^{-4} = 0.0625$, so we can't get 0.1 exactly.
 - To get a better approximation, we can use more significant digits. The best approximation of 0.1 in IEEE 754 double precision is $7205759403792794 \times 2^{-56}$ (i.e. an integer between 2^{52} and 2^{53} multiplied by an exponent of 2).
- Don't worry if this is confusing, computers are confusing at a fundamental level: everything they do is ultimately done with 1s and 0s... Ugh... Headache.



Things to be aware of throughout

- Round off errors (errors due to finite precision and storage of numbers during computation).
 - Overflow/underflow above max/min numbers ($\sim 10^{\pm 308}$)
 - Limited storage for significant digits (e.g. will affect numerical differentiation)
 - Subtraction can lead to loss of significant digits, for example consider the root of the following equation, using 4 significant digits for the computation:
$$x^2 - 20x + 1 = 0$$
$$x = 10 \pm \sqrt{99}$$
$$x = 10 \pm 9.950$$
$$x = 19.95 \text{ & } 0.05$$
- Truncation errors (errors due to the mathematical approximations we will derive).
 - Good practise is to accompany all numerical results with an error estimate
- Stability
 - Stable: small changes in initial data lead to small changes in the final result

Chapter 2: Numerical Calculus & Interpolation

Dr Richard J van Arkel

$$y = \frac{b-a}{b+a} \cdot \ln\left(\frac{b+a}{b-a}\right) \cdot \frac{1}{x^2}$$

Overview

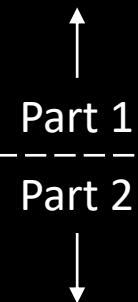
- Numerical integration part 1: the Trapezium Rule
- Numerical interpolation: Lagrange, Newton, Splines, 2D, 3D
- Numerical integration part 2: error estimation and improved accuracy
- Numerical differentiation: forwards, backwards, central

Chapter 2: Numerical Integration | Introduction

Numerical Integration Overview

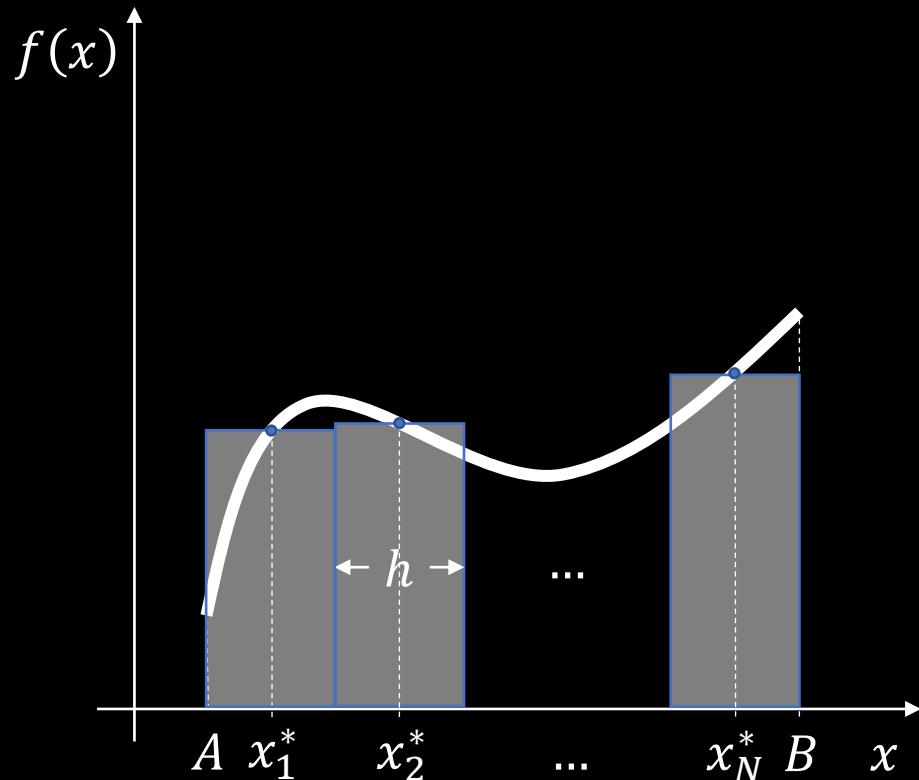
Students should be able to:

- Divide an integral into appropriate subintervals and predict whether it will converge or diverge when evaluated numerically
- Numerically calculate integrals using different schemes:
 - Riemann Sums (derive & apply)
 - Trapezium Rule (derive & apply)
 - Simpson's Rule (derive & apply)
 - Adaptive Integration (apply)
 - Gauss Quadrature (apply)
- Estimate numerical integration error



The Rectangular Rule (Middle Riemann Sum)

Aim: estimate the area under the curve. Solution: add up rectangular areas with heights = curve.

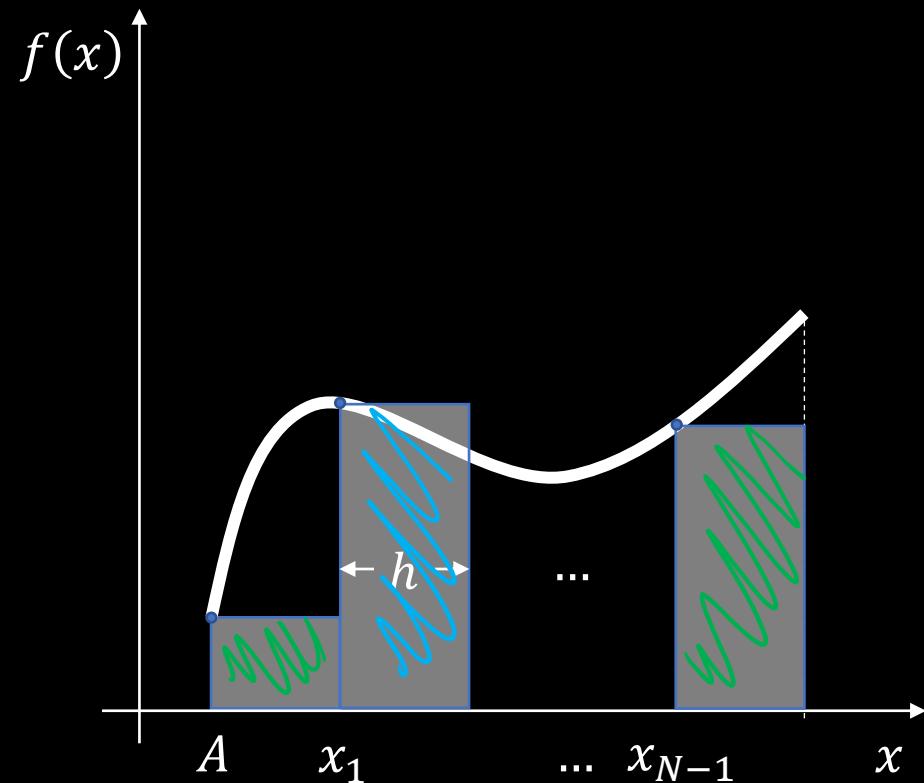


For a fixed subinterval width $h = \frac{B-A}{N}$

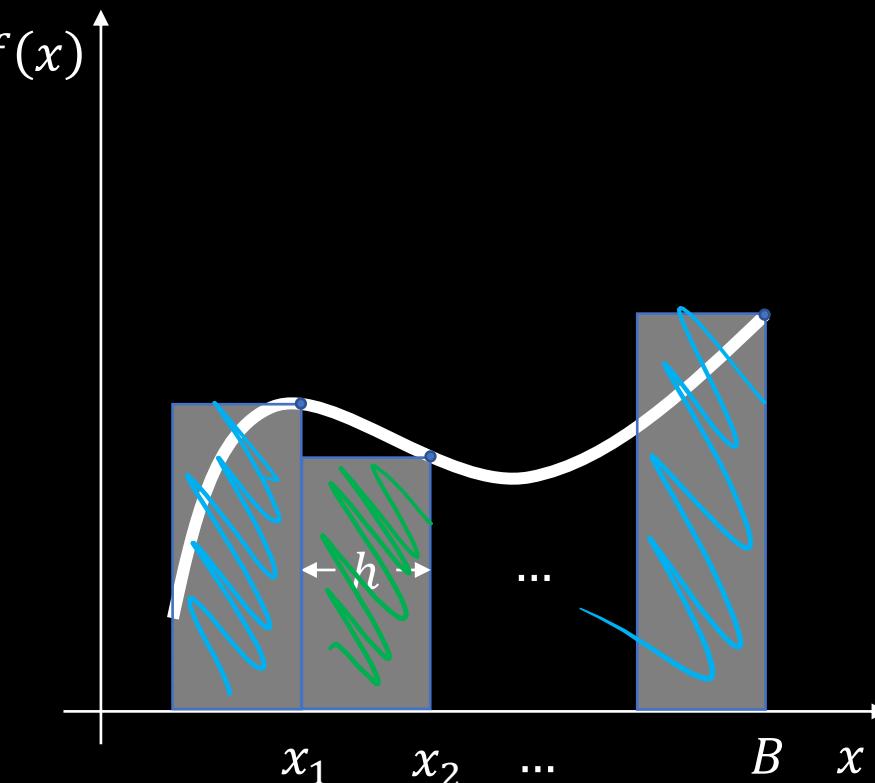
$$I = \int_A^B f(x)dx \approx h \sum_{n=1}^N f(x_n^*)$$

Where N is the number of subintervals, x_n^* is the midpoint of the n th subinterval and $f(x_n^*)$ is the value of the function evaluated at each midpoint (i.e. $f(x_n^*)$ is the height of each rectangle).

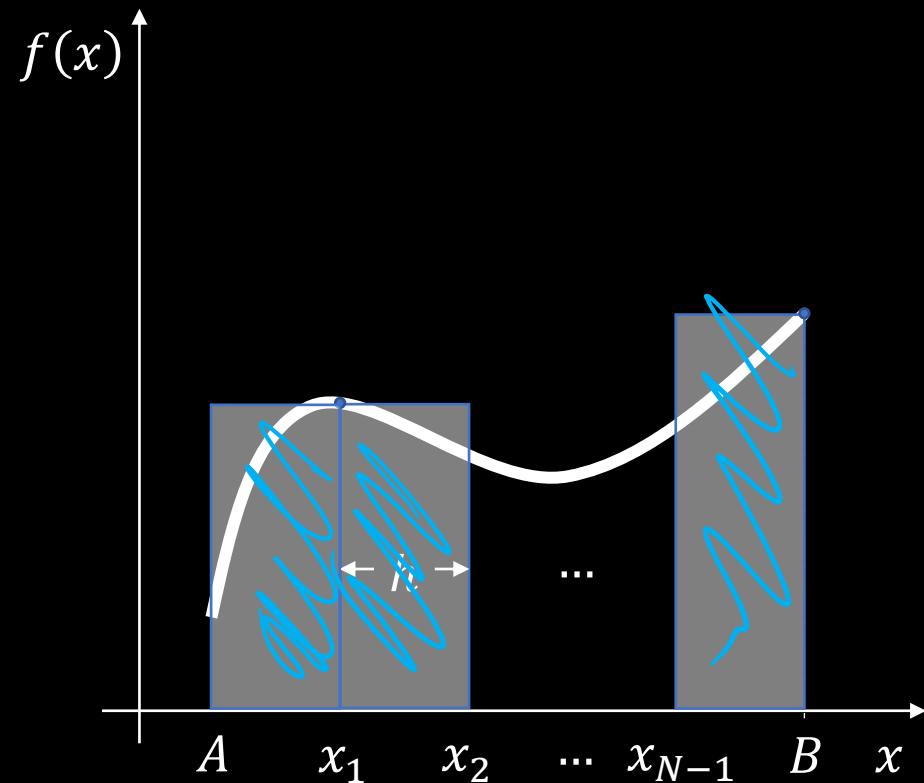
Left Riemann Sum



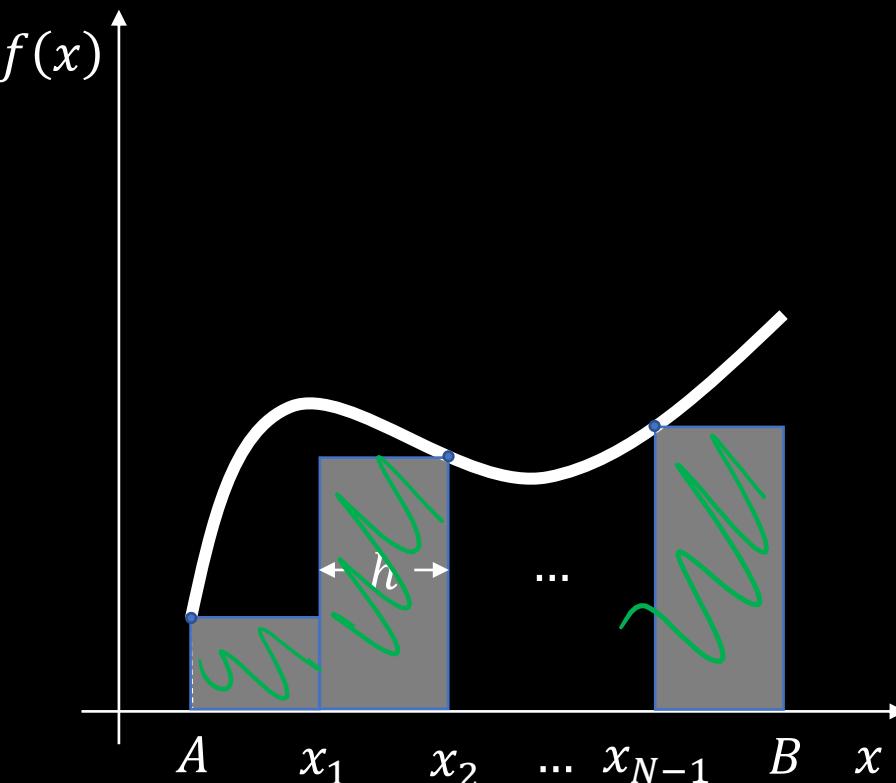
Right Riemann Sum



Upper Riemann Sum

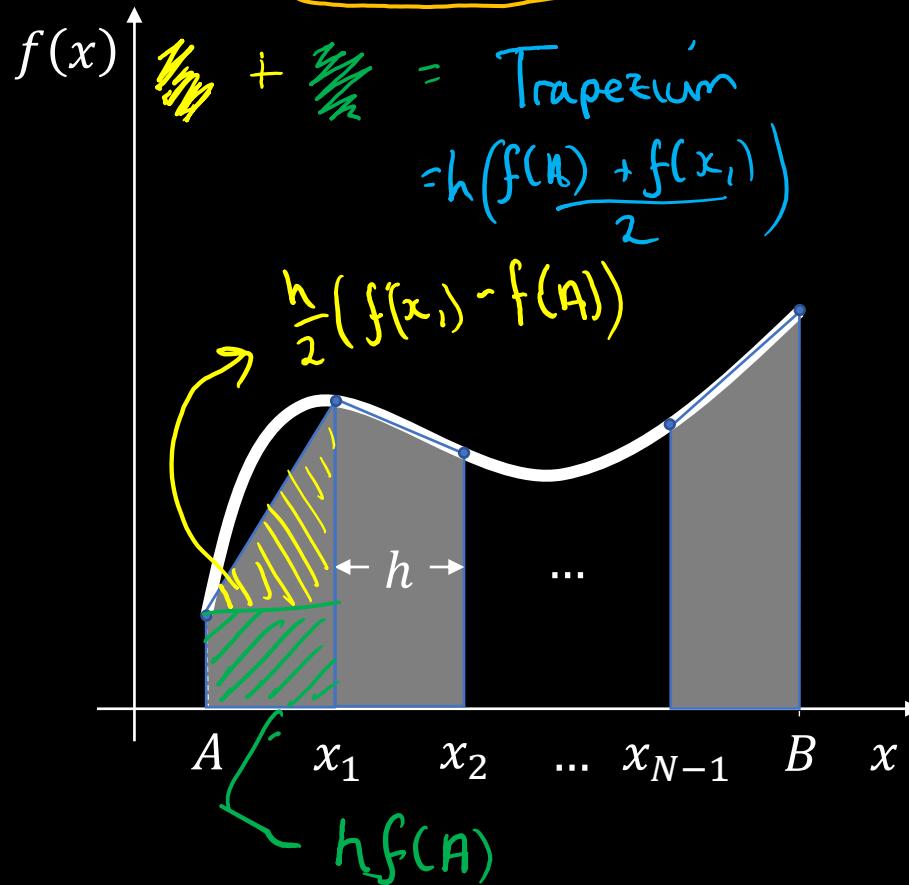


Lower Riemann Sum



The Trapezium Rule

$$I \approx h \left[\frac{f(A) + f(x_1)}{2} + \frac{f(x_1) + f(x_2)}{2} + \frac{f(x_2) + f(x_3)}{2} + \dots + \frac{f(x_{N-1}) + f(B)}{2} \right]$$



For a fixed subinterval $h = \frac{B-A}{N}$

$$I = \int_A^B f(x) dx \approx h \left[\frac{f(A)}{2} + \sum_{n=1}^{N-1} f(x_n) + \frac{f(B)}{2} \right]$$

Where N is the number of subintervals, x_n is the end point of the n th subinterval and $f(x_n)$ is the function evaluated at this point.

Compare to Left/Right Riemann sum in tutorial question.

Worked Example 1: Trapezium Rule

Integrate the following function (based on today's date) using the trapezium rule when:

$$I = \int_0^B \frac{1}{\sqrt{x^{17.10} + 2023}} dx$$

- a) $B = 2$ and with four subintervals

$$x = [0, 0.5, 1, 1.5, 2]; h = 0.5; I \approx 0.5 \left[\frac{f(0)}{2} + f(0.5) + f(1) + f(1.5) + \frac{f(2)}{2} \right]$$
$$= 0.5(0.0111 + 0.0222 + 0.0222 + 0.0181 + 0.0013)$$
$$= 0.0375$$

- b) $B = 2$ and with ten subintervals. What do you notice? Why?
0.0373
- c) With ten subintervals, what happens for large B ? Why?
 $h \frac{f(0)}{2}$

- d) Evaluate the integral for $B \rightarrow \infty$

$$0.0381$$

- e) What happens when $I = \int_0^B \frac{1}{\sqrt{x^{17.10} - 2023}} dx?$

@ $x^{17.1} = 2023$ $f \rightarrow \infty$ maybe $I \rightarrow \infty$?

$x < 2023 \Rightarrow$ complex

- f) What would have happened if today's lecture was on 1st October 2023 instead?

$$I = \int_0^B \frac{1}{\sqrt{x^{17.10} + 2023}} dx$$

Chapter 2: Numerical Integration | Convergence

Improper Integrals

$$I_1 = \lim_{B \rightarrow \infty} \int_A^B f(x) dx$$

$$I_2 = \lim_{A \rightarrow -\infty} \int_A^B f(x) dx$$

$$I_3 = \lim_{B \rightarrow C^-} \int_A^B f(x) dx$$

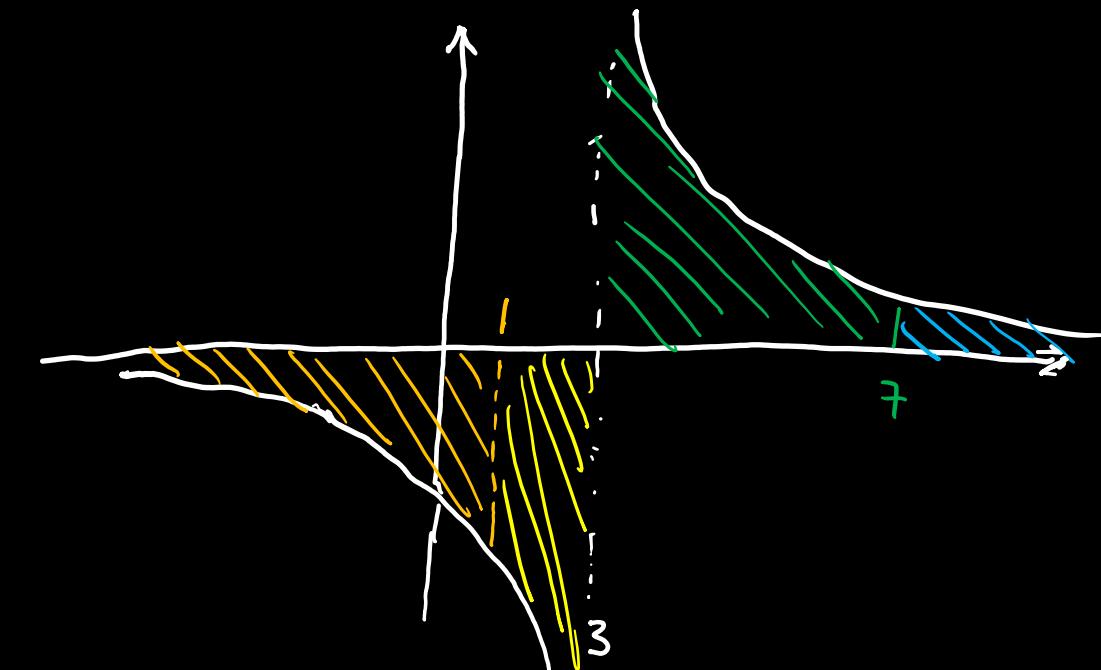
$$I_4 = \lim_{A \rightarrow C^+} \int_A^B f(x) dx$$

$$I = \int_{-\infty}^{\infty} \frac{1}{x-3} dx = \int_{-\infty}^1 \frac{1}{x-3} dx + \int_1^3 \frac{1}{x-3} dx + \int_3^7 \frac{1}{x-3} dx + \int_7^{\infty} \frac{1}{x-3} dx$$

Used when there is a vertical asymptote at $x = C$ within the integration interval:

$B \rightarrow C^-$ means B tending to C from less than C

$A \rightarrow C^+$ means A tending to C from greater than C

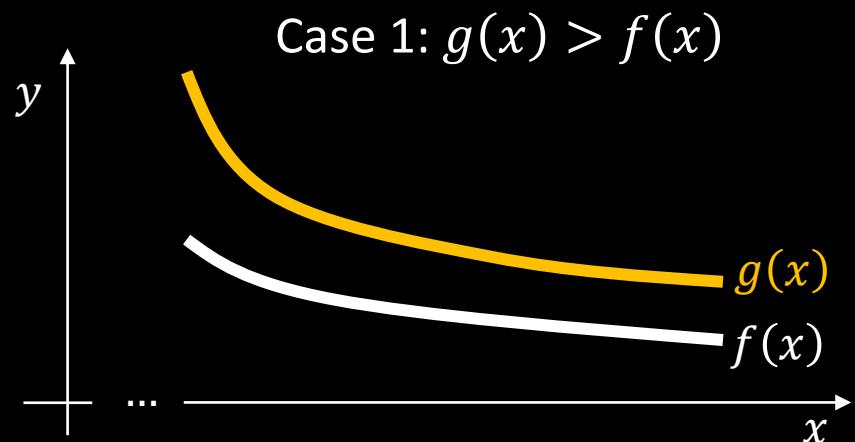


Integral Convergence: Standard Comparison Test (SCT)

Motivation: we want to know if our numerical integration of $f(x)$ will converge.

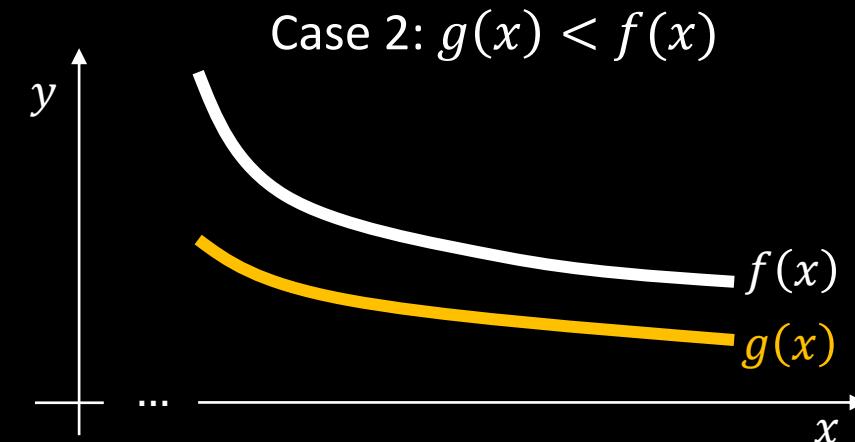
We can do this through testing if a larger function converges, or a smaller function diverges.

If $f(x)$ and $g(x)$ are positive and decreasing, then:



If $\int_A^\infty g(x) dx$ converges, $\int_A^\infty f(x) dx$ converges.

However, if $\int_A^\infty g(x) dx$ diverges we can't infer anything about $\int_A^\infty f(x) dx$ with SCT



If $\int_A^\infty g(x) dx$ diverges, $\int_A^\infty f(x) dx$ diverges.

However, if $\int_A^\infty g(x) dx$ converges we can't infer anything about $\int_A^\infty f(x) dx$ with SCT

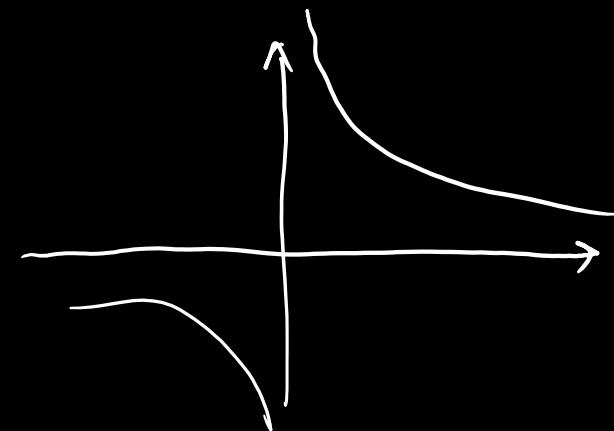
Worked Example 2: Improper Integral Convergence

$$\lim_{B \rightarrow \infty} \int_1^B \frac{1}{x^p} dx = \lim_{B \rightarrow \infty} \left[\frac{x^{1-p}}{1-p} \right]_1^B = \lim_{B \rightarrow \infty} \left\{ \frac{B^{1-p}}{1-p} + \frac{1}{p-1} \right\}$$

If $p > 1$ then B is on the denominator \Rightarrow as $B \rightarrow \infty$, $I \rightarrow \frac{1}{p-1}$

If $p \leq 1$ then B is on the numerator \Rightarrow as $B \rightarrow \infty$, $I \rightarrow \infty$

$$\text{If } p=1 \quad \int_1^\infty \frac{1}{x} dx = \left[\ln(x) \right]_1^\infty = \ln(\infty) - \ln(1) = \ln(\infty) = \infty$$



Abuse of notation is rather convenient... I'm going to adopt it going forwards, and I won't penalise it in an exam, e.g.

$$I = \int_1^\infty \frac{1}{x^{0.5}} dx = [2x^{0.5}]_1^\infty = \infty$$

$$I = \int_1^\infty \frac{1}{x^2} dx = [-x^{-1}]_1^\infty = 1$$

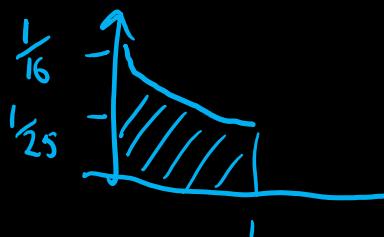
Worked Example 3

Does the following integral converge?

$$I = \int_0^\infty \frac{1}{(x^2+4)^2} dx$$

$$= \int_0^1 \frac{1}{(x^2+4)^2} dx + \int_1^\infty \frac{1}{(x^2+4)^2} dx$$

finite



let's compare to $g(x) = \frac{1}{x^4}$

is $g(x) > f(x)$?

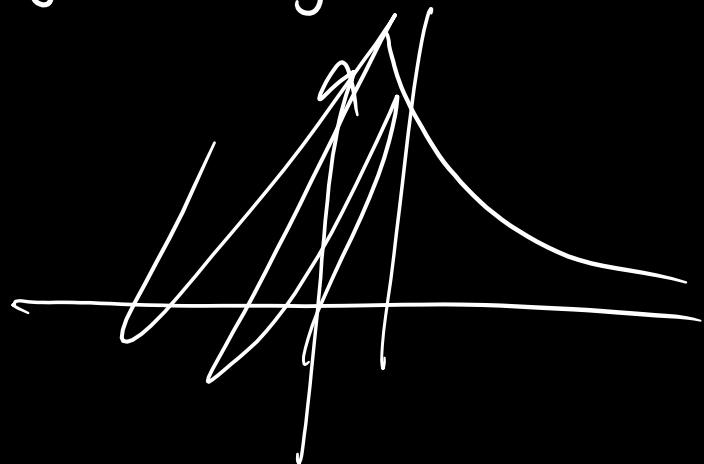
$$\frac{1}{x^4} > \frac{1}{(x^2+4)^2}$$

$(x^2+4)^2 > x^4$ THIS IS TRUE for $x > 1$

→ yes

we know $\int \frac{1}{x^4} dx$ converges as $x \rightarrow \infty$ as $P > 1$

∴ $\int_0^\infty \frac{1}{(x^2+4)^2} dx = \text{finite} + \text{convergent} \Rightarrow$ worth using Trapezium Rule



Worked Example 4

$$I = \int_3^\infty \frac{1}{(x^2 - 4)^2} dx$$

compare to $g(x) = \frac{1}{x^4}$

is $g(x) > f(x)$?

is $\frac{1}{x^4} > \frac{1}{(x^2 - 4)^2}$

multiply by $x^4(x^2 - 4)^2$

$$(x^2 - 4)^2 > x^4 \quad \text{NOT TRUE when } x > 3$$

∴ No...

Integral Convergence: Limit Comparison Test (LCT)

If we can't infer convergence with a SCT, we can try a LCT.

If $f(x)$ and $g(x)$ are positive and decreasing, and:

$$\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = C$$

where C is a positive constant, then we say that $\int_A^\infty f(x) dx$ and $\int_A^\infty g(x) dx$ behave the same way:

- Either both integrals converge
- Or both integrals diverge

Note, we can also apply the LCT to understand what happens as a function approaches a constant on the x-axis (where the value tends to infinity on the y-axis).

Worked Example 4 Revisited

$$\lim_{x \rightarrow \infty} \left\{ \frac{f(x)}{g(x)} \right\} = \lim_{x \rightarrow \infty} \left\{ \frac{\left(\frac{1}{(x^2 - 4)^2} \right)}{\left(\frac{1}{x^4} \right)} \right\}$$

multiply top & bottom by $x^4(x^2 - 4)^2$

$$\lim_{x \rightarrow \infty} \left\{ \frac{x^4}{(x^2 - 4)^2} \right\} \Rightarrow \text{L'Hopital's Rule}$$

Take x^2 out of the bracket

$$\lim_{x \rightarrow \infty} \left\{ \frac{x^4}{x^4 \left(1 - \frac{4}{x^2} \right)^2} \right\} = \lim_{x \rightarrow \infty} \left\{ \frac{1}{\left(1 - \frac{4}{x^2} \right)^2} \right\} = 1$$

$\Rightarrow f(x) \& g(x)$ behave the same way as $x \rightarrow \infty$ \therefore Integral of $f(x)$ converges as $x \rightarrow \infty$

Worked Example 1 Revisited

17th October 2023 function: $I_{17th} = \int_0^\infty \frac{1}{\sqrt{x^{17.10} + 2023}} dx$

- The natural comparison is that $\sqrt{x^{17.10} + 2023} > x^{8.55}$, therefore $\frac{1}{\sqrt{x^{17.10} + 2023}} < \frac{1}{x^{8.55}}$ for all $x > 0$
- We know that $\int_1^\infty \frac{1}{x^{8.55}} dx$ converges as power > 1 , therefore by SCT, $\int_1^\infty \frac{1}{\sqrt{x^{17.10} + 2023}} dx$ must also converge
- Also $\int_0^1 \frac{1}{\sqrt{x^{17.10} + 2023}} dx$ is finite. Therefore $\int_0^\infty \frac{1}{\sqrt{x^{17.10} + 2023}} dx = \text{convergent} + \text{finite} = \text{convergent}$
- We can estimate this limit with numerical integration.

1st October 2023 function: $I_{1st} = \int_0^\infty \frac{1}{\sqrt{x^{1.10} + 2023}} dx$

- LCT compared to $g(x) = \frac{1}{x^{1.10/2}} = \frac{1}{x^{0.55}} \approx \frac{1}{\sqrt{x^{1.1}}}$
- $\lim_{x \rightarrow \infty} \frac{\frac{1}{\sqrt{x^{1.10} + 2023}}}{\frac{1}{x^{1.10/2}}} = \lim_{x \rightarrow \infty} \frac{(x^{1.10})^{1/2}}{(x^{1.10} + 2023)^{1/2}} = \lim_{x \rightarrow \infty} \frac{1}{\left(1 + \frac{2023}{x^{1.10}}\right)^{1/2}} = 1$ which implies that $f(x) \sim g(x)$.
- We know that $\int_1^\infty \frac{1}{x^{1.10/2}} dx$ diverges as its power < 1 , therefore by LCT, I_{1st} must diverge.

Convergence of infinite series: integral comparison

If $f(x)$ is positive and decreasing:

$$\int_1^\infty f(x)dx \leq \sum_{n=1}^\infty f(n) \leq \int_1^\infty f(x)dx + f(1)$$

If $\int_A^\infty f(x)dx$ converges, then the infinite series $\sum_{n=A}^\infty f(n)$ converges

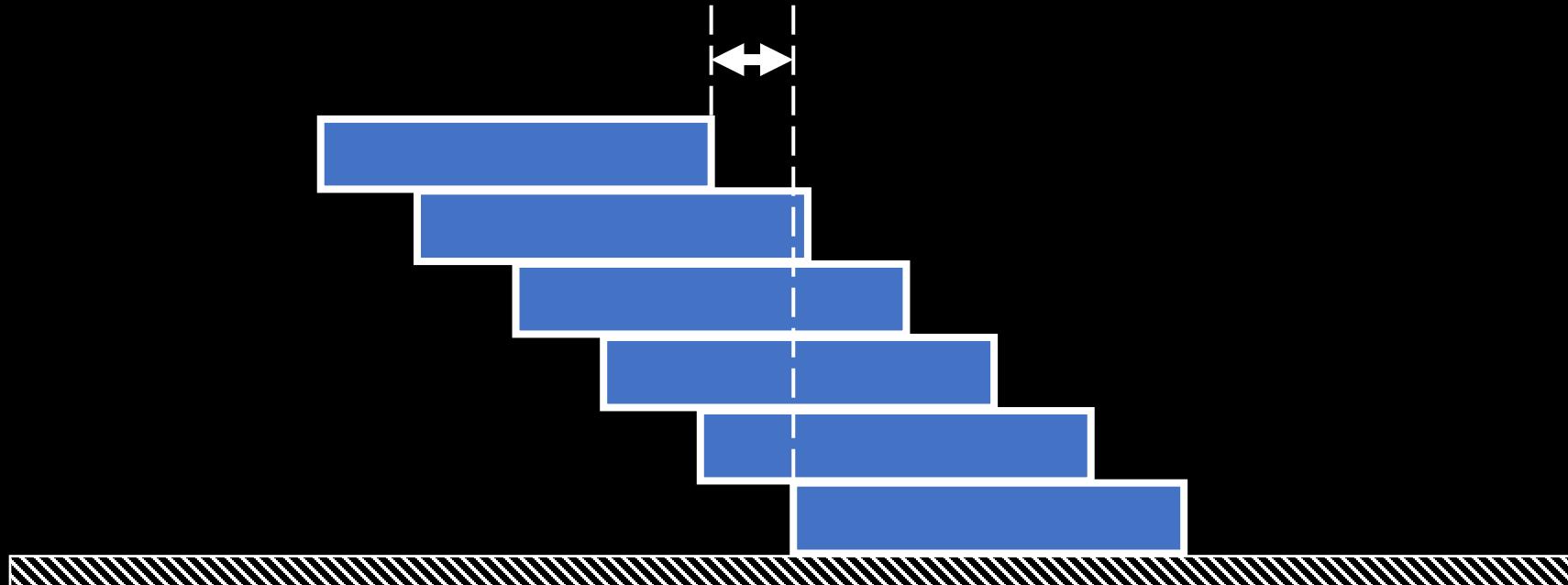
If $\int_A^\infty f(x)dx$ diverges, then the infinite series $\sum_{n=A}^\infty f(n)$ diverges.

In general the integral convergence is much easier to evaluate.

Proof using Upper/Lower Riemann sums (a tutorial question).

A thought experiment...

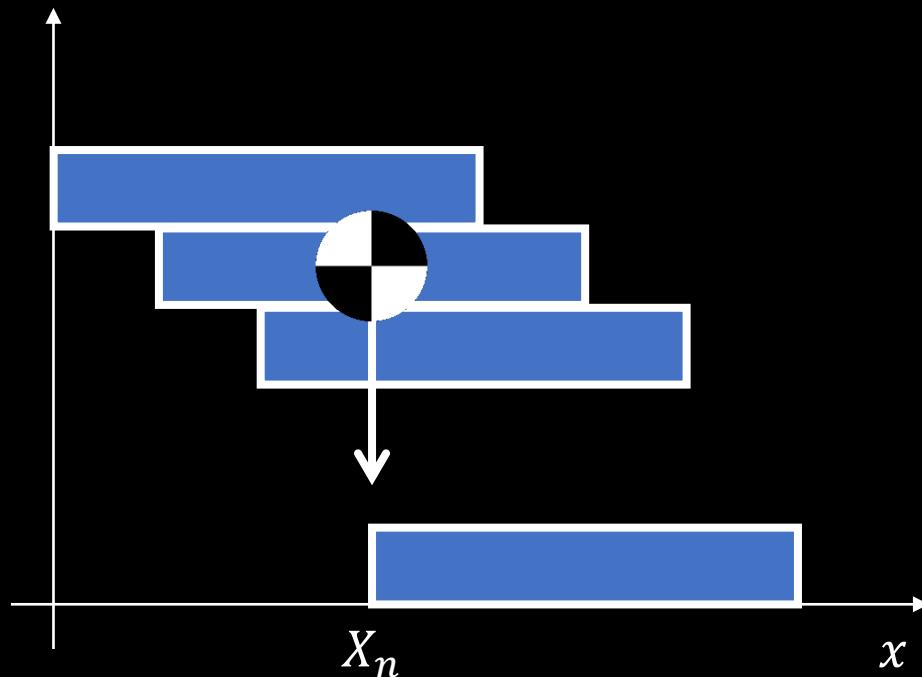
- How far can we go before the tower collapses? Is the below possible?



A thought experiment: hint #1

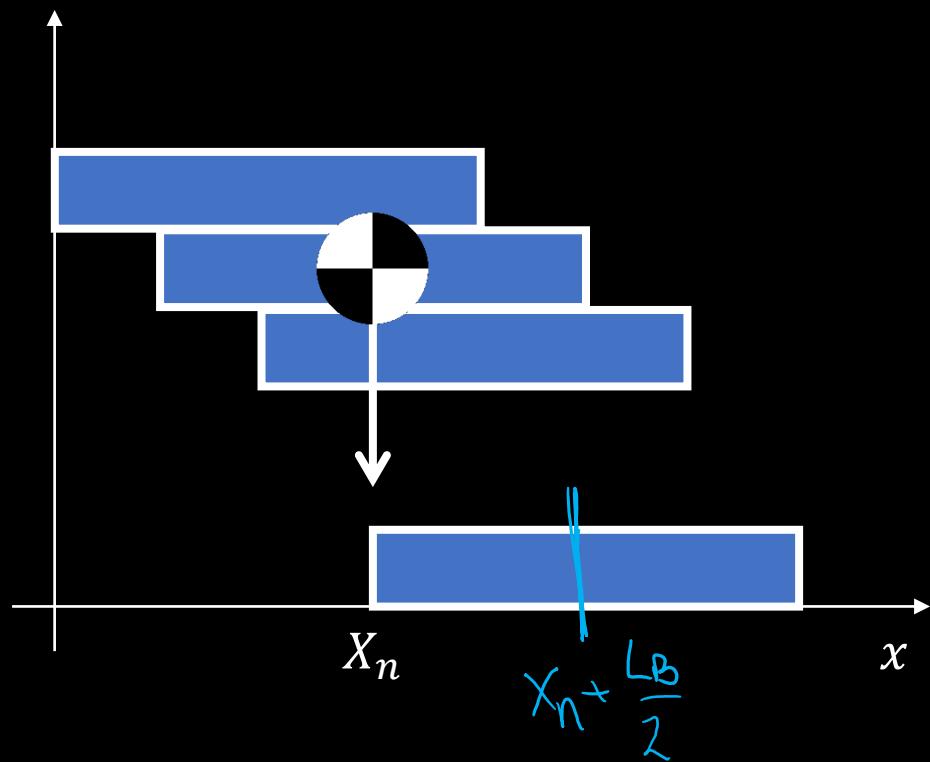
Add blocks underneath: it will never fall over if the centre of mass of the stack is over the new block.

Put the centre of mass of the stack at the edge of the new block to maximise the horizontal shift.



Consider how the centre of mass moves: what is X_{n+1} ? This will help you make a series...

A thought experiment: hint #2



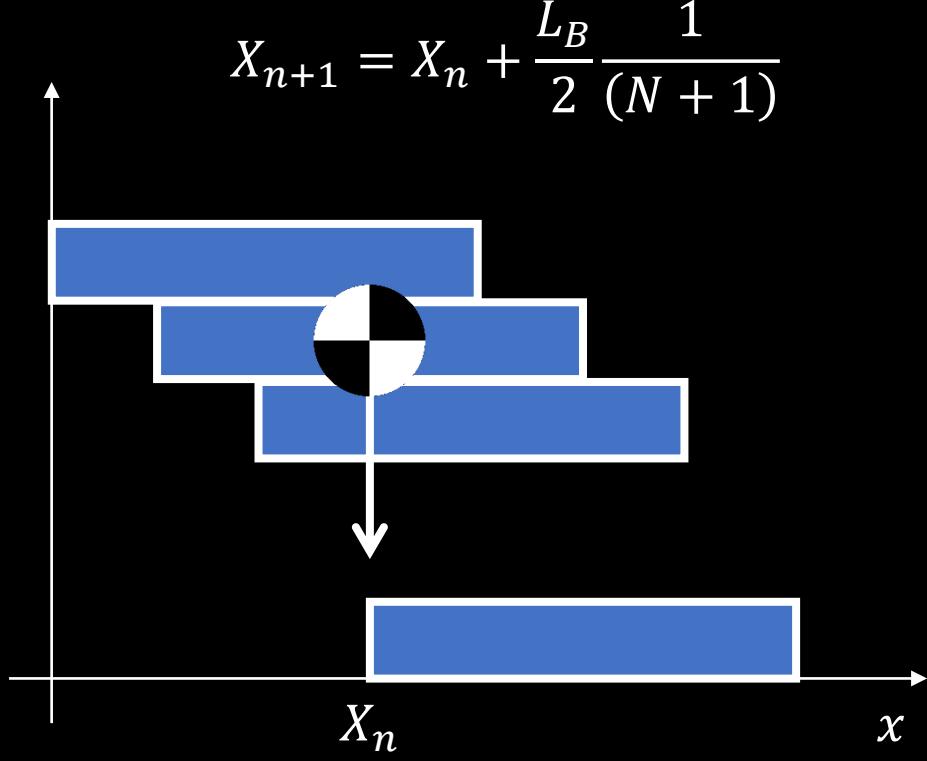
$$X_{n+1} = \frac{NW_B X_n + W_B \left(X_n + \frac{L_B}{2} \right)}{W_B(N+1)}$$

$$X_{n+1} = \frac{X_n(N+1) + \frac{L_B}{2}}{(N+1)} = X_n + \frac{L_B}{2} \frac{1}{(N+1)}$$

Where X_n is the centre of mass of a group of N blocks, W_B is the weight of a single block, and L_B is the length of a block

Number of Blocks	Centre of Mass (X_n)

A thought experiment: hint #3



Number of Blocks	Centre of Mass (X_n)
1	$\frac{L_B}{2}(1)$
2	$\frac{L_B}{2}\left(1 + \frac{1}{2}\right)$
3	$\frac{L_B}{2}\left(1 + \frac{1}{2} + \frac{1}{3}\right)$
4	$\frac{L_B}{2}\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}\right)$
N	$\frac{L_B}{2}\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{N}\right)$

$$X_N = \frac{L_B}{2} \sum_{n=1}^N \frac{1}{n}$$

A thought experiment: the answer

$$\int_1^\infty f(x)dx \leq \sum_{n=1}^{\infty} f(n) \leq \int_1^\infty f(x)dx + f(1)$$

Substituting in $f(x)$:

$$\lim_{N \rightarrow \infty} \frac{L_B}{2} \int_1^N \frac{1}{x} dx \leq \lim_{N \rightarrow \infty} \frac{L_B}{2} \sum_{n=1}^N \frac{1}{n} \leq \lim_{N \rightarrow \infty} \frac{L_B}{2} \int_1^N \frac{1}{x} dx + \frac{L_B}{2}$$

The integral is divergent and so the series is divergent: there is no limit to how far we can go! But...

Integrating:

$$\frac{L_B}{2} \ln(N) \leq X_N \leq \frac{L_B}{2} (\ln(N) + 1)$$

From the max value: $\frac{L_B}{2} (\ln(N) + 1) \geq X_N$, Therefore $N \geq e^{\left(\frac{2X_N}{L_B} - 1\right)}$

From the min value: $\frac{L_B}{2} \ln(N) \leq X_N$, Therefore $N \leq e^{\left(\frac{2X_N}{L_B}\right)}$

A thought experiment: the implications

$$e^{\left(\frac{2X_N}{L_B}-1\right)} \leq N \leq e^{\left(\frac{2X_N}{L_B}\right)}$$

If we want to extend past the start point, we need to shift the centre of mass $X_N = L_B$, then the number of blocks needed is:

$$2.7 = e^{(1)} \leq N \leq e^{(2)} = 7.4$$

What if we want to go the length of the lecture theatre bench? We need to shift the centre of mass $X_N = 12.5L_B$:

$$2.6 \times 10^{10} = e^{(24)} \leq N \leq e^{(25)} = 7.2 \times 10^{10}$$

If a block is 3 cm high, then the minimum height of the tower would be: 7.8×10^5 km, which is further than the distance to the moon.

Infinity is great for maths, but bad for the brain

Summary

- Approximate integrals by summing up areas of known shapes (e.g. rectangles, trapeziums, etc).
- There are no limitations on what you can integrate with these numeric methods...
...however, the computer will blindly integrate nonsense, giving a nonsense answer.
- Need to be aware of:
 - How the function behaves over the integration interval: is it positive? Is it decreasing? Are there any vertical asymptotes/discontinuities?
 - Limiting behaviour (convergence/divergence)
 - The influence of the subinterval width h and the integration interval size $[A, B]$
- Coming up:
 - Error estimation
 - More accurate methods
- But to get there we need to learn to interpolate

Chapter 2: Numerical Interpolation | Lagrange & Error Formula

Numerical Interpolation Overview

Students should be able to:

- Understand the notation of, and be able to apply different interpolation schemes:
 - Lagrange
 - Newton
 - Cubic Spline
- Calculate interpolation error:
 - Crude estimates
 - Error bounds
- Interpolate in 2D:
 - Using both sequential and matrix methods for regularly spaced nodes
 - Using triangles for irregularly spaced nodes

Interpolation Notation and Basic Concept

We choose to use polynomials as they are easy to differentiate and integrate, and because the error can often be reduced to a desired accuracy by increasing the degree of the polynomial (assuming that there are enough known points).

→ complex function / experimental data

For an arbitrary function $y = f(x)$, with $N + 1$ known points:

$$y_0 = f(x_0), \quad y_1 = f(x_1), \quad y_2 = f(x_2), \quad \dots, \quad y_N = f(x_N)$$

where the x coordinates $(x_0, x_1, x_2, \dots, x_N)$ are known as nodes.

The idea is to find an interpolating polynomial $p_n(x)$ such that:

$$p_n(x_0) = y_0, \quad p_n(x_1) = y_1, \quad p_n(x_2) = y_2, \quad \dots, \quad p_n(x_N) = y_N$$

where $n \leq N$. Thus $p_n(x_j) = f(x_j)$ for $j = 0, 1, 2, \dots, N$. I.e. the interpolating polynomial is equal to the function at the known points.

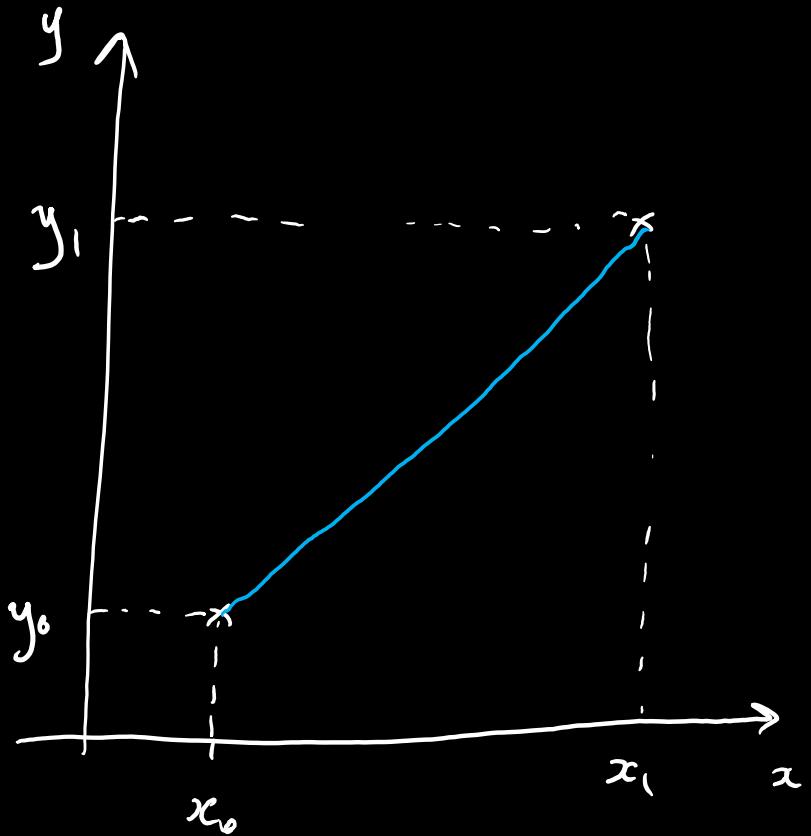
Lagrange Interpolation

- Multiply each known point, y_j , by a polynomial $L_j(x)$ and sum these polynomials to find the interpolation polynomial $p_n(x)$:

$$p_n(x) = \sum_{j=0}^n L_j(x) y_j$$

- Each polynomial $L_j(x)$ is:
 - also of the n th degree, with $n + 1$ nodes (known points)
 - has a value of 1 at the j th node (at $x = x_j$)
 - has a value of 0 at all the other nodes
 - Doing so ensures that $p_n(x_0) = y_0, p_n(x_1) = y_1, p_n(x_2) = y_2, \dots, p_n(x_n) = y_n$
- This becomes clearer with some examples...

Lagrange Interpolation Diagram



(linear \Rightarrow two known points
 \Rightarrow two Lagrange Polynomials)

$$\begin{aligned}L_0(x) &= 1 @ x_0 \\&= 0 @ x_1 \\&= \text{Something else inbetween}\end{aligned}$$
$$\begin{aligned}L_1(x) &= 1 @ x_1 \\&= 0 @ x_0 \\&= \text{Something else inbetween}\end{aligned}$$
$$p_1(x) = y_0 L_0(x) + y_1 L_1(x)\end{aligned}$$

Linear Lagrange Interpolation

- The functions can be defined as:

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}, \quad L_1(x) = \frac{x - x_0}{x_1 - x_0}$$

- Check: when $x = x_0$ then $L_0(x_0) = 1$, and $L_1(x_0) = 0$. When $x = x_1$ then $L_0(x_1) = 0$, and $L_1(x_1) = 1$.
- Therefore:

$$p_1(x) = \sum_{j=0}^1 L_j(x) y_j = L_0(x)y_0 + L_1(x)y_1 = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1$$

- This can be rearranged to the classical linear interpolation formulas you did at school.
- It is linear, as the Lagrange polynomials ($L_0(x)$ and $L_1(x)$) are linear, and hence the summed interpolation polynomial $p_1(x)$ is linear (degree $n = 1$). I.e. it can be rearranged into the form:

$$p_1(x) = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 = \frac{y_0 - y_1}{x_0 - x_1} x + \frac{(x_0 y_1 - x_1 y_0)}{x_0 - x_1} = mx + c$$

Quadratic Lagrange Interpolation

- As for Linear, we can write down quadratic expressions that are 0 or 1 at nodes x_0, x_1 and x_2 :

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = 1 @ x_0 ; = 0 @ x_1 \text{ or } x_2$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = 1 @ x_1 ; = 0 @ x_0 \text{ or } x_2$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = 1 @ x_2 ; = 0 @ x_0 \text{ or } x_1$$

- Defining these functions in this way means that at x_0 : $L_0(x_0) = 1, L_1(x_0) = 0$ and $L_2(x_0) = 0$. At x_1 : $L_0(x_1) = 0, L_1(x_1) = 1, L_2(x_1) = 0$. At x_2 : $L_0(x_2) = 0, L_1(x_2) = 0, L_2(x_2) = 1$.
- The quadratic interpolation polynomial is thus:

$$p_2(x) = \sum_{j=0}^2 L_j(x) y_j = L_0(x)y_0 + L_1(x)y_1 + L_2(x)y_2 = (\text{sub in the above})$$

- Note that $p_2(x)$ has the quadratic form $ax^2 + bx + c$ when this expression is expanded in full.

Generalising

$$\sum_{j=0}^n y_j = y_0 + y_1 + y_2 + \dots + y_n$$

$$\prod_{k=0}^n y_k = y_0 y_1 y_2 \times \dots \times y_n$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

- Note that for $L_1(x)$, the numerator contains all other x_k terms except for $k = 1$.
 - I.e. it contains nodes x_0 and x_2 , but does not contain node x_1 .
- The denominator is the same as the numerator, but with $x = x_1$.
- We could write this as:

$$L_1(x) = \prod_{\substack{k=0 \\ k \neq 1}}^2 \frac{(x - x_k)}{(x_1 - x_k)}$$

- Generalising for $L_0(x), L_1(x), L_2(x)$

$$L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^2 \frac{(x - x_k)}{(x_j - x_k)}$$

Lagrange Interpolation of the n th Degree

$$p_n(x) = \sum_{j=0}^n y_j L_j(x)$$

Where:

$$L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^n \frac{(x - x_k)}{(x_j - x_k)}$$

Or in condensed notation:

$$p_n(x) = \sum_{j=0}^n y_j \prod_{\substack{k=0 \\ k \neq j}}^n \frac{(x - x_k)}{(x_j - x_k)}$$

Worked Example 5

$$\left. \begin{array}{l} p_n(x) = \sum_{j=0}^n y_j L_j(x) \\ L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^n \frac{(x - x_k)}{(x_j - x_k)} \end{array} \right\}$$

Interpolate $f(x) = \sin(x)$ over the range $1 \leq x \leq 2$ using linear, quadratic and cubic Lagrange interpolation.

Linear: $\Rightarrow n=1 ; x_0=1 , x_1=2 ; y_0 = \sin(1) = 0.8415 , y_1 = \sin(2) = 0.9093$

Step 1) find my Lagrange polynomials

$$L_0(x) = \frac{(x - x_1)}{(x_0 - x_1)} = \frac{(x - 2)}{(1 - 2)} = 2 - x$$

$$L_1(x) = \frac{(x - x_0)}{(x_1 - x_0)} = \frac{(x - 1)}{(2 - 1)} = x - 1$$

$$\begin{aligned} p_1(x) &= y_0 L_0(x) + y_1 L_1(x) = 0.8415(2-x) + 0.9093(x-1) \\ &= 0.0678x + 0.7737 \end{aligned}$$

$$p_n(x) = \sum_{j=0}^n y_j L_j(x)$$

$$L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^n \frac{(x - x_k)}{(x_j - x_k)}$$

Worked Example 5

↳ new problem

Quadratic (using $x_0 = 1$, $x_1 = 1.5$, and $x_2 = 2$):

$n = 2$

$$j=0 \quad L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 1.5)(x - 2)}{(1 - 1.5)(1 - 2)} = 2x^2 - 7x + 6$$

$$j = 1 \quad L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{(x - 1)(x - 2)}{(1.5 - 1)(1.5 - 2)} = -4x^2 + 12x - 8$$

$$j = 2 \quad L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(x - 1)(x - 1.5)}{(2 - 1)(2 - 1.5)} = 2x^2 - 5x + 3$$

$$p_2(x) = \underbrace{f(1)(2x^2 - 7x + 6)}_{f(1.5)(-4x^2 + 12x - 8)} + \underbrace{f(1.5)(-4x^2 + 12x - 8)}_{f(2)(2x^2 - 5x + 3)}$$

$$= 0.8415(2x^2 - 7x + 6) + 0.9975(-4x^2 + 12x - 8) + 0.9093(2x^2 - 5x + 3)$$

$$= -0.4884x^2 + 1.533x - 0.2032$$

$$p_n(x) = \sum_{j=0}^n y_j L_j(x)$$

$$L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^n \frac{(x - x_k)}{(x_j - x_k)}$$

Worked Example 5

Cubic (using $x_0 = 1, x_1 = 4/3, x_2 = 5/3$ and $x_3 = 2$):

$$L_0(x) = \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)} = \frac{(x - 4/3)(x - 5/3)(x - 2)}{(1 - 4/3)(1 - 5/3)(1 - 2)} = -\frac{9x^3}{2} + \frac{45x^2}{2} - 37x + 20$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)} = \frac{(x - 1)(x - 5/3)(x - 2)}{(4/3 - 1)(4/3 - 5/3)(4/3 - 2)} = \frac{27x^3}{2} - 63x^2 + \frac{189x}{2} - 45$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)} = \frac{(x - 1)(x - 4/3)(x - 2)}{(5/3 - 1)(5/3 - 4/3)(5/3 - 2)} = -\frac{27x^3}{2} + \frac{117x^2}{2} - 81x + 36$$

$$L_3(x) = \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)} = \frac{(x - 1)(x - 4/3)(x - 5/3)}{(2 - 1)(2 - 4/3)(2 - 5/3)} = \frac{9x^3}{2} - 18x^2 + \frac{47x}{2} - 10$$

Note: it can sometimes be quicker/easier to not expand the numerator when using pen and paper

$$p_n(x) = \sum_{j=0}^n y_j L_j(x)$$

Worked Example 5

Noting that: $f(1) = 0.8415, f(4/3) = 0.9719, f(5/3) = 0.9954$ and $f(2) = 0.9093$

$$\begin{aligned} p_3(x) &= 0.8415 \left(-\frac{9x^3}{2} + \frac{45x^2}{2} - 37x + 20 \right) + 0.9719 \left(\frac{27x^3}{2} - 63x^2 + \frac{189x}{2} - 45 \right) \\ &\quad + 0.9954 \left(-\frac{27x^3}{2} + \frac{117x^2}{2} - 81x + 36 \right) + 0.9093 \left(\frac{9x^3}{2} - 18x^2 + \frac{47x}{2} - 10 \right) \end{aligned}$$

- If I evaluate with MatLab/Python precision:

$$p_3(x) = -0.01163x^3 - 0.4350x^2 + 1.454x - 0.1661$$

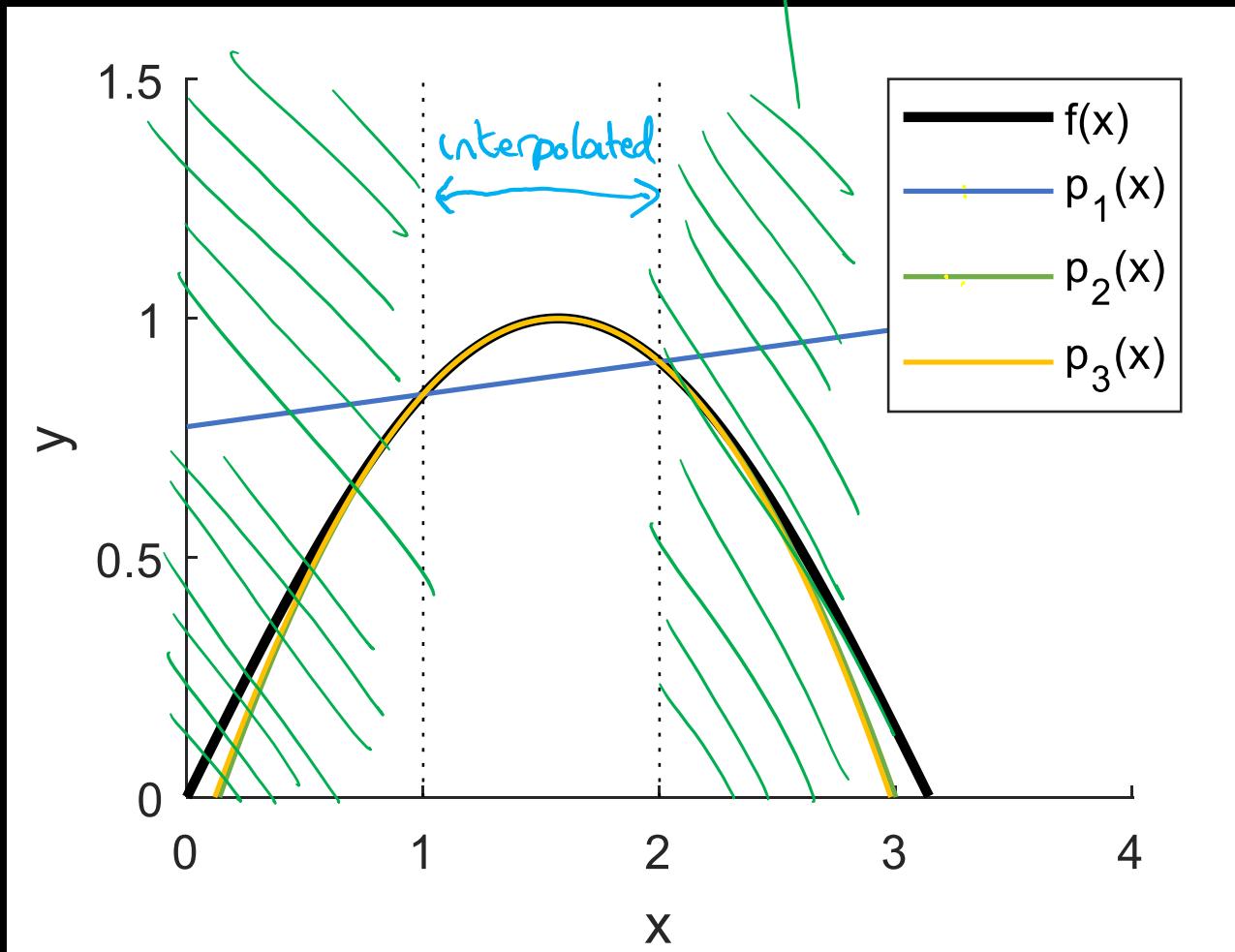
- If I evaluate with the 4 significant digits above:

$$p_3(x) = -0.01215x^3 - 0.43245x^2 + 1.4502x - 0.1641$$

- Working with 4 significant digits resulted in an answer only accurate to ~2 significant digits. – why?
- The moral: rounding errors can mess with your results in unexpected ways.

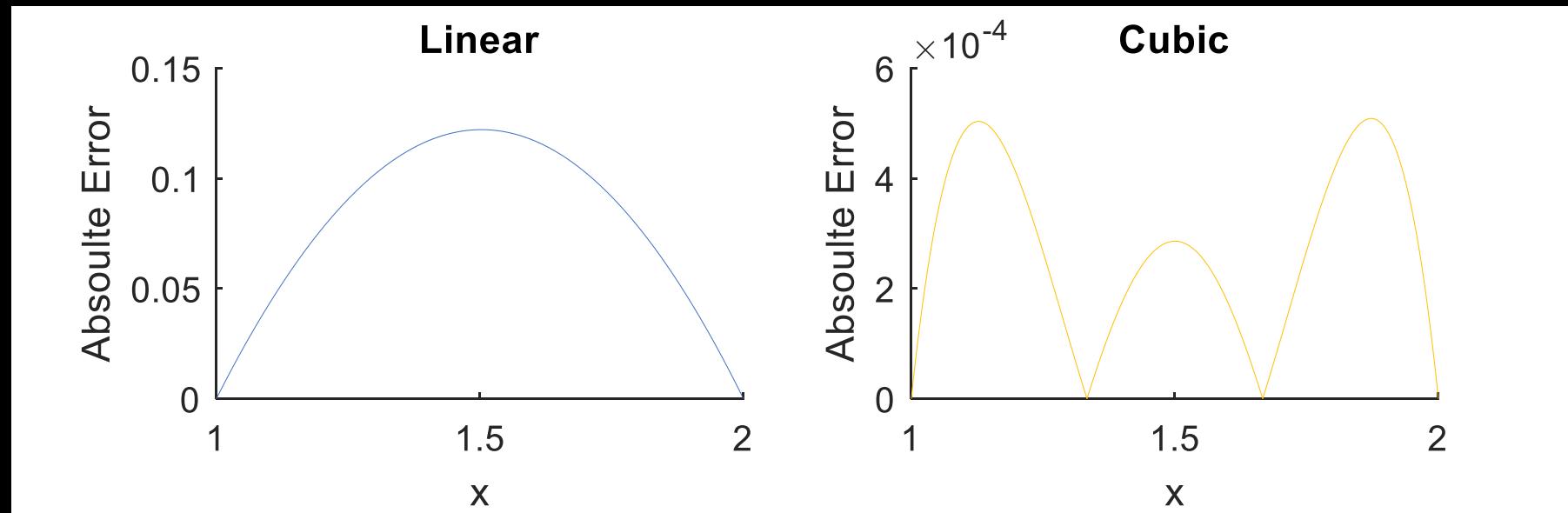
Worked Example 5

extrapolation
⇒ large errors



Interpolation Error

In WE5 we knew the true function ($\sin x$), so can calculate the interpolation error directly:



Error Analysis

For polynomial interpolation, the error can be calculated by a formula:

$$\epsilon_n(x) = f(x) - p_n(x) = \frac{f^{n+1}(\xi)}{(n+1)!} \prod_{j=0}^n (x - x_j)$$

However, this formula requires knowledge of $f(x)$, which must be differentiable $n + 1$ times, limiting its use. Moreover, the value ξ is often unknown, therefore it is most commonly applied to find the maximum possible error by finding:

$$\max_{x \in [A,B]} |\epsilon_n(x)| = \frac{\max_{x \in [A,B]} |f^{n+1}(x)|}{(n+1)!} \max_{x \in [A,B]} \left| \prod_{j=0}^n (x - x_j) \right|$$

It can also be used to determine the point spacing required for a desired accuracy.

Proof ← NOT IN EXAM

First we define a new function $F(t)$:

$$F(t) = f(t) - p_n(t) - \epsilon_n(t)$$

Where $\epsilon_n(t)$ is the product of a constant C and a product term that ensures it is zero at every node:

$$\epsilon_n(t) = C \prod_{j=0}^n (t - x_j)$$

Therefore, $F(t)$ is zero at $n + 2$ points: at each node ($t = x_j$), and also at an arbitrary evaluation point ' x ', (when $f(x) = p_n(x) + \epsilon_n(x)$). We now need to find the constant C . To do so we differentiate (with respect to t) $n + 1$ times:

$$F^{n+1}(t) = f^{n+1}(t) - p_n^{n+1}(t) - \epsilon_n^{n+1}(t)$$

We know that a $p_n^{n+1}(t) = 0$ (as its highest power was n) and $\epsilon_n^{n+1}(t) = C(n + 1)!$ as only the t^{n+1} term 'survives' differentiation $n + 1$ times. Therefore:

$$F^{n+1}(t) = f^{n+1}(t) - C(n + 1)!$$

We now use Rolle's theorem (a special case of the Mean Value theorem): between any two zeros of $F(t)$, there must be at least one zero of $F^1(t)$. As $F(t)$ has $n + 2$ zeros, this theorem implies $F^1(t)$ must have $n + 1$ zeros, it follows that $F^2(t)$ must have n zeros, $F^3(t)$ must have $n - 1$ zeros, and so on until we reach $F^{n+1}(t)$ has a single zero (assuming that $F(t)$ can be differentiated $n + 1$ times). Define the point at which this zero occurs as $t = \xi$ to get:

$$F^{n+1}(\xi) = 0 = f^{n+1}(\xi) - C(n + 1)!$$

Rearranging gives $C = \frac{f^{n+1}(\xi)}{(n+1)!}$. Substitute this in the expression above to complete the proof.

$$\max_{x \in [A,B]} |\epsilon_n(x)| = \frac{\max_{x \in [A,B]} |f^{n+1}(x)|}{(n+1)!} \left| \prod_{j=0}^n (x - x_j) \right|$$

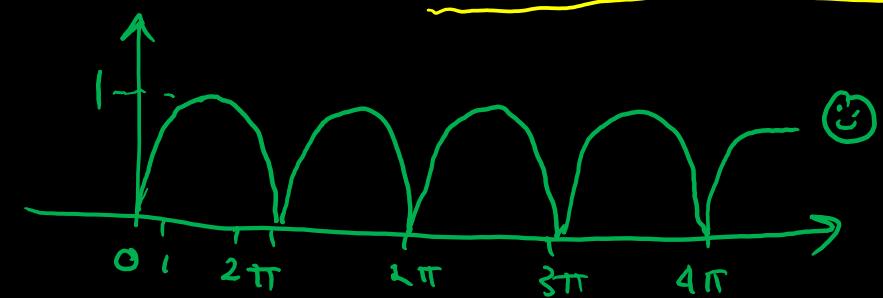
Worked Example 7

Compare the maximum possible errors for the linear, quadratic and cubic expressions derived in WE5 where $f(x) = \sin x$ and interpolating between $1 \leq x \leq 2$. For notational ease, define $\omega_{n+1}(x) = \prod_{j=0}^n (x - x_j)$

linear: $\Rightarrow n = 1$ (highest power of $x = 1$)

$$f'(x) = \cos x \quad \max_{1 \leq x \leq 2} |\sin x| = 1$$

$$f''(x) = -\sin x$$



$$\omega_2(x) = (x - x_0)(x - x_1) = (x - 1)(x - 2) = x^2 - 3x + 2$$

$$\text{max when } \frac{d\omega_2}{dx} = 0 \Rightarrow 2x - 3 = 0 \text{ max when } x = 1.5$$

$$\max_{1 \leq x \leq 2} |\omega_2(x)| = |\omega_2(1.5)| = |(1.5 - 1)(1.5 - 2)| = 0.25$$

$$\max \epsilon = \frac{1}{2!} \times 0.25 = 0.125$$

$$\max_{x \in [A,B]} |\epsilon_n(x)| = \frac{\max_{x \in [A,B]} |f^{n+1}(x)|}{(n+1)!} \left| \prod_{j=0}^n (x - x_j) \right|$$

Worked Example 7

Quadratic $n = 2$:

Therefore need max of the third derivative which is: $f^3(x) = -\cos x$

$$\max_{x \in [1,2]} |f^3(x)| = \max_{x \in [1,2]} |- \cos x| = 0.5403 \quad (\text{which is } |- \cos 1|; \text{ sketch } |- \cos x| \text{ between 1 and 2 to verify})$$

Finding the max of the product term:

$$\omega_3(x) = \left| \prod_{j=0}^2 (x - x_j) \right| = |(x - 1)(x - 1.5)(x - 2)| = |x^3 - 4.5x^2 + 6.5x - 3|$$

$$\frac{d\omega_3}{dx} = 3x^2 - 9x + 6.5$$

Solving for $\frac{d\omega_3}{dx} = 0$:

$$x = 1.5 \pm \frac{\sqrt{3}}{6}$$

$$\max_{x \in [1,2]} |\omega_3(x)| = \left| \omega_3 \left(1.5 - \frac{\sqrt{3}}{6} \right) \right| = \left| \left(0.5 - \frac{\sqrt{3}}{6} \right) \left(-\frac{\sqrt{3}}{6} \right) \left(-0.5 - \frac{\sqrt{3}}{6} \right) \right| = 4.811 \times 10^{-2}$$

$$\max_{x \in [1,2]} |\epsilon_2(x)| = \frac{\max_{x \in [A,B]} |f^3(x)|}{3!} \max_{x \in [1,2]} |\omega_3(x)| = \frac{0.5403}{3!} \times 4.811 \times 10^{-2} = 4.333 \times 10^{-3}$$

$$\max_{x \in [A,B]} |\epsilon_n(x)| = \frac{\max_{x \in [A,B]} |f^{n+1}(x)|}{(n+1)!} \max_{x \in [A,B]} \left| \prod_{j=0}^n (x - x_j) \right|$$

Worked Example 7

Cubic $n = 3$:

Therefore need max of the fourth derivative which is: $f^4(x) = \sin x$

$$\max_{x \in [1,2]} |f^4(x)| = \max_{x \in [1,2]} |\sin x| = 1 \quad (\text{sketch } |\sin x| \text{ between 1 and 2 to verify this})$$

Finding the max of the product term:

$$\omega_4(x) = \left| \prod_{j=0}^3 (x - x_j) \right| = |(x-1)(x-4/3)(x-5/3)(x-2)|$$

Solving for $\frac{d\omega_4}{dx} = 0$: $x = 1.5, 1.5 \pm \frac{\sqrt{5}}{6}$ (you will learn how to find these roots later in the course)

These are the turning points. Not sure at a glance which gives the absolute maximum value so will calculate all three:

$$|\omega_4(1.5)| = |(1/2)(1/6)(-1/6)(-1/2)| = \frac{1}{144} = 6.944 \times 10^{-3}$$

$$\left| \omega_4 \left(1.5 + \frac{\sqrt{5}}{6} \right) \right| = \left| \left(0.5 + \frac{\sqrt{5}}{6} \right) \left(\frac{1+\sqrt{5}}{6} \right) \left(\frac{\sqrt{5}-1}{6} \right) \left(\frac{\sqrt{5}}{6} - 0.5 \right) \right| = 1.234 \times 10^{-2}$$

$$\left| \omega_4 \left(1.5 - \frac{\sqrt{5}}{6} \right) \right| = \left| \left(0.5 - \frac{\sqrt{5}}{6} \right) \left(\frac{1-\sqrt{5}}{6} \right) \left(\frac{-\sqrt{5}-1}{6} \right) \left(\frac{-\sqrt{5}}{6} - 0.5 \right) \right| = 1.234 \times 10^{-2}$$

$$\max_{x \in [1,2]} |\omega_4(x)| = 1.234 \times 10^{-2}$$

$$\max_{x \in [1,2]} |\epsilon_3(x)| = \frac{\max_{x \in [A,B]} |f^4(x)|}{4!} \max_{x \in [1,2]} |\omega_4(x)| = \frac{1}{4!} \times 1.234 \times 10^{-2} = 5.144 \times 10^{-4}$$

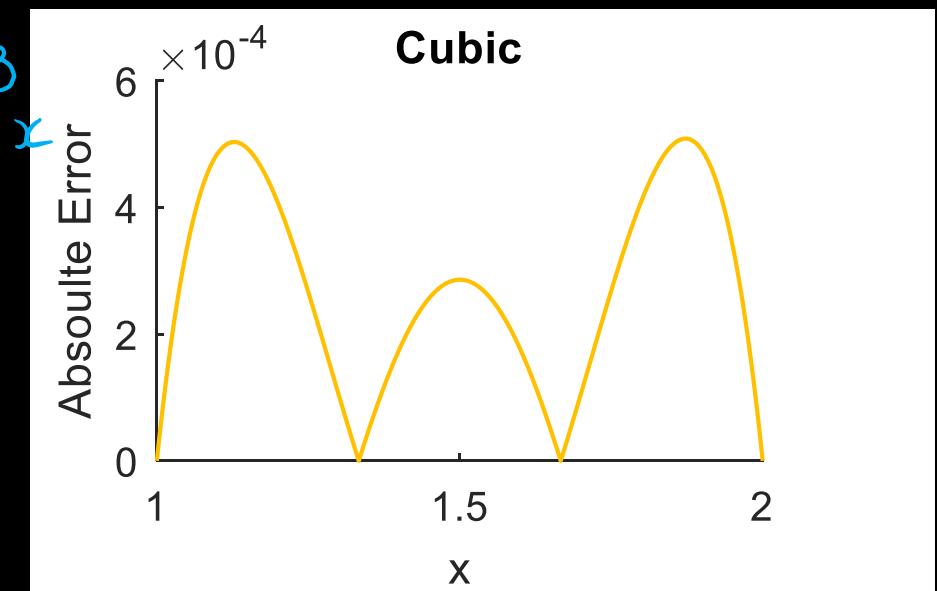
Worked Example 7

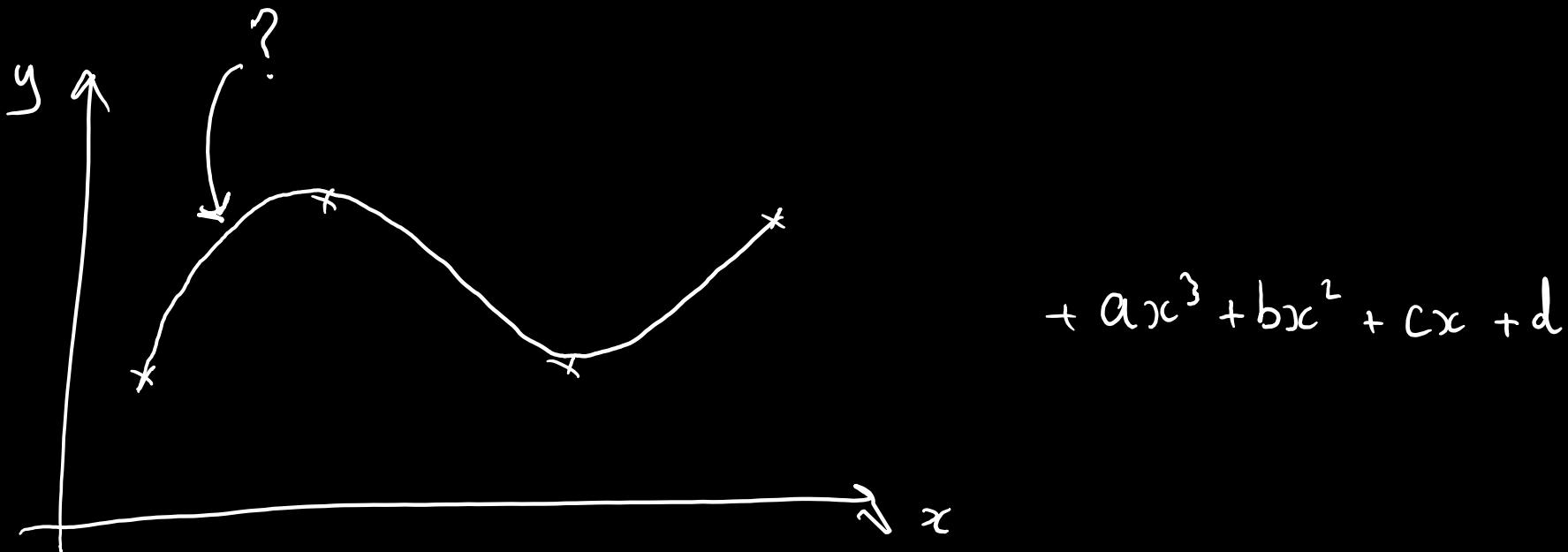
Putting it all together (don't forget the $\frac{1}{(n+1)!}$) and comparing the maximum possible error to the example values we calculated for WE6:

values from previous 3 slides

n	$\max_{x \in [1,2]} \epsilon_n(x) $	$ \epsilon_n\left(\frac{\pi}{2}\right) $	$ \epsilon_n(1.65) $
1	1.25×10^{-1}	1.198×10^{-1}	1.113×10^{-1}
2	4.333×10^{-3}	1.513×10^{-4}	1.861×10^{-4}
3	5.144×10^{-4}	2.299×10^{-4}	4.951×10^{-5}

Sample error knowing sin x





Chapter 2: Numerical Interpolation | Newton & Crude Error Analyses

Error Analyses

If α is the true value, $\tilde{\alpha}$ the numerical estimate, and ϵ the error of the numerical result, then:

$$\alpha = \tilde{\alpha} + \epsilon$$

If two numerical analyses are performed to estimate the same true value, then it must be true that:

$$\tilde{\alpha}_1 + \epsilon_1 = \tilde{\alpha}_2 + \epsilon_2$$

Rearranging:

$$\epsilon_1 - \epsilon_2 = \tilde{\alpha}_2 - \tilde{\alpha}_1$$

If the second estimate is more accurate than the first, then $|\epsilon_1| > |\epsilon_2|$, and so:

$$\epsilon_1 \approx \tilde{\alpha}_2 - \tilde{\alpha}_1$$

provides a crude method to estimate the error of any numerical method.

For Lagrange interpolation, the basic error principle for an n th degree polynomial would be:

$$\epsilon_n(x) = p_{n+1}(x) - p_n(x)$$

error for quadratic = cubic estimate - quadratic estimate

$$p_n(x) = \sum_{j=0}^n y_j \prod_{\substack{k=0 \\ k \neq j}}^n \frac{(x - x_k)}{(x_j - x_k)}$$

Worked Example 6

Linear, quadratic, cubic functions to $\sin(x)$

Estimate the error for WE5 using the basic error principle and compare to the true error value at the points $x = \pi/2$ and $x = 1.65$.

x	n	$p_n(x)$	Estimate ($p_{n+1}(x) - p_n(x)$)	True Error ($f(x) - p_n(x)$)
$\pi/2$	linear 1	0.8802	1.197×10^{-1}	1.198×10^{-1}
	quadratic 2	0.9998	-7.858×10^{-5}	-1.513×10^{-4}
	cubic 3	0.9998	Requires new interpolation	2.299×10^{-4}

Motivation

During WE6 we produced the following table:

x	n	$p_n(x)$	Estimate ($p_{n+1}(x) - p_n(x)$)	True Error ($f(x) - p_n(x)$)
$\pi/2$	1	0.8802	1.197×10^{-1}	1.198×10^{-1}
	2	0.9998	-7.858×10^{-5}	-1.513×10^{-4}
	3	0.9998	Requires new interpolation	2.299×10^{-4}
1.65	1	0.8856	1.111×10^{-1}	1.113×10^{-1}
	2	0.9967	1.365×10^{-4}	1.861×10^{-4}
	3	0.9968	Requires new interpolation	4.951×10^{-5}

Requires new interpolation \Rightarrow “ugh”, “please no, no more, please...”, “why would you do this to me?”, “I take it back, I never should have written I like maths on my personal statement”, etc

We can mitigate the need for this with Newton’s interpolation scheme, but to do so we will need to understand some new notation...

Newton's Divided Difference: Diagram

Node (x_j)	$f(x_j)$	1 st Divided Difference	2 nd Divided Difference	3 rd Divided Difference
x_0	y_0	$y_{0,1} = \frac{y_1 - y_0}{x_1 - x_0}$		
x_1	y_1		$y_{0,1,2} = \frac{y_{1,2} - y_{0,1}}{x_2 - x_0}$	
x_2	y_2	$y_{1,2} = \frac{y_2 - y_1}{x_2 - x_1}$		$y_{0,1,2,3} = \frac{y_{1,2,3} - y_{0,1,2}}{x_3 - x_0}$
x_3	y_3		$y_{1,2,3} = \frac{y_{2,3} - y_{1,2}}{x_3 - x_1}$	
x_4	y_4	$y_{2,3} = \frac{y_3 - y_2}{x_3 - x_2}$	$y_{2,3,4} = \frac{y_{3,4} - y_{2,3}}{x_4 - x_2}$	$y_{1,2,3,4} = \frac{y_{2,3,4} - y_{1,2,3}}{x_4 - x_1}$

Newton's Divided Difference: Notation

$$f[x_0] = f(x_0)$$

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

$$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} = \frac{\frac{f(x_3) - f(x_2)}{x_3 - x_2} - \frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_3 - x_0}$$

and so on...

Newton's Divided Difference: Definition

For the $n + 1$ points x_0, x_1, \dots, x_n of a function $f(x)$:

$$f[x_0] \equiv f(x_0)$$

$$f[x_0, x_1, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0}$$

Note if using notation y instead of $f(x)$ this can also be written as either of the below:

$$[y_0] \equiv y_0$$

$$y_{0,1,\dots,n} = \frac{y_{1,\dots,n} - y_{0,\dots,n-1}}{x_n - x_0}$$

$$[y_0, y_1, \dots, y_n] = \frac{[y_1, \dots, y_n] - [y_0, \dots, y_{n-1}]}{x_n - x_0}$$

Newton's Divided Difference: Interpolation

We again map a polynomial $p_n(x)$ to $f(x)$ such that $p_n(x_0) = f(x_0)$, $p_n(x_1) = f(x_1)$, ..., $p_n(x_n) = f(x_n)$, but this time we base the n th polynomial on the $(n - 1)$ th polynomial:

$$p_n(x) = p_{n-1}(x) + g_n(x) \quad \text{Equation \#1}$$

Thus we need to find $g_n(x)$. If nodes $j = 0, 1, \dots, (n - 1)$ are used to define both $p_n(x_j)$ and $p_{n-1}(x_j)$, then both polynomials are equal to each other at the nodes up to the $(n - 1)$ th node.

Therefore for $j = 0, 1, \dots, (n - 1)$, $g_n(x_j)$ must be zero. This means that $g_n(x)$ must be a polynomial of degree n of the form:

$$g_n(x) = a_n(x - x_0)(x - x_1)(\dots)(x - x_{n-1}) = a_n \prod_{j=0}^{n-1} (x - x_j) \quad \text{Equation \#2}$$

$(x - x_0)(x - x_1)(x - x_2)(\dots)$

The constant a_n can be found by evaluating this at $x = x_n$ and substituting into Equation #1 rearranged (noting that $p_n(x_n) = f(x_n)$):

$$a_n = \frac{f(x_n) - p_{n-1}(x_n)}{(x_n - x_0)(x_n - x_1)(\dots)(x_n - x_{n-1})} = \frac{f(x_n) - p_{n-1}(x_n)}{\prod_{j=0}^{n-1} (x_n - x_j)} \quad \text{Equation \#3}$$

Newton's Divided Difference Interpolation

n	Constants (a_n) Calculated with #3	Interpolation Polynomials ($p_n(x)$) Substitute #3 into #2, and #2 into #1
0	$a_0 = f(x_0) = y_0 = f[x_0]$	$p_0(x) = \underline{y_0} = \text{constant for all } x$
1	$a_1 = \frac{y_1 - y_0}{x_1 - x_0} = f[x_0, x_1]$	$p_1(x) = \underline{y_0} + (x - x_0)\underline{f[x_0, x_1]}$
2	$a_2 = \frac{y_2 - y_0 - (x_2 - x_0)f[x_0, x_1]}{(x_2 - x_1)(x_2 - x_0)} = f[x_0, x_1, x_2]$	$\begin{aligned} p_2(x) &= \underline{y_0} + \underline{(x - x_0)f[x_0, x_1]} + (x - x_1)(x - x_0)f[x_0, x_1, x_2] \\ &= \underline{y_0} + \underline{(x - x_0)f[x_0, x_1]} + \prod_{j=0}^1 (x - x_j) f[x_0, x_1, x_2] \end{aligned}$
...

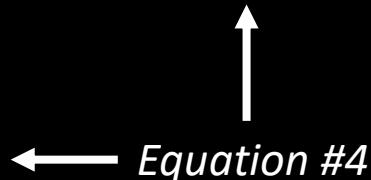
$$p_k(x) = \underline{y_0} + \underline{(x - x_0)f[x_0, x_1]} + \prod_{j=0}^1 (x - x_j) f[x_0, x_1, x_2] + \prod_{j=0}^2 (x - x_j) f[x_0, x_1, x_2, x_3] + \cdots + \prod_{j=0}^{k-1} (x - x_j) f[x_0, \dots, x_k]$$

Newton's Divided Difference Interpolation

$$p_k(x) = y_0 + (x - x_0)f[x_0, x_1] + \prod_{j=0}^1 (x - x_j)f[x_0, x_1, x_2] + \prod_{j=0}^2 (x - x_j)f[x_0, x_1, x_2, x_3] + \cdots + \prod_{j=0}^{k-1} (x - x_j)f[x_0, \dots, x_k]$$

Is the same as:

$$p_k(x) = y_0 + \sum_{i=1}^k \left\{ f[x_0, \dots, x_i] \prod_{j=0}^{i-1} (x - x_j) \right\}$$



Worked Example 8

Simplifying $= p_3(x) = -0.01163x^3 - 0.4350x^2 + 1.454x - 0.1661$

Interpolate $f(x) = \sin(x)$ over the range $1 \leq x \leq 2$ up to $p_3(x)$ and compared to WE5.

step1) find our
divided differences

j	x_j	$f(x_j)$	$f[x_j, x_{j+1}]$	$f[x_j, x_{j+1}, x_{j+2}]$	$f[x_j, \dots, x_{j+3}]$
0	1	0.8415	0.3914	$\frac{0.8415 - 0.9719}{(4/3) - 1}$	
1	4/3	0.9719		$\frac{-0.4815}{0.07041} = \frac{0.07041 - 0.3914}{(5/3) - 1}$	
2	5/3	0.9954	0.07041	-0.4931	
3	2	0.9093		-0.2583	

$$p_k(x) = y_0 + (x - x_0)f[x_0, x_1] + \prod_{j=0}^{k-1}(x - x_j)f[x_0, x_1, x_2, \dots, x_k]$$

linear $= p_1(x) = 0.8415 + 0.3914(x - 1)$

quadratic $= p_2(x) = 0.8415 + 0.3914(x - 1) - 0.4815(x - 1)(x - 4/3)$

cubic $= p_3(x) = 0.8415 + 0.3914(x - 1) - 0.4815(x - 1)(x - 4/3) - 0.01163(x - 1)(x - 4/3)(x - 5/3)$

From WE5

$$\begin{aligned} p_3(x) &= 0.8415 \left(-\frac{9x^3}{2} + \frac{45x^2}{2} - 37x + 20 \right) + 0.9719 \left(\frac{27x^3}{2} - 63x^2 + \frac{189x}{2} - 45 \right) \\ &\quad + 0.9954 \left(-\frac{27x^3}{2} + \frac{117x^2}{2} - 81x + 36 \right) + 0.9093 \left(\frac{9x^3}{2} - 18x^2 + \frac{47x}{2} - 10 \right) \end{aligned}$$

$$p_3(x) = -0.01163x^3 - 0.4350x^2 + 1.454x - 0.1661$$

What do you notice? Why? What does this mean for the errors and our error formula?

Equal Spacing: Forward Divided Difference

$$x_1 = x_0 + h, \quad x_2 = x_0 + 2h, \quad x_3 = x_0 + 3h, \quad \dots, \quad x_n = x_0 + nh$$

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{1! h}$$
$$\Delta y_0 = y_1 - y_0 \quad \Delta y_1 = y_2 - y_1$$

$$f[x_0, x_1, x_2] = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} = \frac{\frac{\Delta y_1 - \Delta y_0}{1! h} - \frac{\Delta y_0}{1! h}}{2h} = \frac{\Delta y_1 - \Delta y_0}{2! h^2} = \frac{\Delta^2 y_0}{2! h^2}$$

$$f[x_0, x_1, x_2, x_3] = \frac{\frac{f(x_3) - f(x_2)}{x_3 - x_2} - \frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_3 - x_0} = \frac{\frac{\Delta^2 y_1 - \Delta^2 y_0}{2! h^2} - \frac{\Delta^2 y_0}{2! h^2}}{3h} = \frac{\Delta^2 y_1 - \Delta^2 y_0}{3! h^3} = \frac{\Delta^3 y_0}{3! h^3}$$

⋮

$$f[x_0, \dots, x_k] = \frac{\Delta^k y_0}{k! h^k}$$

Equation #5

Equal Spacing: Backward Divided Difference

$$x_{-1} = x_0 - h, \quad x_{-2} = x_0 - 2h, \quad x_{-3} = x_0 - 3h, \quad \dots, \quad x_{-n} = x_0 - nh$$

$$f[x_0, x_{-1}] = \frac{f(x_0) - f(x_{-1})}{x_0 - x_{-1}} = \frac{y_0 - y_{-1}}{h} = \frac{\nabla y_0}{1! h}$$



$$f[x_0, x_{-1}, x_{-2}] = \frac{\frac{f(x_0) - f(x_{-1})}{x_0 - x_{-1}} - \frac{f(x_{-1}) - f(x_{-2})}{x_{-1} - x_{-2}}}{x_0 - x_{-2}} = \frac{\frac{\nabla y_0}{1! h} - \frac{\nabla y_{-1}}{1! h}}{2h} = \frac{\nabla y_0 - \nabla y_{-1}}{2! h^2} = \frac{\nabla^2 y_0}{2! h^2}$$

$$f[x_0, x_{-1}, x_{-2}, x_{-3}] = \frac{\frac{f(x_0) - f(x_{-1})}{x_0 - x_{-1}} - \frac{f(x_{-1}) - f(x_{-2})}{x_{-1} - x_{-2}} - \frac{f(x_{-2}) - f(x_{-3})}{x_{-2} - x_{-3}}}{x_0 - x_{-3}} = \frac{\frac{\nabla^2 y_0}{2! h^2} - \frac{\nabla^2 y_{-1}}{2! h^2}}{3h} = \frac{\nabla^2 y_0 - \nabla^2 y_{-1}}{3! h^3} = \frac{\nabla^3 y_0}{3! h^3}$$

$$\vdots$$
$$f[x_0, \dots, x_{-k}] = \frac{\nabla^k y_0}{k! h^k}$$

Equation #6

Forward vs Backward Differences (WE8 Revisited)

The same difference table is formed: for forward difference the top value in each column is used to calculate the polynomial, whereas for backwards difference, the bottom value in the column is used.

$j_{forward}$	$j_{backward}$	x_j	y_j	Δy_j or ∇y_j	$\Delta^2 y_j$ or $\nabla^2 y_j$	$\Delta^3 y_j$ or $\nabla^3 y_j$
0	-3	1	0.8415			
1	-2	4/3	0.9719	0.1305	-	-0.1070
2	-1	5/3	0.9954	0.02347	-0.1096	-0.002584
3	0	2	0.9093	-0.08611		

Forward Divided Difference Interpolation

slide 127

slide 130

$$\Gamma = \frac{x - x_0}{h}$$

We define $x = x_0 + rh$ so that we can develop a very condensed notation:

Equation #7 $\rightarrow x - x_0 = rh, x - x_1 = x - (x_0 + h) = (r - 1)h, x - x_2 = x - (x_0 + 2h) = (r - 2)h, \dots$

Then, by subbing equations #5 and #7 into #4 the forward difference becomes:

$$p_n(x) = y_0 + r\Delta y_0 + \frac{r(r-1)}{2!} \Delta^2 y_0 + \dots + \frac{r(r-1)\dots(r-(n-1))}{n!} \Delta^n y_0$$

$$p_n(x) = \sum_{k=0}^n \binom{r}{k} \Delta^k y_0 \quad (1+x)^k$$

$$\begin{array}{cccccc} & & 1 & & & \\ & & 1 & 1 & 1 & \\ & & 1 & 2 & 1 & \\ & & 1 & 3 & 3 & 1 \\ & & 1 & 4 & 6 & 4 & 1 \end{array}$$

Recalling that $\binom{r}{k}$ is notation a binomial coefficient:

$$\binom{r}{k} = \frac{r!}{k!(r-k)!} = \frac{r(r-1)(r-2)\dots(r-(k-1))(r-k)!}{k!(r-k)!} = \frac{r(r-1)(r-2)\dots(r-(k-1))}{k!}$$

Backward Divided Difference Interpolation

We again define $x = x_0 + rh$ so that we can develop a very condensed notation:

$$x - x_0 = rh, \quad x - x_{-1} = x - (x_0 - h) = (r + 1)h, \quad x - x_{-2} = x - (x_0 - 2h) = (r + 2)h, \dots$$

Then, by subbing equations #6 and #7 into #4, the backward difference becomes:

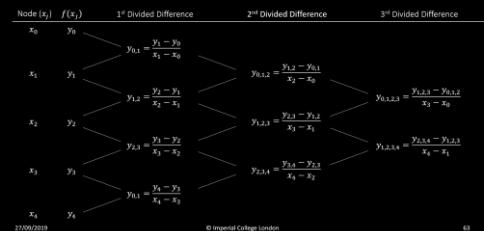
$$p_n(x) = y_0 + r\nabla y_0 + \frac{r(r+1)}{2!} \nabla^2 y_0 + \dots + \frac{r(r+1)\dots(r+(n-1))}{n!} \nabla^n y_0$$

$$p_n(x) = \sum_{k=0}^n (-1)^k \binom{-r}{k} \nabla^k y_0$$

The Beauty of Notation

All of the algebra, algorithms and meaning behind these slides (and those in between):

Newton's Divided Difference: Diagram



27/09/2019 © Imperial College London

Newton's Divided Difference: Notation

$$\begin{aligned} f[x_0] &= f(x_0) \\ f[x_0, x_1] &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \\ f[x_0, x_1, x_2] &= \frac{f(x_1, x_2) - f(x_0, x_1)}{x_2 - x_0} = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} \\ f[x_0, x_1, x_2, x_3] &= \frac{f(x_1, x_2, x_3) - f(x_0, x_1, x_2)}{x_3 - x_0} = \frac{\frac{f(x_3) - f(x_2)}{x_3 - x_2} - \frac{f(x_2) - f(x_1)}{x_2 - x_1} + \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_3 - x_0} \end{aligned}$$

and so on...

27/09/2019 © Imperial College London

Newton's Divided Difference Interpolation

n	Constants ($a_{n,i}$) Calculated with #3	Interpolation Polynomials ($p_n(x)$) Substitute #3 into #2, and #2 into #1
0	$f(x_0) = y_0 = f[x_0]$	$p_0(x) = y_0 = \text{constant for all } x$
1	$\frac{y_1 - y_0}{x_1 - x_0} = f[x_0, x_1]$	$p_1(x) = y_0 + (x - x_0)f[x_0, x_1]$
2	$\frac{y_2 - y_0 - (x_2 - x_0)f[x_0, x_1]}{(x_2 - x_1)(x_2 - x_0)} = f[x_0, x_1, x_2]$	$p_2(x) = y_0 + (x - x_0)f[x_0, x_1] + \frac{1}{2} \prod_{j=0}^{1} (x - x_j)f[x_0, x_1, x_2]$
...
k	$y_k + (x - x_0)f[x_0, x_1] + \prod_{j=0}^{k-1} (x - x_j)f[x_0, x_1, x_2, \dots, x_k]$	$p_k(x) = y_0 + (x - x_0)f[x_0, x_1] + \prod_{j=0}^{k-1} (x - x_j)f[x_0, x_1, x_2, \dots, x_k]$

27/09/2019 © Imperial College London

Forward vs Backward Difference (WE8 Revisited)

The same difference table is formed; for forward difference the top value in each column is used to calculate the polynomial, whereas for backwards difference, the bottom value in the column is used.

<i>Forward</i>	<i>Backward</i>	x_j	$f(x_j)$	$f[x_j, x_{j+1}]$	$f[x_j, x_{j+1}, x_{j+2}]$	$f[x_j, \dots, x_{j+k}]$
0	-3	1	0.8415			
1	-2	4/3	0.9719	0.3914		
2	-1	5/3	0.9954	0.07041	-0.4813	
3	0	2	0.9995	0.2593	-0.4934	-0.01163

27/09/2019 © Imperial College London

is described by just two formulae:

$$p_n(x) = \sum_{k=0}^n \binom{r}{k} \Delta^k y_0, \quad p_n(x) = \sum_{k=0}^n (-1)^k \binom{-r}{k} \nabla^k y_0$$

- These formulae only require addition, multiplication and looping – perfect for stupid computers
- I personally find, when I don't think I understand some notation, it helps to look at the problem from a computer's point of view. I expand the notation by writing out all the terms step-by-step. This takes time, is repetitive, and often requires writing down a lot of algebra, but there is no substitute for doing so, it really helps visualise that the notation is just adding, multiplying and repeating.

"For numerical methods: the skill is in being able to read notation and follow what it means and implies."

Lagrange vs Newton

- For the same order interpolation polynomial, there are no accuracy differences between the two methods
- Lagrange interpolation is advantageous when the same x_j nodes are used for multiple functions
- Newton interpolation is advantageous when the aim is to increase the interpolations polynomial's order until the desired accuracy is achieved (assuming sufficient nodes available)

Chapter 2: Numerical Interpolation | Splines & 2D Matrix Methods

Worked Example 9: Runge's Phenomenon

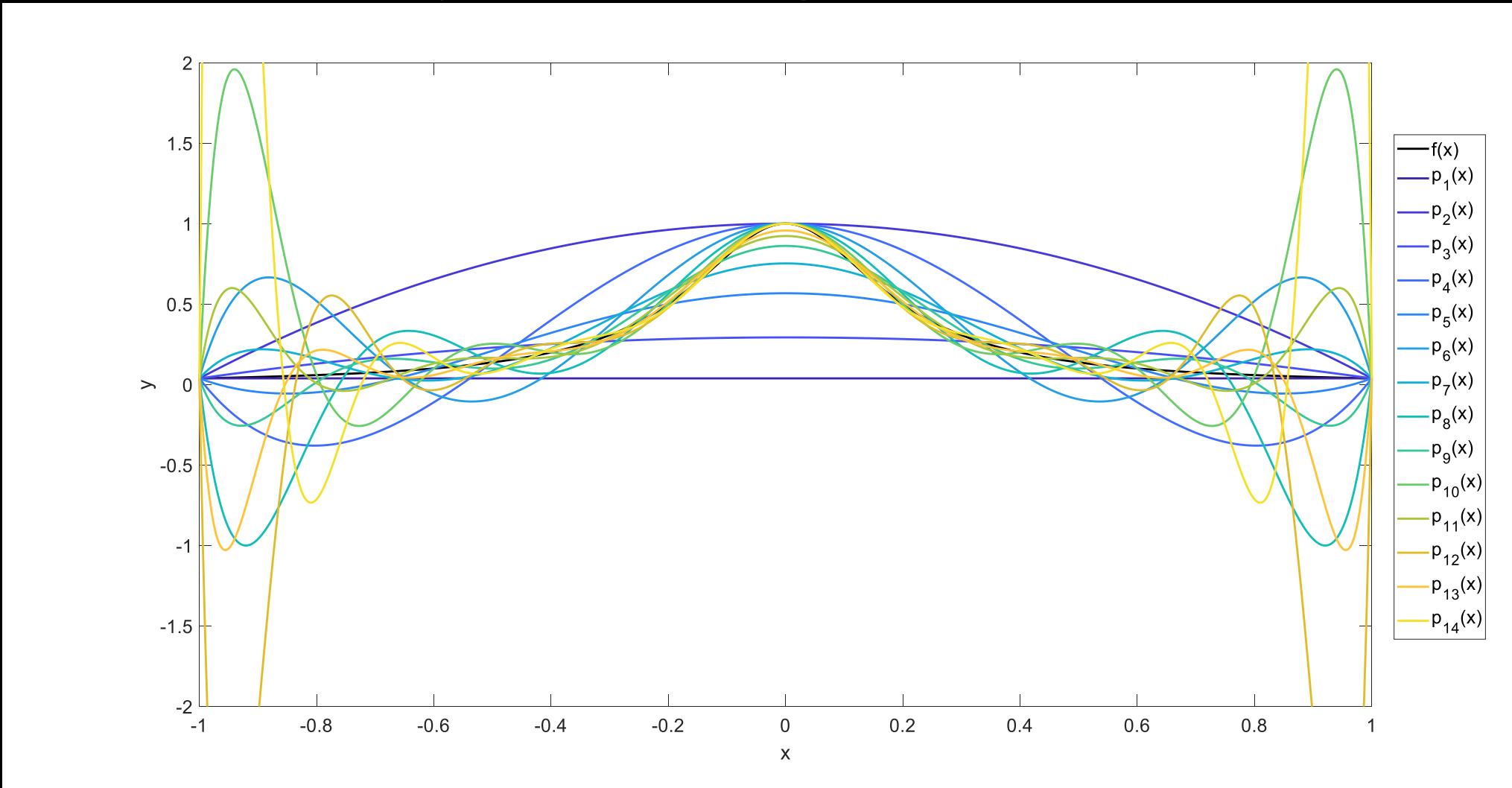
Let's consider interpolation of:

$$f(x) = \frac{1}{(1 + 25x^2)}$$

with increasing degree polynomials.

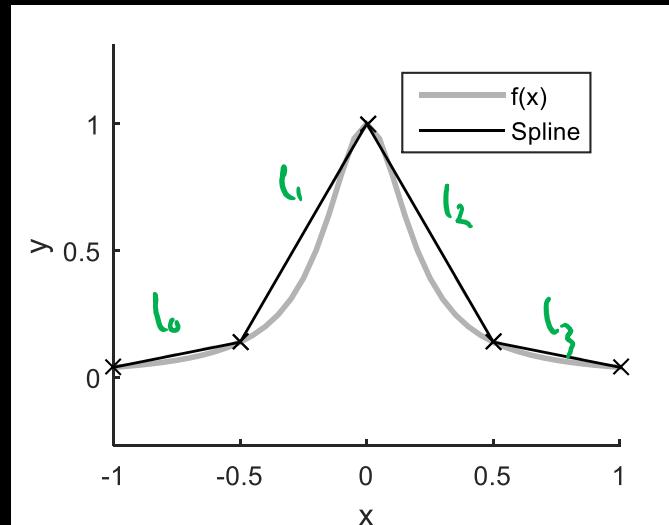
I've coded this in MatLab using the Newton's Divided Difference Polynomial Interpolation script from last time.

Worked Example 9: Runge's Phenomenon



The solution: Splines

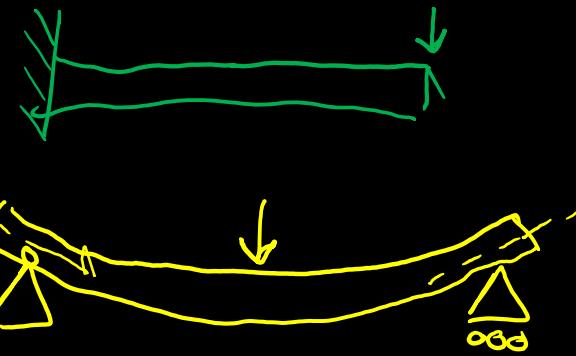
- Practically important – splines are used frequently in CAD software.
- Spline interpolation is *piecewise polynomial interpolation*
 - The interval is divided into multiple, separate polynomial interpolations.
- Piecewise linear interpolation is ugly (below) – the sharp corners would not be suitable for practical applications such as the body of a car or aircraft.



$$S(x) = \begin{cases} l_0(x) & -1 \leq x < -0.5 \\ l_1(x) & -0.5 \leq x < 0 \\ l_2(x) & 0 \leq x < 0.5 \\ l_3(x) & 0.5 \leq x \leq 1 \end{cases}$$

- Cubic splines are much more common and practically useful.

Cubic Splines: Definition



- Call the cubic spline function $S(x)$. For nodes $j = 0, 1, \dots, n$, with known x_j, y_j :
$$S(x_0) = f(x_0) = y_0, \quad S(x_1) = f(x_1) = y_1, \quad \dots, \quad S(x_n) = f(x_n) = y_n$$
- Furthermore, between the nodes, our cubic spline function consists of a series of polynomials with degree ≤ 3 :
$$S(x) = q_0(x) \text{ for } x_0 \leq x \leq x_1, \quad S(x) = q_1(x) \text{ for } x_1 \leq x \leq x_2, \dots, \quad S(x) = q_{n-1}(x) \text{ between } x_{n-1} \leq x \leq x_n$$
- To ensure a continuous smooth function, we define that the spline has continuous first and second derivatives:

- match gradient @ nodes $q'_j(x_j) = q'_{j-1}(x_j)$ and $q''_j(x_j) = q''_{j-1}(x_j)$ for $j = 1, 2, \dots, (n - 1)$ match curvature @ nodes*
- For notational ease, define the nodal derivatives: $v_j = q'_j(x_j) = q'_{j-1}(x_j)$, $v_{j+1} = q'_{j+1}(x_{j+1}) = q'_j(x_{j+1})$, etc.
 - Finally, for a uniquely determined spline, we need boundary conditions for the endpoints. Commonly either:
 - Clamped with a fixed gradient: $S'(x_0) = f'(x_0) = y'_0 = v_0$ and $S'(x_n) = f'(x_n) = y'_n = v_n$
 - Natural, also known as free, with zero curvature: $S''(x_0) = 0$ and $S''(x_n) = 0$. This condition is common for CAD shapes with free ends.

Cubic Splines: Calculating the Cubic Polynomials

The cubic polynomials have the standard form:

$$q_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

The procedure for identifying the polynomials is well established. First, the derivatives for contained nodes $j = 1, 2, \dots, n - 1$ need to be found. The following formula is used to find a linear system of equations which can be solved to find $y'_1, y'_2, \dots, y'_{n-1}$:

$$\frac{v_{j-1}}{x_j - x_{j-1}} + 2\left(\frac{1}{x_j - x_{j-1}} + \frac{1}{x_{j+1} - x_j}\right)v_j + \frac{v_{j+1}}{x_{j+1} - x_j} = 3\left(\frac{\nabla y_j}{(x_j - x_{j-1})^2} + \frac{\nabla y_{j+1}}{(x_{j+1} - x_j)^2}\right)$$

$\nabla = \text{knowns}$ $\nabla = \text{unknowns}$

Where ∇y_j is the notation for a backwards difference (as before) $\nabla y_j = y_j - y_{j-1}$.

Cubic Splines: Calculating the Cubic Polynomials

Then the coefficients can be found:

$$a_j = y_j$$

$$b_j = v_j$$

$$c_j = \frac{3\nabla y_{j+1}}{(x_{j+1} - x_j)^2} - \frac{v_{j+1} + 2v_j}{x_{j+1} - x_j}$$

$$d_j = \frac{-2\nabla y_{j+1}}{(x_{j+1} - x_j)^3} + \frac{v_{j+1} - v_j}{(x_{j+1} - x_j)^2}$$

You will derive these in your tutorials

Worked Example 10: Cubic Spline Interpolation of Runge's Phenomenon

cubic

Calculate the spline function $S(x)$ with 5 nodes for $y = f(x) = \frac{1}{(1+25x^2)}$ for $-1 \leq x \leq 1$ using clamped boundary conditions. $x = [-1, -0.5, 0, 0.5, 1]$; $y = [0.0385, 0.138, 1, 0.138, 0.0385]$
Clamped; choose to match $f'(x)$ at the boundaries

$$f'(x) = -\frac{50x}{(1+25x^2)^2} \quad v_0 = f'(x_0) = 0.0740 \quad ; \quad v_4 = f'(x_4) = -0.0740$$

We will apply our formula:

$$\frac{v_{j-1}}{x_j - x_{j-1}} + 2\left(\frac{1}{x_j - x_{j-1}} + \frac{1}{x_{j+1} - x_j}\right)v_j + \frac{v_{j+1}}{x_{j+1} - x_j} = 3\left(\frac{\nabla y_j}{(x_j - x_{j-1})^2} + \frac{\nabla y_{j+1}}{(x_{j+1} - x_j)^2}\right)$$

equal spacing $\Rightarrow x_j - x_{j-1} = x_{j+1} - x_j = h$

$$\frac{v_{j-1}}{h} + 2\left(\frac{1}{h} + \frac{1}{h}\right)v_j + \frac{v_{j+1}}{h} = 3\left(\frac{y_j - y_{j-1}}{h^2} + \frac{y_{j+1} - y_j}{h^2}\right)$$

Worked Example 10: Cubic Spline Interpolation of Runge's Phenomenon

$$v_{j-1} + 4v_j + v_{j+1} = \frac{3}{h} (y_{j+1} - y_{j-1})$$

$j=0$; \Rightarrow Boundary condition $\Rightarrow v_0 = 0.0740$

$$j=1; v_0 + 4v_1 + v_2 = \frac{3}{h} (y_2 - y_0)$$

$$j=2; v_1 + 4v_2 + v_3 = \frac{3}{h} (y_3 - y_1)$$

$$j=3; v_2 + 4v_3 + v_4 = \frac{3}{h} (y_4 - y_2)$$

$$j=4; \Rightarrow B.C. \Rightarrow v_4 = -0.0740$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} 0.0740 \\ \frac{3}{h} (y_2 - y_0) \\ \frac{3}{h} (y_3 - y_1) \\ \frac{3}{h} (y_4 - y_2) \\ -0.0740 \end{pmatrix}$$

A

$\overline{A} = \underline{v} = \underline{d}$

↑ know ↑ unknown ↑ know

$$\underline{v} = \underline{A}^{-1} \underline{d}$$



You have to let it all go, Neo. Fear, doubt and disbelief. Free your mind.

Worked Example 10: Cubic Spline Interpolation of Runge's Phenomenon

Substitute in values:

$$\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0.0740 \\ 3(1 - 0.0385)/0.5 \\ 3(0.138 - 0.138)/0.5 \\ 3(0.0385 - 1)/0.5 \\ -0.0740 \end{bmatrix}$$

Solving: $v = [0.0740, 1.42, 0, -1.42, -0.0740]$

Worked Example 10: Cubic Spline Interpolation of Runge's Phenomenon

Find the spline polynomials:

$$a_0 = y_0 = 0.0385$$

$$b_0 = v_0 = 0.0740$$

green = known
blue = just calculated

$$c_0 = \frac{3\nabla y_1}{(x_1 - x_0)^2} - \frac{v_1 + 2v_0}{x_1 - x_0} = -1.95$$

$$d_0 = \frac{-2\nabla y_1}{(x_1 - x_0)^3} + \frac{v_1 + v_0}{(x_1 - x_0)^2} = 4.40$$

Note $x_0 = -1$

$$\begin{aligned} q_0(x) &= a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3 = \\ &= 0.0385 + 0.0740(x + 1) - 1.95(x + 1)^2 + 4.40(x + 1)^3 \end{aligned}$$

Re-find constants for $q_1(x)$, $q_2(x)$ and $q_3(x)$

$$-1 \leq x \leq -0.5$$

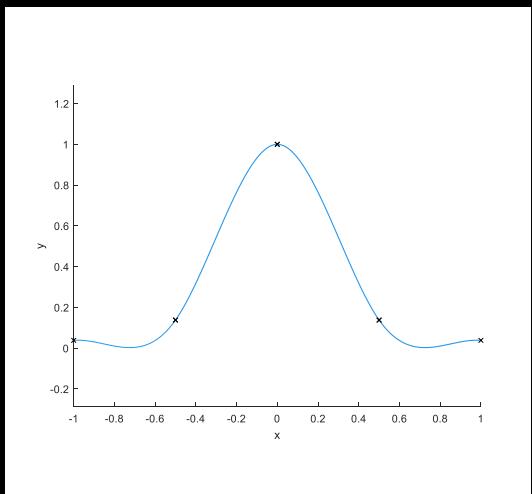
Worked Example 10: Cubic Spline Interpolation of Runge's Phenomenon

$$S(x) = \begin{cases} 0.0385 + 0.0740(x + 1) - 1.95(x + 1)^2 + 4.40(x + 1)^3 & \text{between } -1 \leq x \leq -0.5 \\ 0.138 + 1.42(x + 0.5) + 4.65(x + 0.5)^2 - 8.10(x + 0.5)^3 & \text{between } -0.5 \leq x \leq 0 \\ 1 - 7.50x^2 + 8.10x^3 & \text{between } 0 \leq x \leq 0.5 \\ 0.138 - 1.42(x - 0.5) + 4.65(x - 0.5)^2 - 4.40(x - 0.5)^3 & \text{between } 0.5 \leq x \leq 1 \end{cases}$$

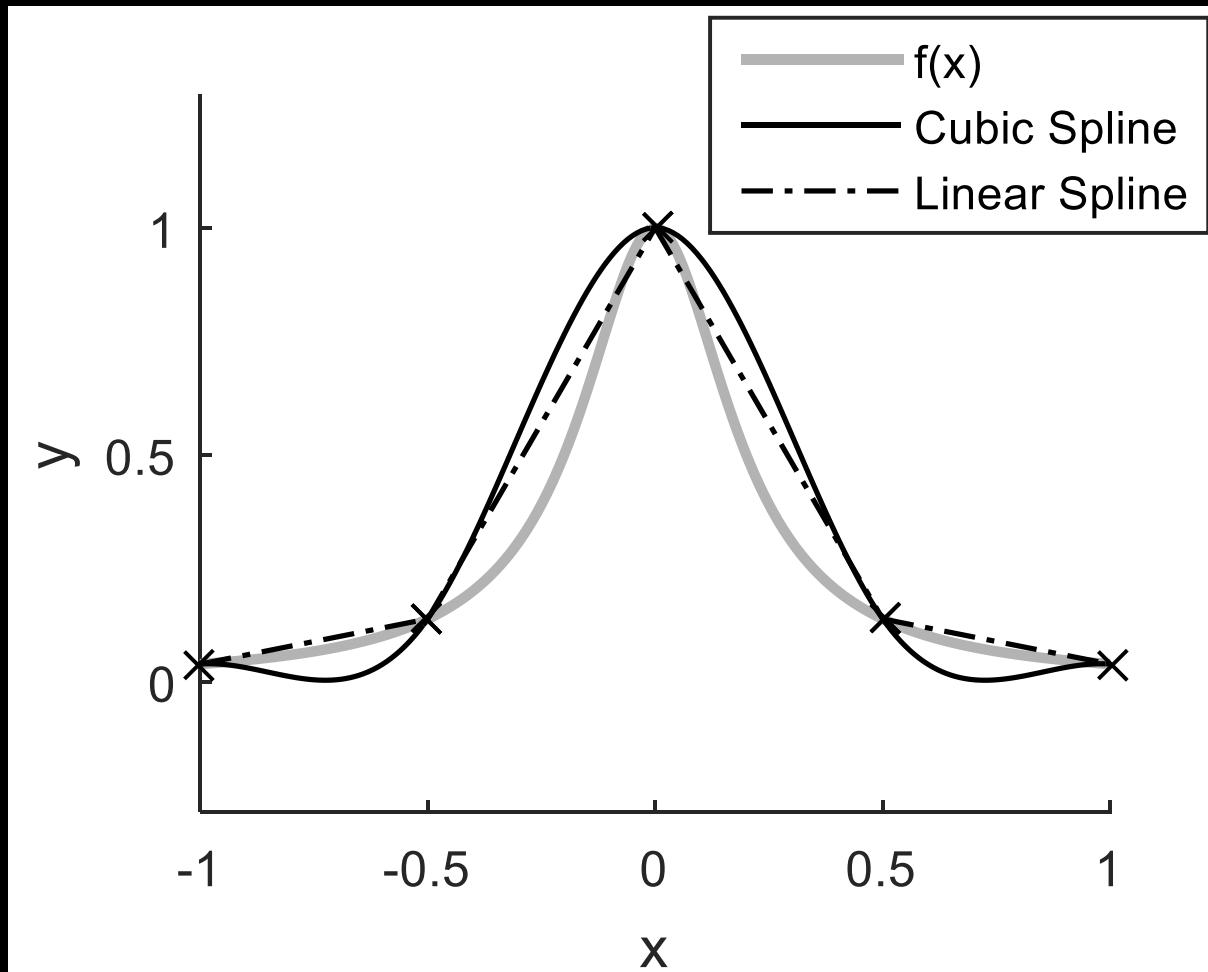
Mental test: what is the minimum number of nodes that are needed to find a cubic spline polynomial?

How does this compare to a Lagrange or Newton polynomial?

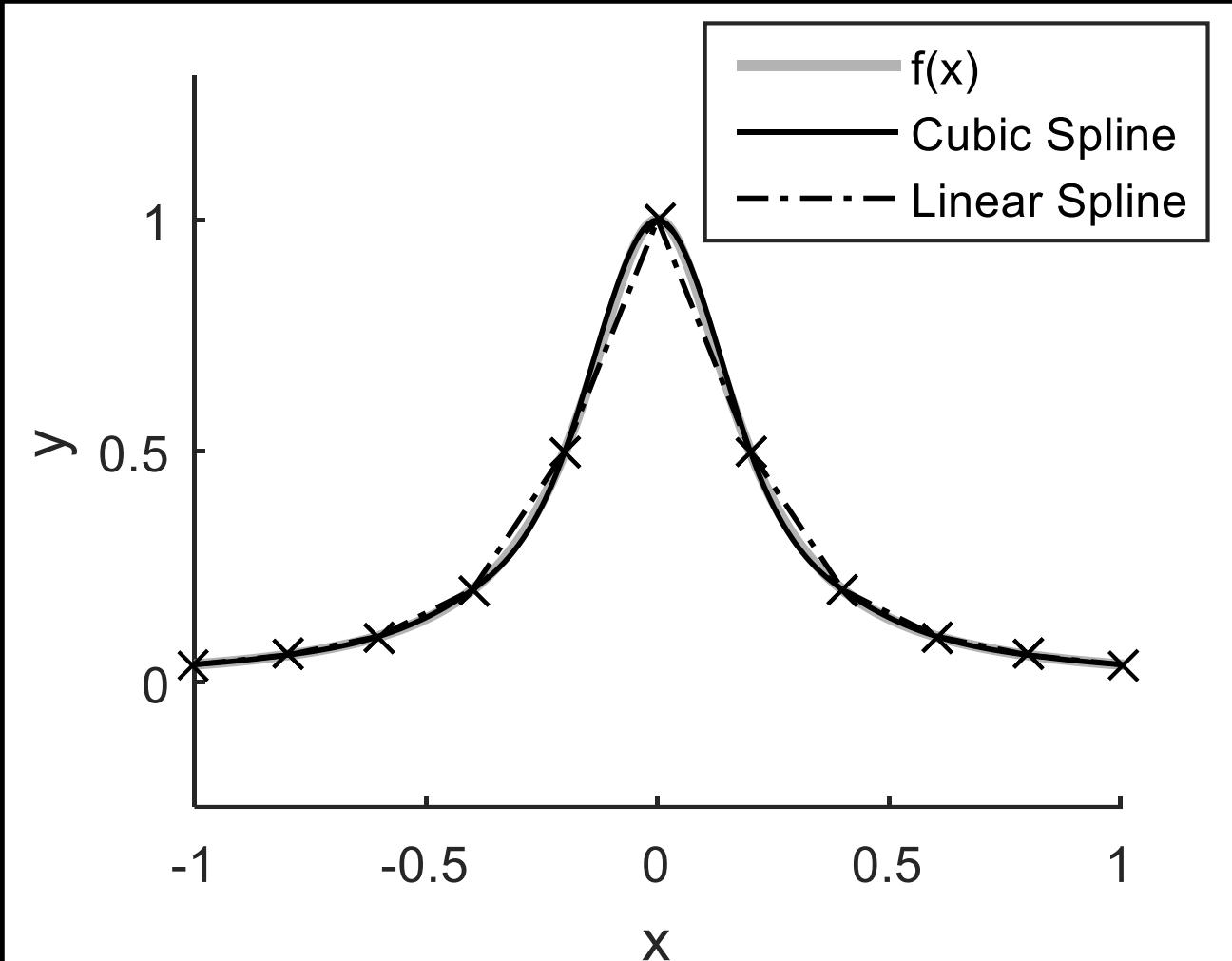
Why?



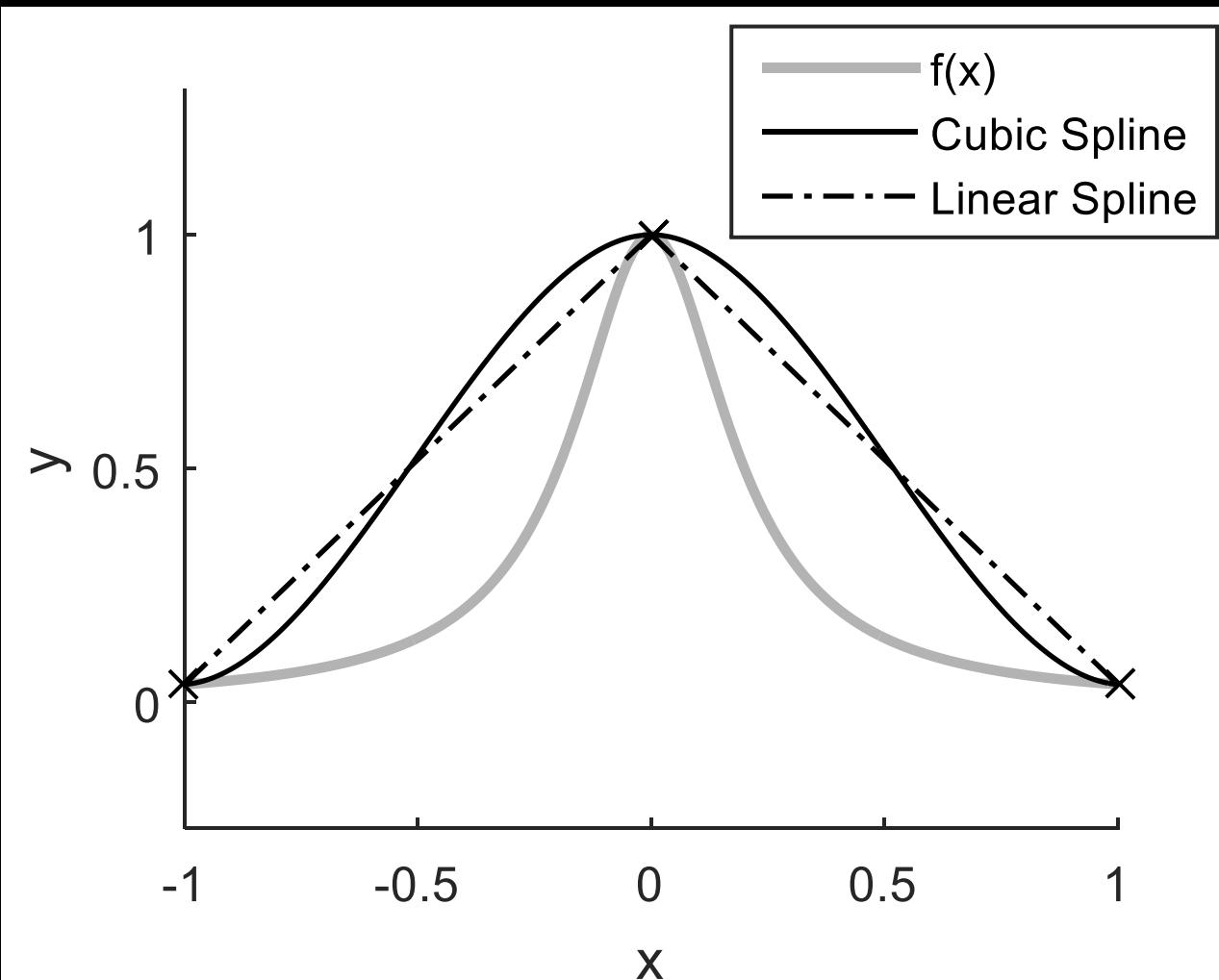
WE10 Plotted (5 nodes)



With 11 nodes

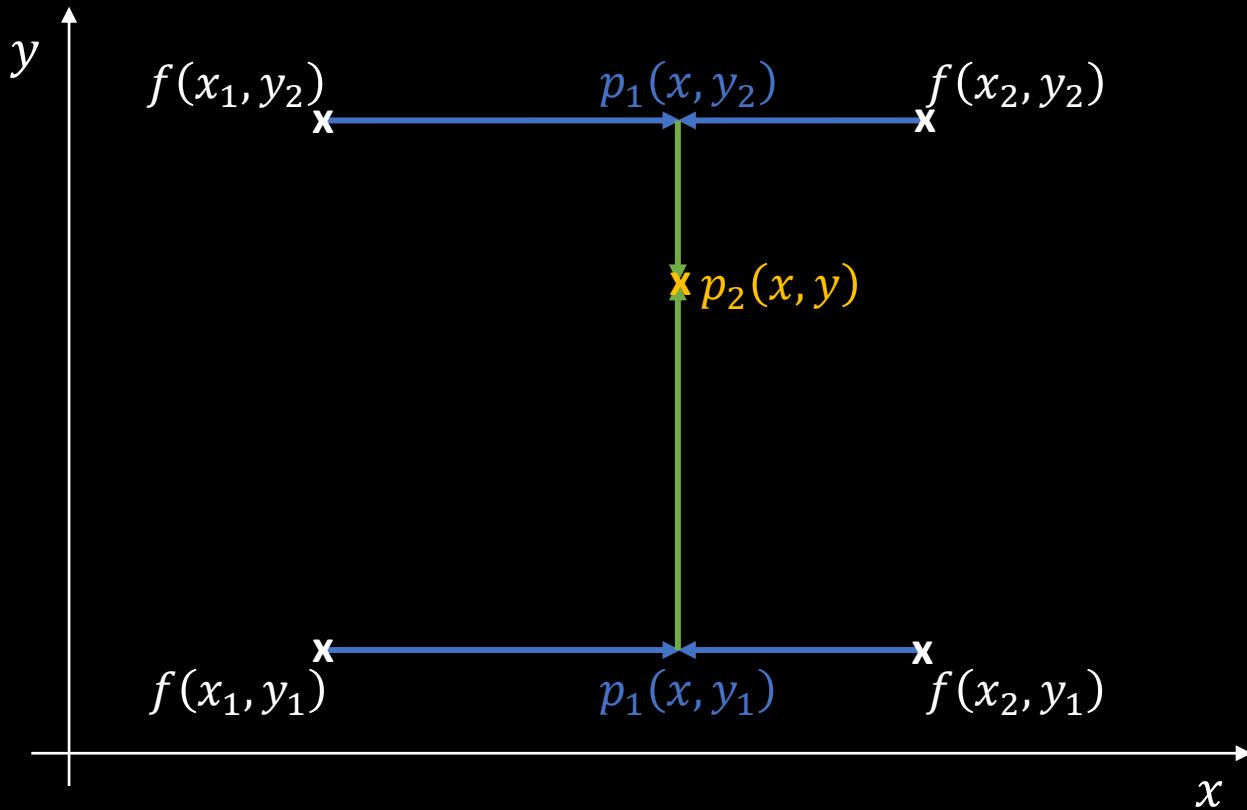


With 3 nodes



Interpolation in 2D

- We want to approximate a function $z = f(x, y)$ with an interpolating polynomial $p_2(x, y)$
- This can be achieved stepwise:



Interpolation in 2D: Bilinear Interpolation

- Using Lagrange interpolation (you could use any linear interpolation scheme)
- First we interpolate in the x direction to get a polynomial as a function of x at both y_1 and y_2 :

$$p_1(x, y_1) = \frac{x - x_2}{x_1 - x_2} f(x_1, y_1) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_1)$$

$$p_1(x, y_2) = \frac{x - x_2}{x_1 - x_2} f(x_1, y_2) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_2)$$

- Then these are interpolated in the y direction to complete the 2D interpolation:

$$\begin{aligned} z(x, y) &\approx p_2(x, y) = \frac{y - y_2}{y_1 - y_2} p_1(x, y_1) + \frac{y - y_1}{y_2 - y_1} p_1(x, y_2) \\ &= \frac{1}{(y_1 - y_2)(x_1 - x_2)} [(x - x_2)(y - y_2)f(x_1, y_1) - (x - x_1)(y - y_2)f(x_2, y_1) - (x - x_2)(y - y_1)f(x_1, y_2) + (x - x_1)(y - y_1)f(x_2, y_2)] \end{aligned}$$

- This works but is very ugly...

Interpolation in 2D: Bilinear Interpolation

More elegantly, the following form can be used:

$$p_2(x, y) = a_0 + a_1x + a_2y + a_3xy$$

Where:

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_1 & x_2y_1 \\ 1 & x_1 & y_2 & x_1y_2 \\ 1 & x_2 & y_2 & x_2y_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} f(x_1, y_1) \\ f(x_2, y_1) \\ f(x_1, y_2) \\ f(x_2, y_2) \end{bmatrix}$$

And so the constants can be found by inverting the matrix:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_1 & x_2y_1 \\ 1 & x_1 & y_2 & x_1y_2 \\ 1 & x_2 & y_2 & x_2y_2 \end{bmatrix}^{-1} \begin{bmatrix} f(x_1, y_1) \\ f(x_2, y_1) \\ f(x_1, y_2) \\ f(x_2, y_2) \end{bmatrix} = \frac{1}{(y_1 - y_2)(x_1 - x_2)} \begin{bmatrix} x_2y_2 & -x_1y_2 & -x_2y_1 & x_1y_1 \\ -y_2 & y_2 & y_1 & -y_1 \\ -x_2 & x_1 & x_2 & -x_1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} f(x_1, y_1) \\ f(x_2, y_1) \\ f(x_1, y_2) \\ f(x_2, y_2) \end{bmatrix}$$

Compare this to the previous slide to convince yourself they are the same

Interpolation in 3D: Trilinear

The pattern follows:

$$p(x, y, z) = \begin{bmatrix} 1 & x_1 & y_1 & z_1 & x_1y_1 & x_1z_1 & y_1z_1 & x_1y_1z_1 \\ 1 & x_2 & y_1 & z_1 & x_2y_1 & x_2z_1 & y_1z_1 & x_2y_1z_1 \\ 1 & x_1 & y_2 & z_1 & x_1y_2 & x_1z_1 & y_2z_1 & x_1y_2z_1 \\ 1 & x_2 & y_2 & z_1 & x_2y_2 & x_2z_2 & y_2z_1 & x_2y_2z_1 \\ 1 & x_1 & y_1 & z_2 & x_1y_1 & x_1z_2 & y_1z_2 & x_1y_1z_2 \\ 1 & x_2 & y_1 & z_2 & x_2y_1 & x_2z_2 & y_1z_2 & x_2y_1z_2 \\ 1 & x_1 & y_2 & z_2 & x_1y_2 & x_1z_2 & y_2z_2 & x_1y_2z_2 \\ 1 & x_2 & y_2 & z_2 & x_2y_2 & x_2z_2 & y_2z_2 & x_2y_2z_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} = \begin{bmatrix} f(x_1, y_1, z_1) \\ f(x_2, y_1, z_1) \\ f(x_1, y_2, z_1) \\ f(x_2, y_2, z_1) \\ f(x_1, y_1, z_2) \\ f(x_2, y_1, z_2) \\ f(x_1, y_2, z_2) \\ f(x_2, y_2, z_2) \end{bmatrix}$$

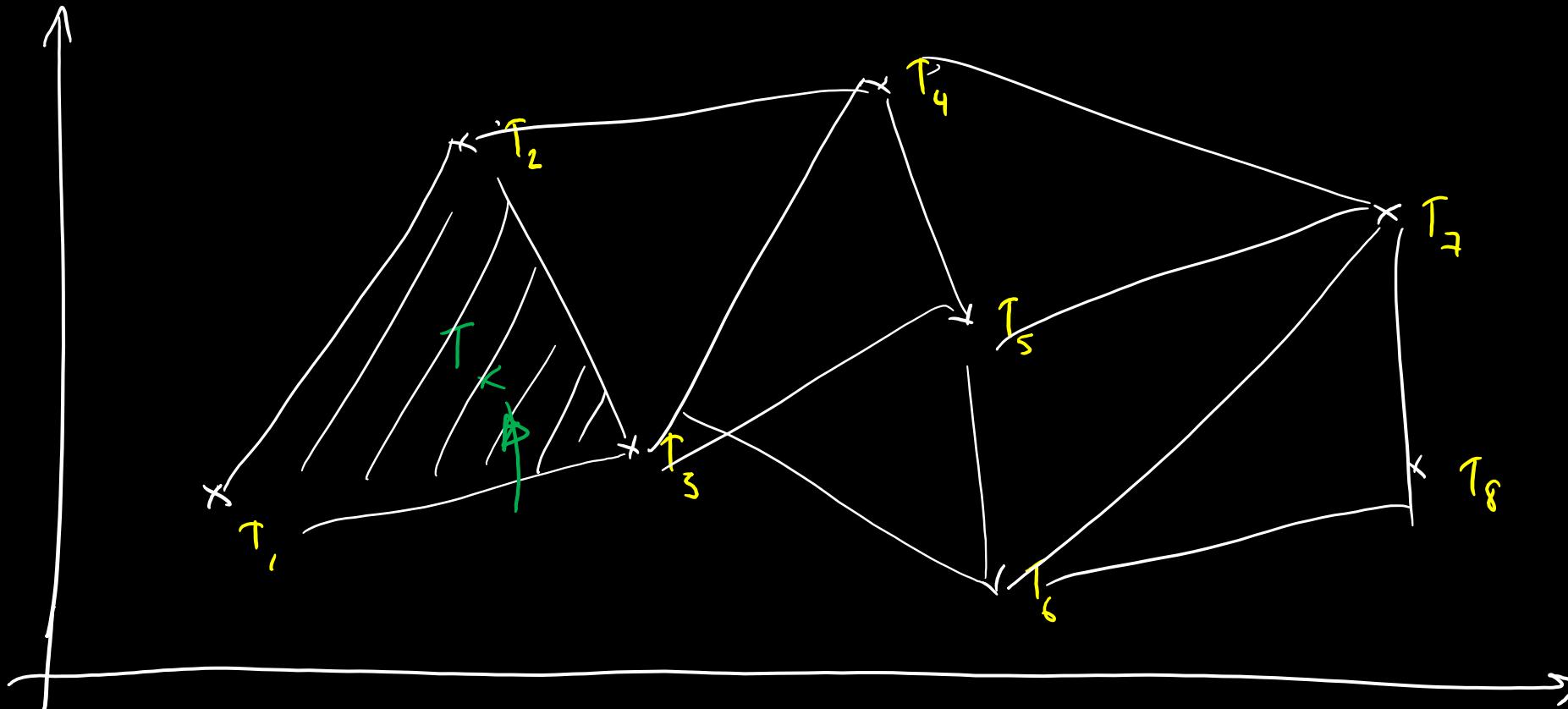
Bilinear and Trilinear interpolation are often used in video games.

Bi and Tricubic Interpolation

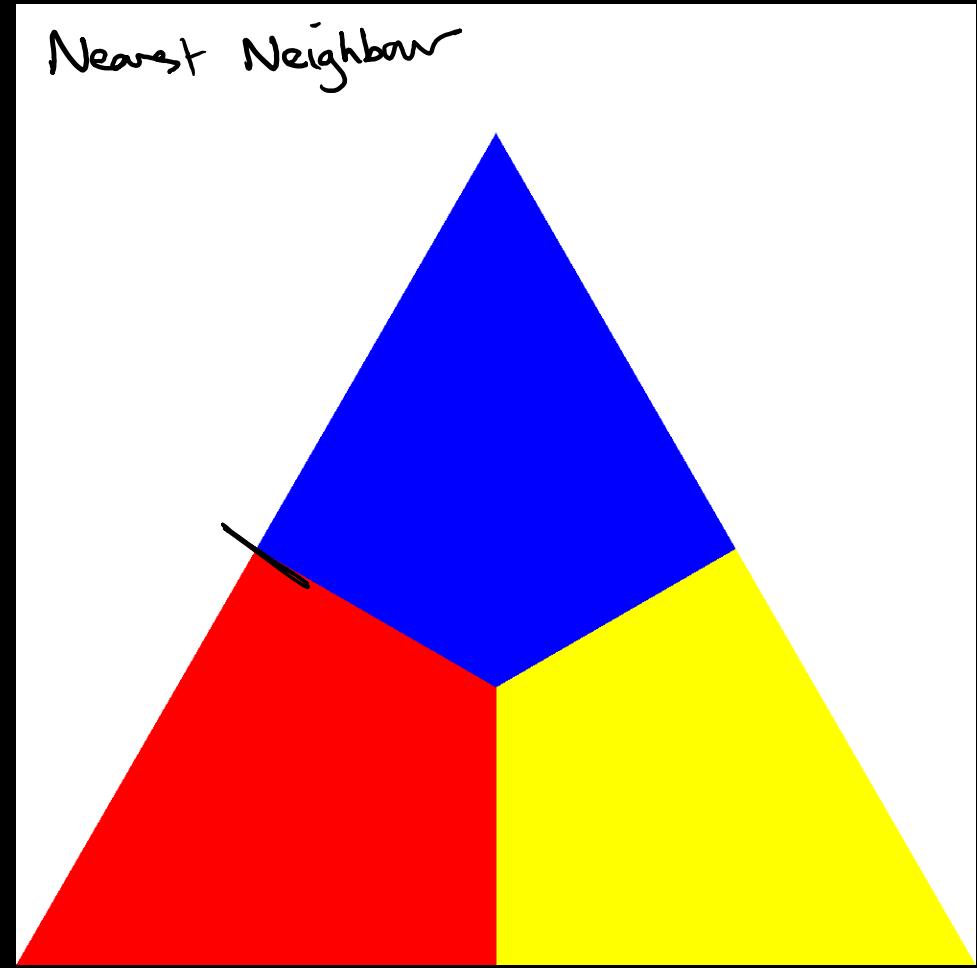
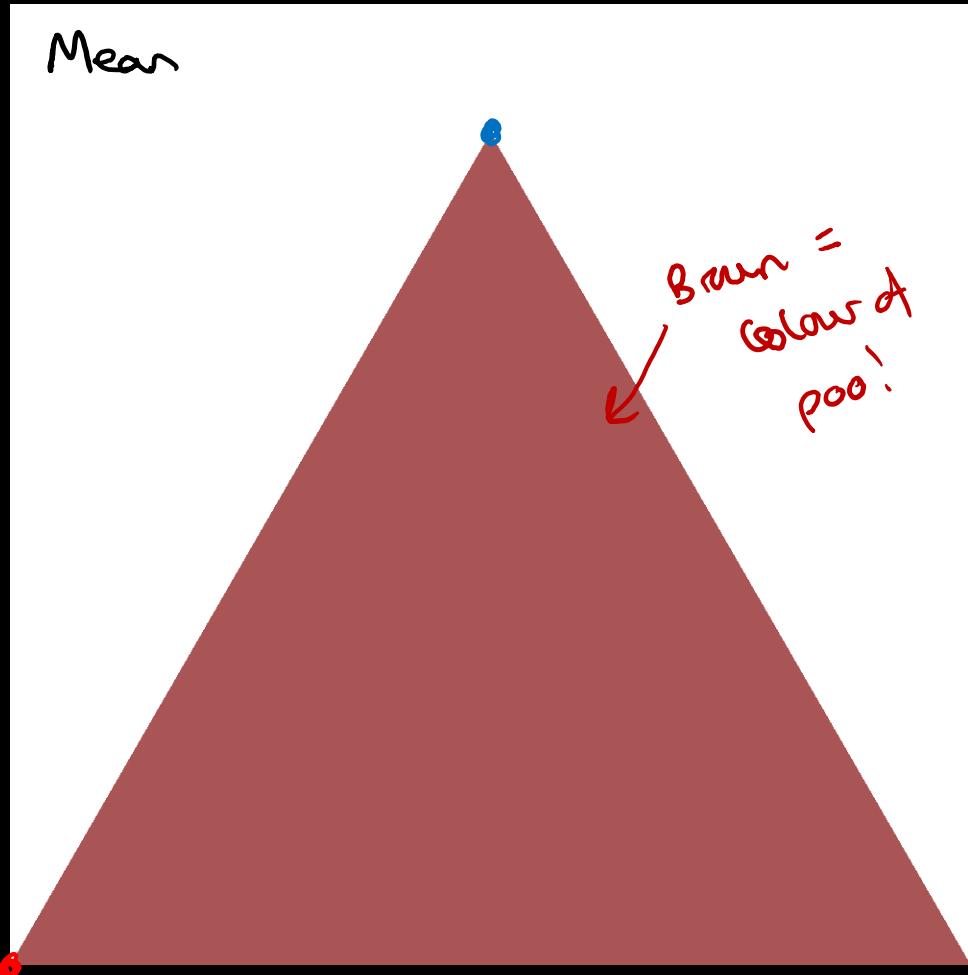
- Bi and tricubic are also possible, either using Lagrange interpolation or splines. Again they can be calculated with sequential interpolation in x , then y , then z , or for computation efficiency, a matrix calculation.
- See *Lekien F and Marsden J, Tricubic interpolation in three dimensions, Int. J. Numer. Meth. Eng.* 2005; 63:455–471 for an example

Chapter 2: Numerical Interpolation | Interpolating Triangulated 2D Domains

2D Interpolation for Unstructured Grids (Fields with Irregularly Spaced Nodes)



Worked Example 11: Interpolating in a Triangle (lazy methods)



Worked Example 11: Interpolating in a Triangle (elegant but flawed)

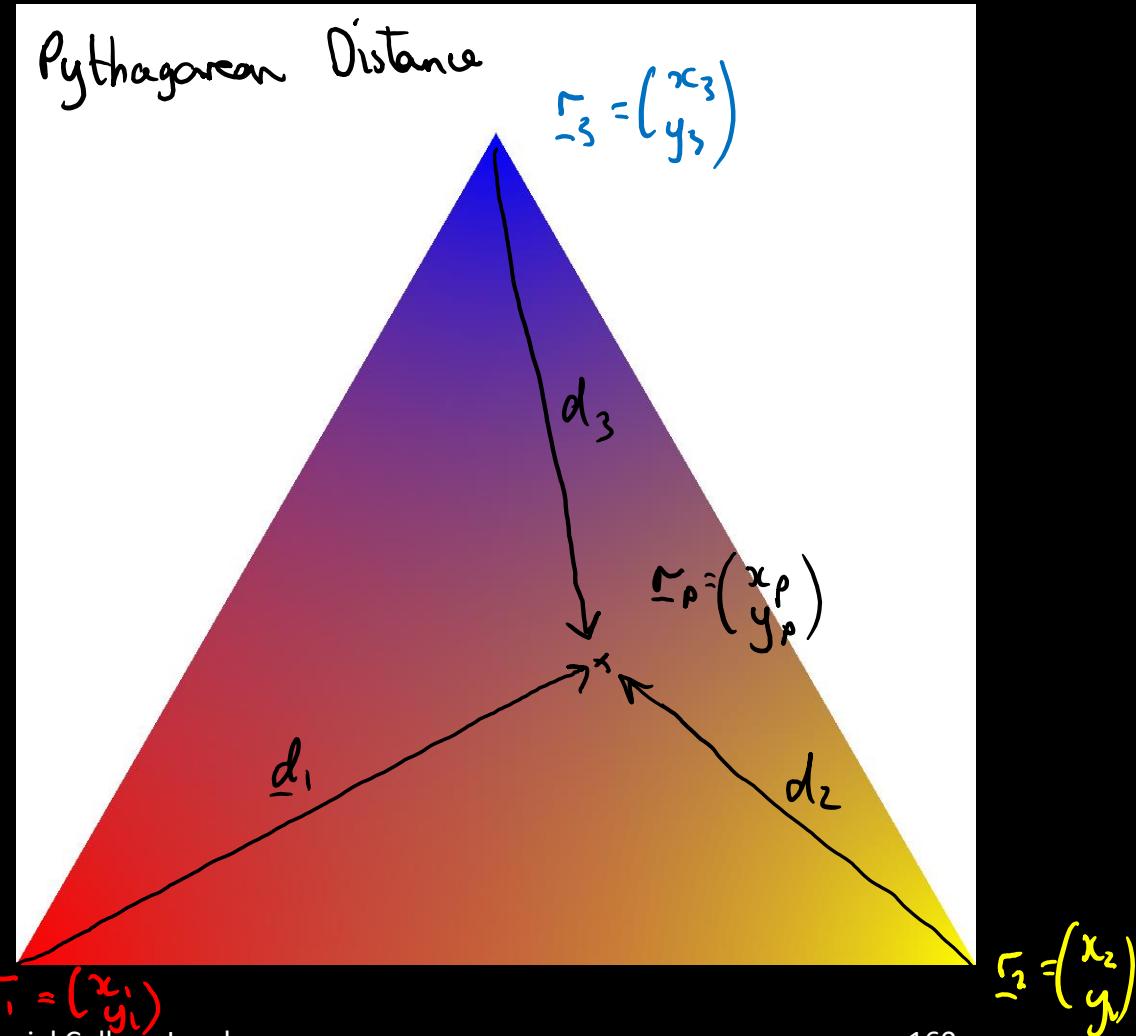
- It is more elegant (but still flawed) to interpolate proportional to inverse distance weightings:

$$d_1 = \sqrt{(x_1 - x_p)^2 + (y_1 - y_p)^2}, \quad w_1 = 1/d_1$$

$$d_2 = \sqrt{(x_2 - x_p)^2 + (y_2 - y_p)^2}, \quad w_2 = 1/d_2$$

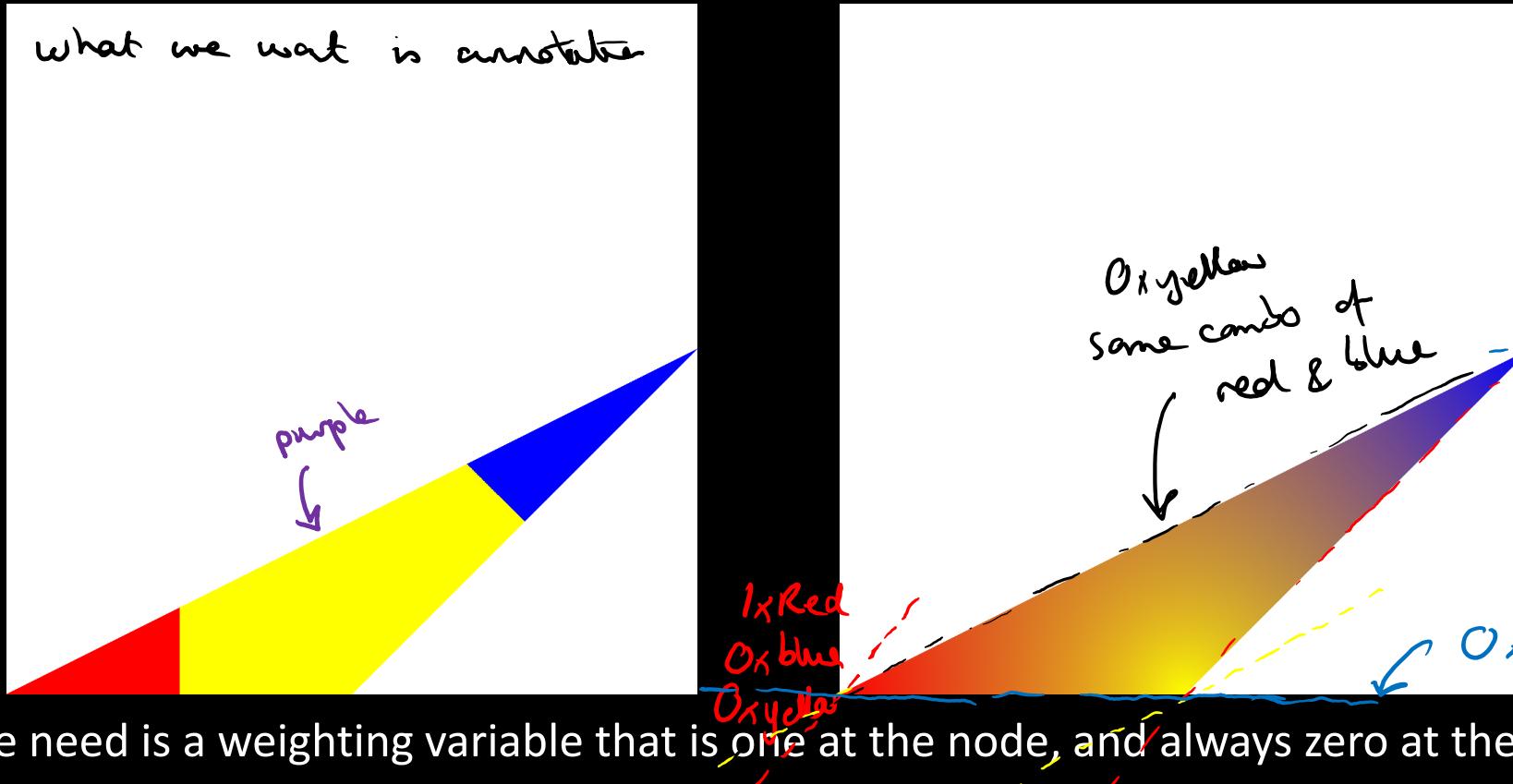
$$d_3 = \sqrt{(x_3 - x_p)^2 + (y_3 - y_p)^2}, \quad w_3 = 1/d_3$$

$$\text{Colour} = \frac{w_1 \text{Red} + w_2 \text{Yellow} + w_3 \text{Blue}}{w_1 + w_2 + w_3}$$



Worked Example 11: Interpolating in a Triangle (the problem)

Nearest neighbour and inverse distance do not work intuitively for obtuse triangles:



What we need is a weighting variable that is one at the node, and always zero at the opposite edge.

The Solution: Barycentric Coordinates

- Consider a triangle with vertices located at $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$. Each point \mathbf{r} inside the triangle can be described as a linear combination of the vertices:

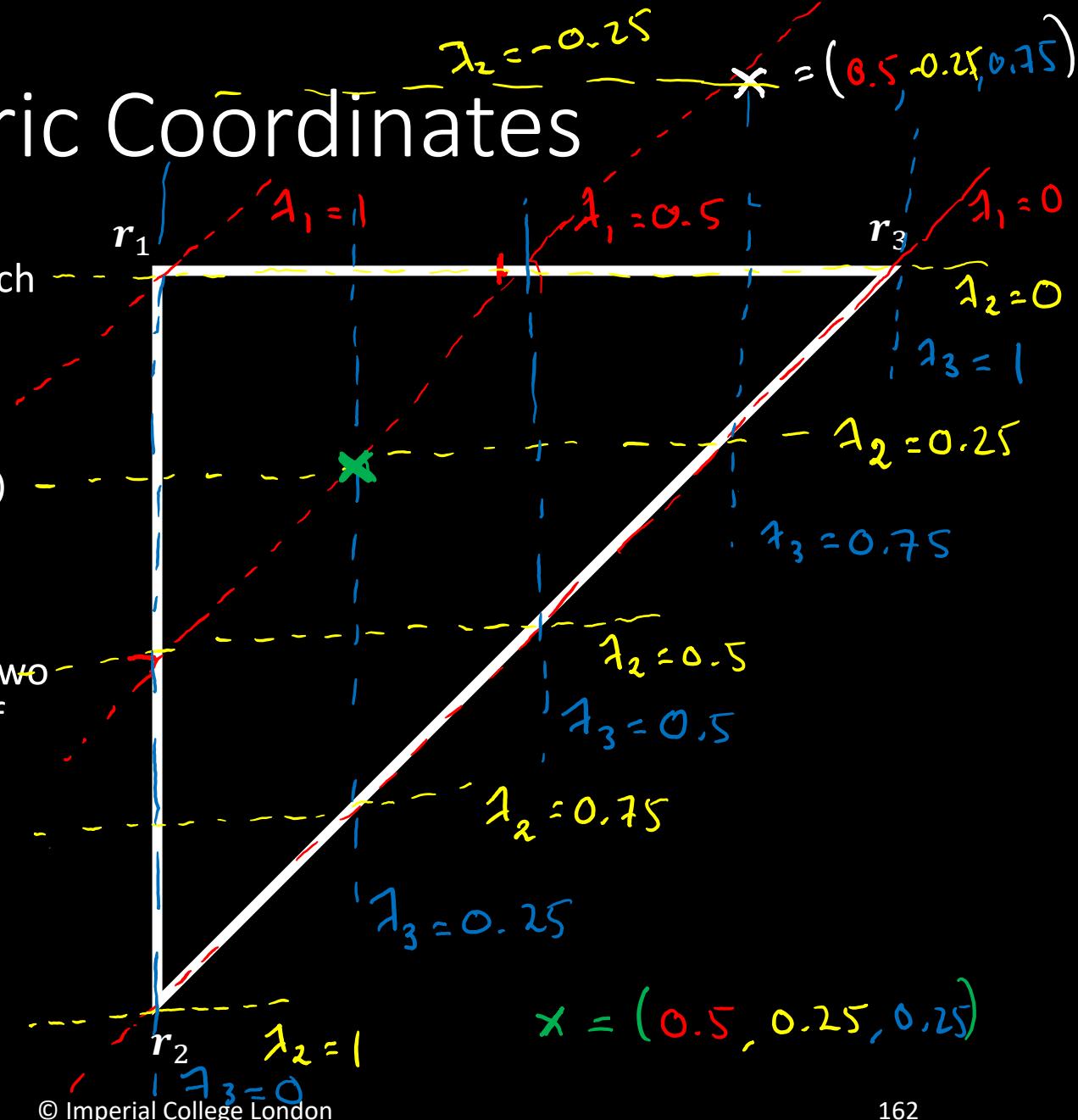
$$\mathbf{r} = \lambda_1 \mathbf{r}_1 + \lambda_2 \mathbf{r}_2 + \lambda_3 \mathbf{r}_3$$

Alternative notation: $\mathbf{r} = (\lambda_1, \lambda_2, \lambda_3)$ or $\mathbf{r} = (\lambda_1 : \lambda_2 : \lambda_3)$

- Each barycentric coordinate (each λ) is 1 at its corresponding vertex, and 0 at the opposite edge.
- However, now we have three coordinates, describing a two-dimensional triangle and hence have a surplus degree of freedom. The coordinates are thus normalised:

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

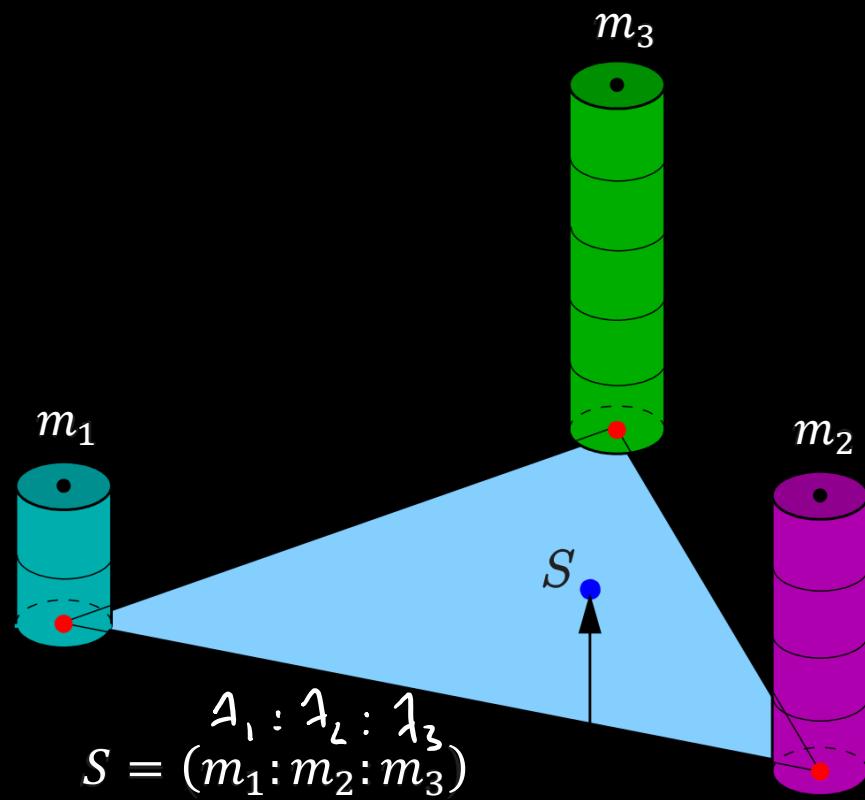
- Barycentric coordinates continue outside the triangle:
 - One/two barycentric coordinates must be negative.
 - Thus, testing if any $\lambda_i < 0$ is a simple method to determine if a point is inside a triangle or not.



For a Triangle: Barycentric = Areal Coordinates

$$= \frac{1}{2} b \times h$$

- Relationship to masses (barycentric):

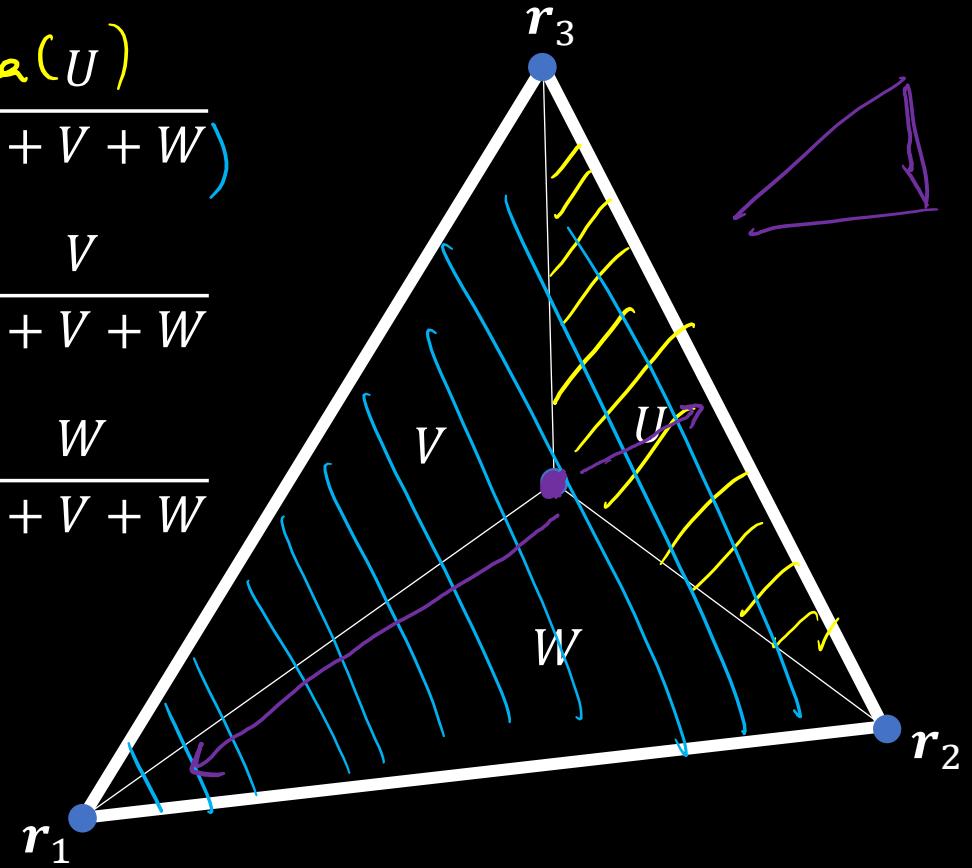


- Relationship to areas (areal):

$$\lambda_1 = \frac{\text{area}(U)}{\text{area}(U + V + W)}$$

$$\lambda_2 = \frac{V}{U + V + W}$$

$$\lambda_3 = \frac{W}{U + V + W}$$



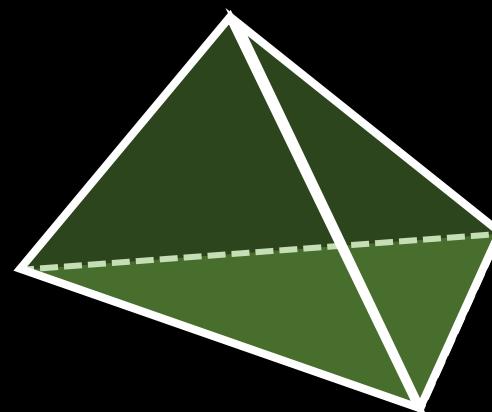
Left image from Ag2gaeh, Own Work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=93193242>

Barycentric Interpolation

- Barycentric coordinates make interpolation inside a triangle simple:

$$f(\mathbf{r}) = \lambda_1 f(\mathbf{r}_1) + \lambda_2 f(\mathbf{r}_2) + \lambda_3 f(\mathbf{r}_3)$$

- Interpolation can only be done inside the triangle, so first use the barycentric coordinates to test if inside the triangle and only interpolate if so.
- To both test whether inside the triangle, and to interpolate, we thus need to find the values for λ_i (next slide).
- The same approach works in 3D as well using tetrahedra rather than triangles and four barycentric coordinates $\lambda_1, \lambda_2, \lambda_3, \lambda_4$.



$$\tilde{z} = \frac{\tilde{w}}{2} = \frac{|\underline{a} \times \underline{b}|}{2}$$

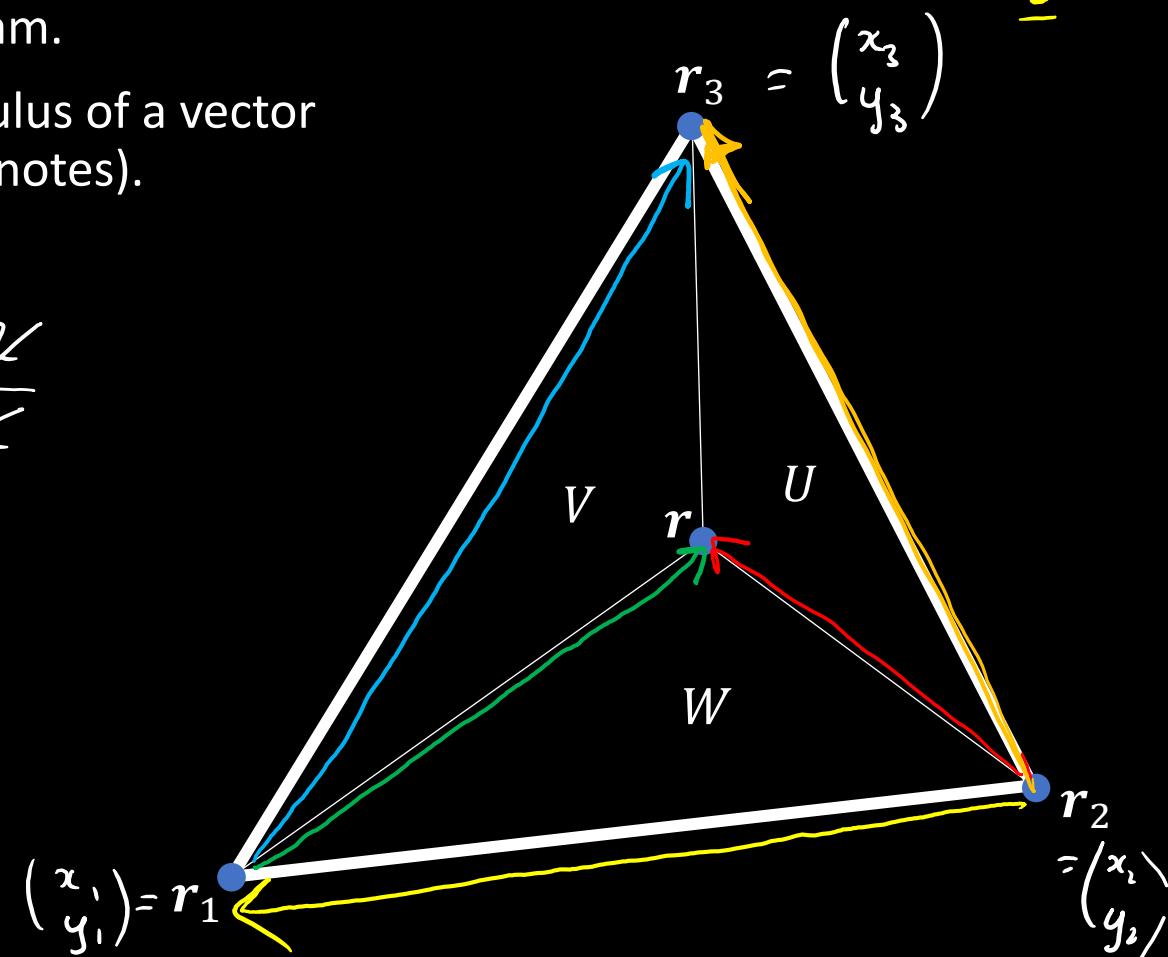
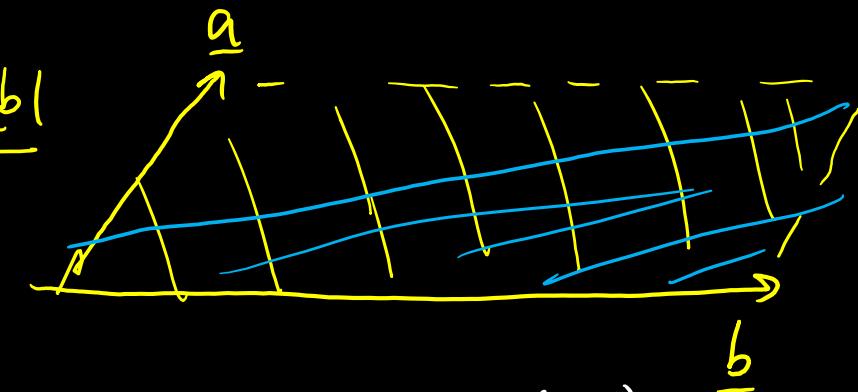
Cartesian to Barycentric – Area Method

- The area of a triangle is half the area of a parallelogram.
- The area of a parallelogram is equivalent to the modulus of a vector cross product $|\underline{a} \times \underline{b}|$ (as described in your first year notes).
- Therefore:

$$\lambda_1 = \frac{\text{area}(U)}{\text{area}(U+V+W)} = \frac{|(\underline{r}_1 - \underline{r}_2) \times (\underline{r}_3 - \underline{r}_2)|}{|(\underline{r}_1 - \underline{r}_2) \times (\underline{r}_3 - \underline{r}_2)|}$$

$$\lambda_2 = \frac{\text{area}(V)}{\text{area}(U+V+W)} = \frac{|(\underline{r}_1 - \underline{r}_1) \times (\underline{r}_3 - \underline{r}_1)|}{|(\underline{r}_1 - \underline{r}_1) \times (\underline{r}_3 - \underline{r}_1)|}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$



Cartesian to Barycentric – Matrix Method

any point $\begin{pmatrix} \underline{x} \\ \underline{y} \end{pmatrix} = \lambda_1 \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \lambda_2 \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + \lambda_3 \begin{pmatrix} x_3 \\ y_3 \end{pmatrix}$
 $\underline{x} = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$
 $\underline{y} = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3$
 $1 = \lambda_1 + \lambda_2 + \lambda_3$

- If the vertices are known in cartesian coordinates, then $\underline{r}_i = (x_i, y_i)$ and hence:

$$\begin{array}{ll} \text{row 1} & \underline{x} = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 \\ \text{row 2} & \underline{y} = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 \\ & 1 = \lambda_1 + \lambda_2 + \lambda_3 \end{array}$$

- These three equations can be written in matrix form:

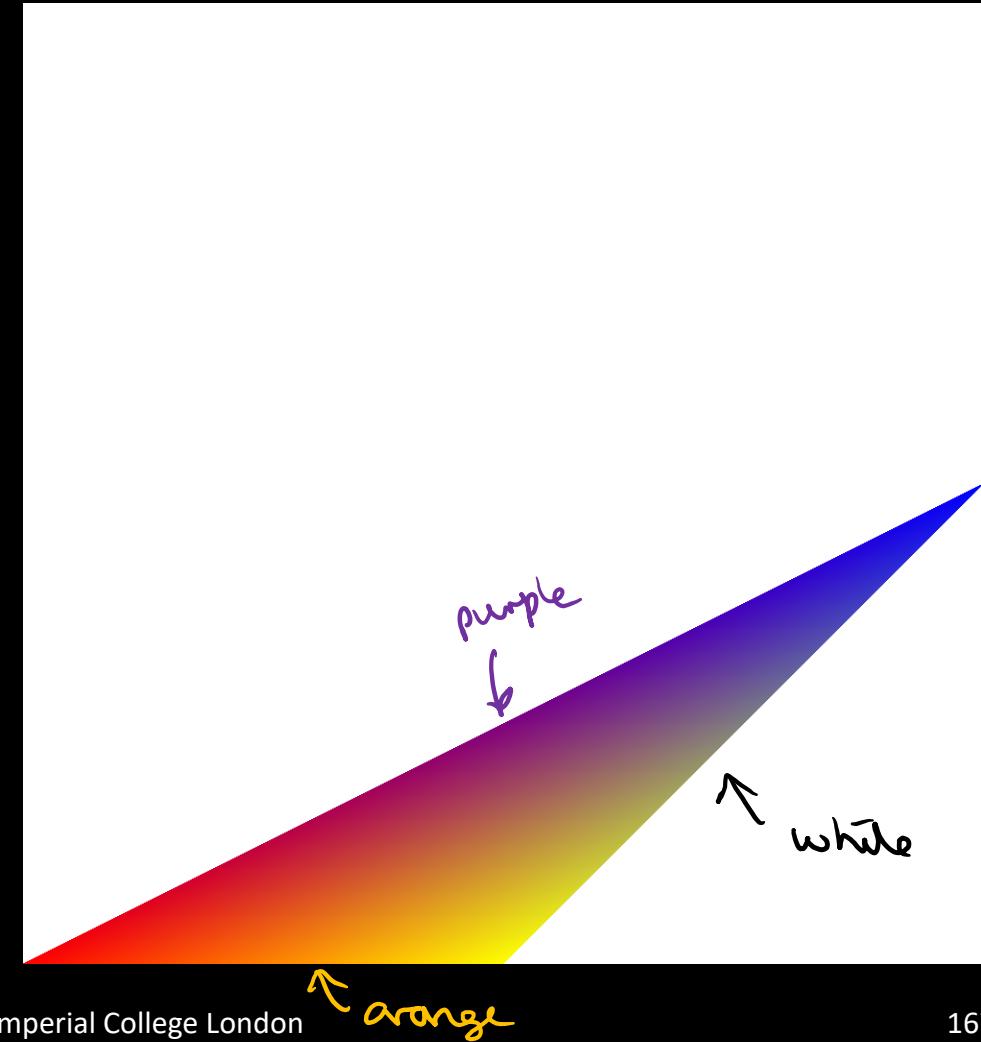
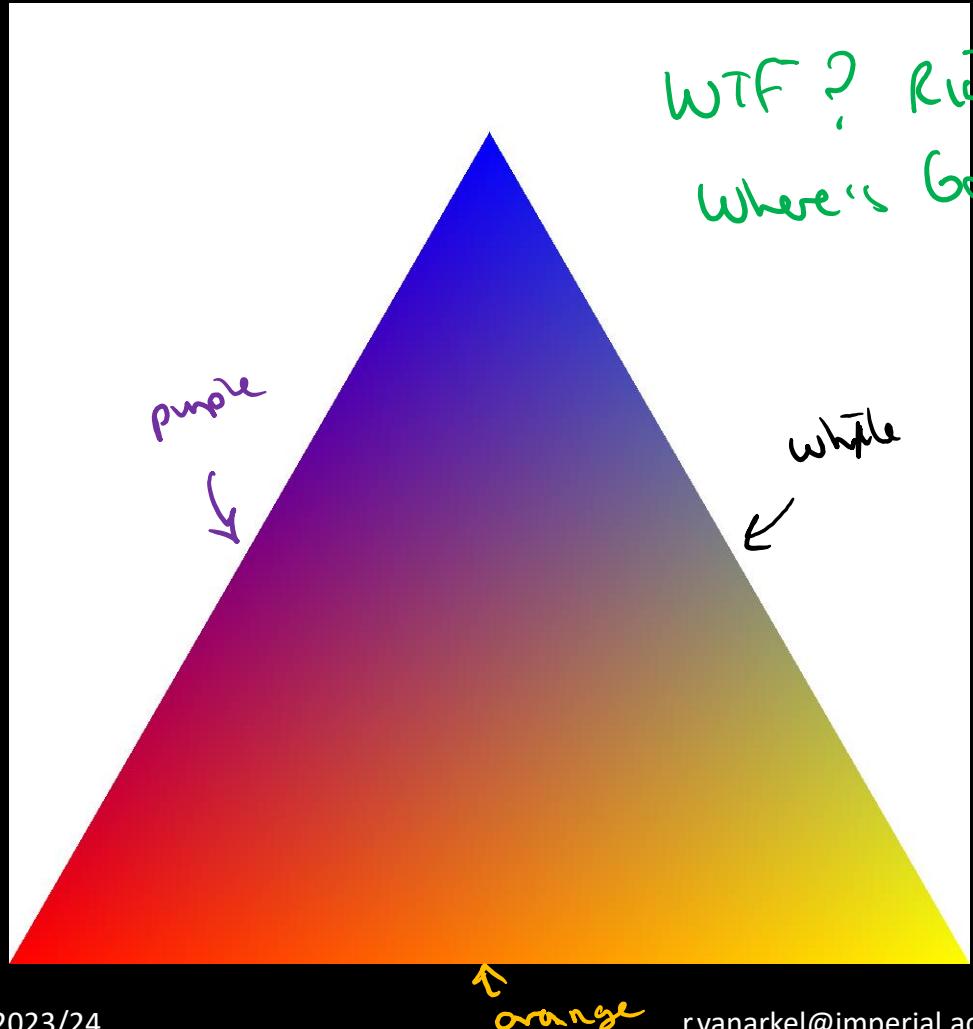
$$\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \begin{pmatrix} \underline{x} \\ \underline{y} \\ 1 \end{pmatrix} \quad \begin{array}{l} \underline{A} \underline{\lambda} = \underline{d} \\ \underline{\lambda} = \underline{A}^{-1} \underline{d} \end{array}$$

- and solved to find λ_1, λ_2 and λ_3 (the latter is more easily found from $\lambda_3 = 1 - \lambda_1 - \lambda_2$) for any \underline{r} :

$$\lambda_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

$$\lambda_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

Worked Example 11: Interpolating in a Triangle (Barycentric Method)



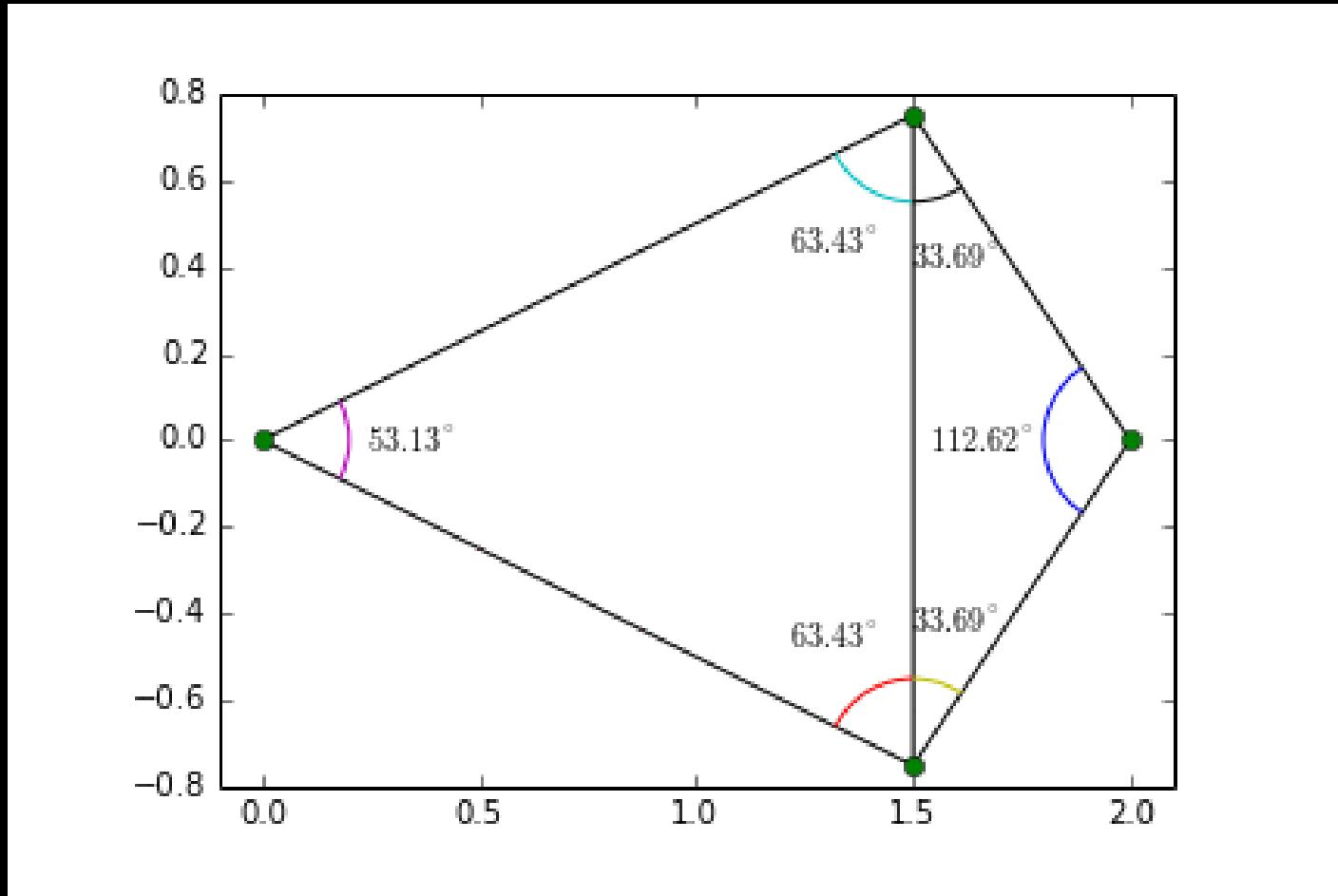
Worked Example 12: Interpolation of 2D Fields

- Demonstrated in lecture

Worked Example 13: Effect of Meshing

- Demonstrated in lecture
- For detailed explanation, including mathematical analysis see: *Weiming Cao, On the Error of Linear Interpolation and the Orientation, Aspect Ratio, and Internal Angles of a Triangle, 2005. SIAM J Numer Anal 43(1), 19–40.*

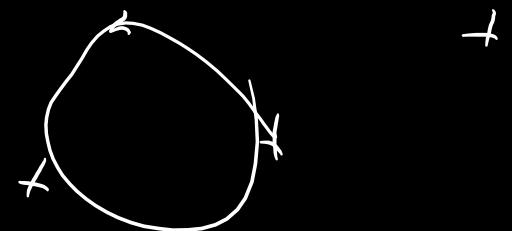
Delaunay Triangulation



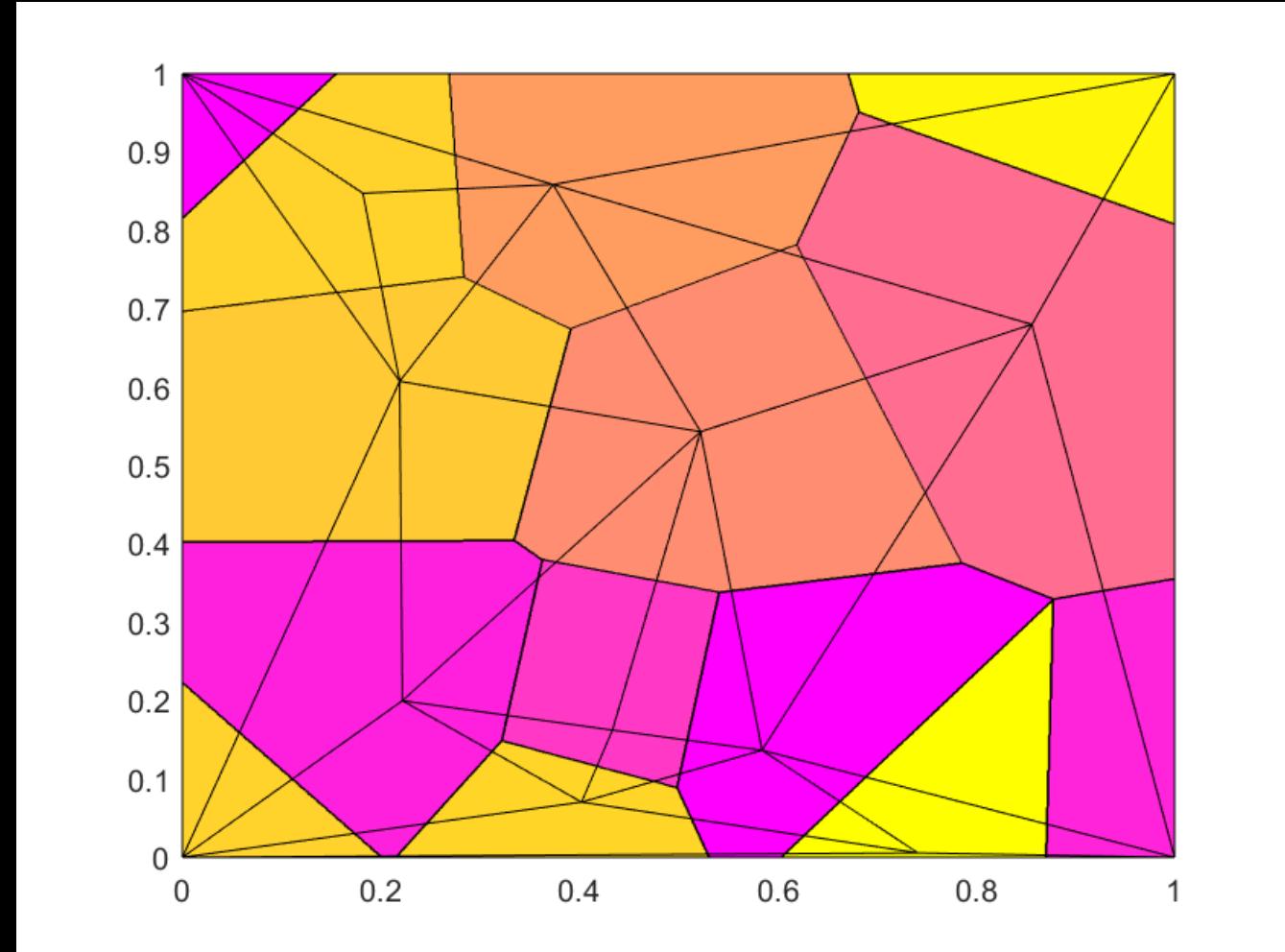
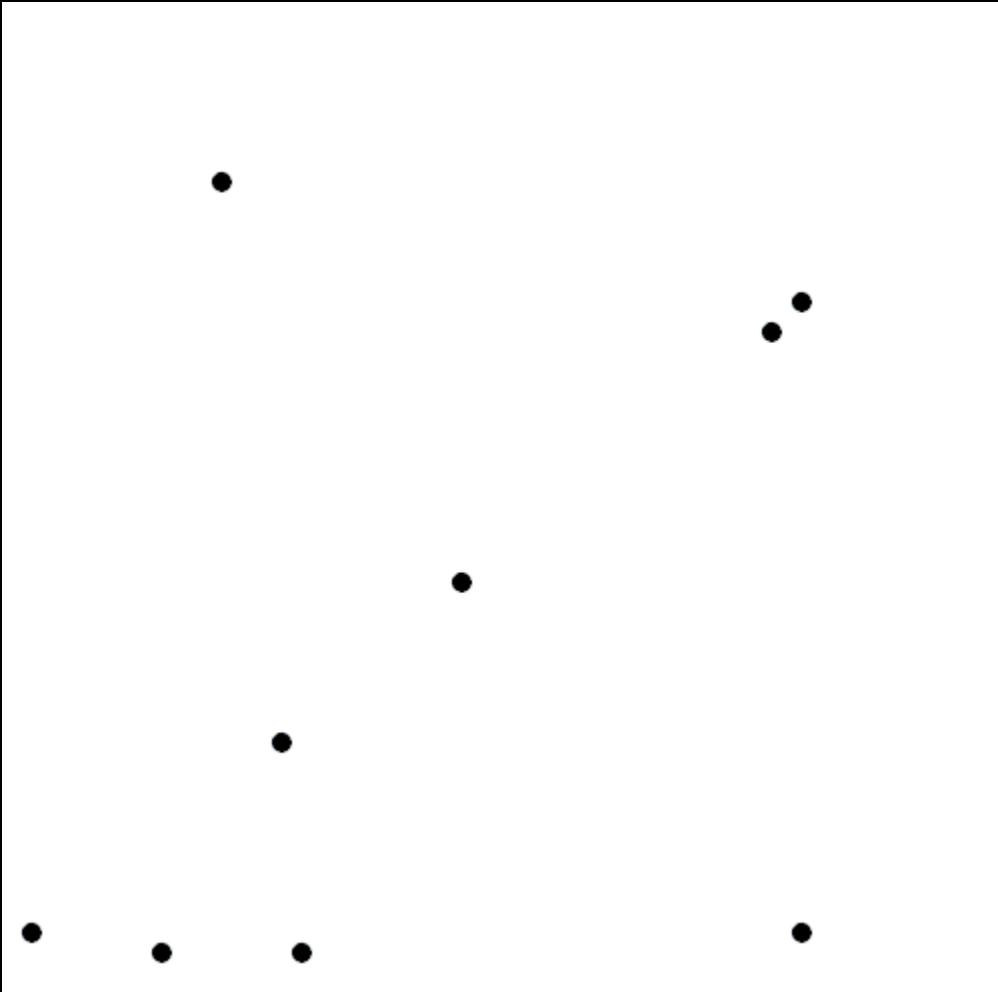
<https://nbviewer.jupyter.org/gist/goretkin/9790518>

Worked Example 14: Bowyer-Watson

```
function BowyerWatson (pointList) // pointList is a set of coordinates defining the points to be triangulated
    triangulation := empty triangle mesh data structure
    add super-triangle to triangulation // must be large enough to completely contain all the points in pointList
    for each point in pointList do // add all the points one at a time to the triangulation
        badTriangles := empty set
        for each triangle in triangulation do // first find all the triangles that are no longer valid due to the insertion
            if point is inside circumcircle of triangle
                add triangle to badTriangles
        polygon := empty set
        for each triangle in badTriangles do // find the boundary of the polygonal hole
            for each edge in triangle do
                if edge is not shared by any other triangles in badTriangles
                    add edge to polygon
        for each triangle in badTriangles do // remove them from the data structure
            remove triangle from triangulation
        for each edge in polygon do // re-triangulate the polygonal hole
            newTri := form a triangle from edge to point
            add newTri to triangulation
    for each triangle in triangulation // done inserting points, now clean up
        if triangle contains a vertex from original super-triangle
            remove triangle from triangulation
    return triangulation
```



Worked Example 15: Voronoi Diagrams (dual of Delaunay Triangulation)



Left) by Jahobr - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=61356414>. Right) Produced with RvA Worked Example Matlab Code

Summary

- Two polynomial interpolation schemes:
 - Lagrange Interpolation
 - Newton Divided Difference Interpolation
- Errors can be calculated exactly with:

$$\epsilon_n(x) = \frac{f^{n+1}(\xi)}{(n+1)!} \prod_{j=0}^n (x - x_j)$$

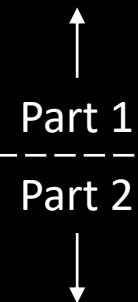
- Errors can be estimated crudely using a more accurate interpolation scheme with:
$$\epsilon_1 \approx \tilde{\alpha}_2 - \tilde{\alpha}_1$$
- More accurate interpolation is commonly achieved by increasing the order of the polynomial, but ‘stiff’ functions can cause problems – in these cases, spline interpolation is useful.
- Interpolation in 2D and 3D on regular grids:
 - Can be calculated sequentially using any technique (Lagrange linear, cubic splines, etc).
 - Matrix methods are easier to program and can be more computationally efficient.
- Interpolation for unstructured grids can be achieved by generating a triangular mesh (e.g. a Delaunay Triangulation) and then interpolating with Barycentric coordinates.

Chapter 2: Numerical Integration | More Accurate Methods (Simpsons & Gauss)

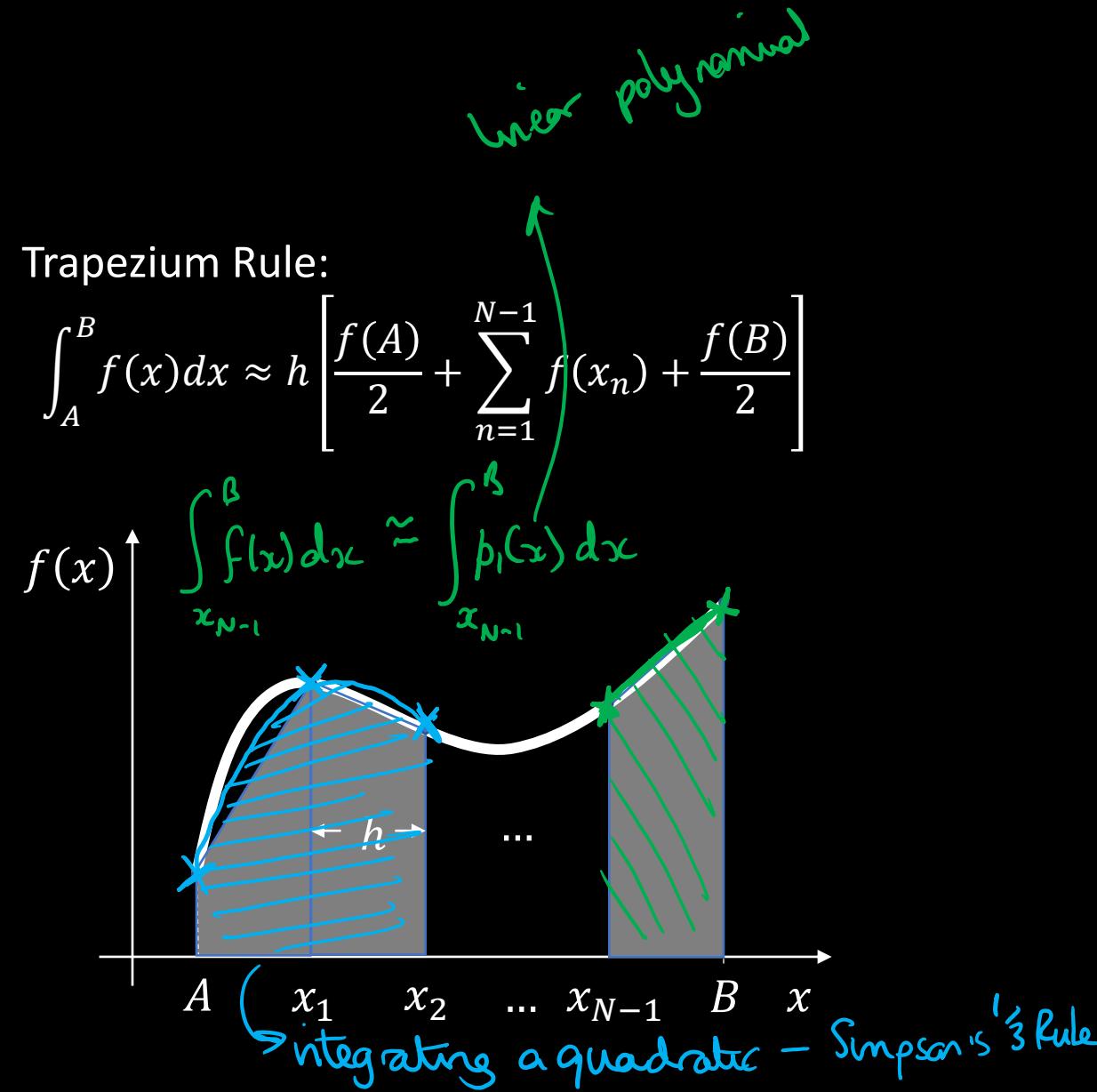
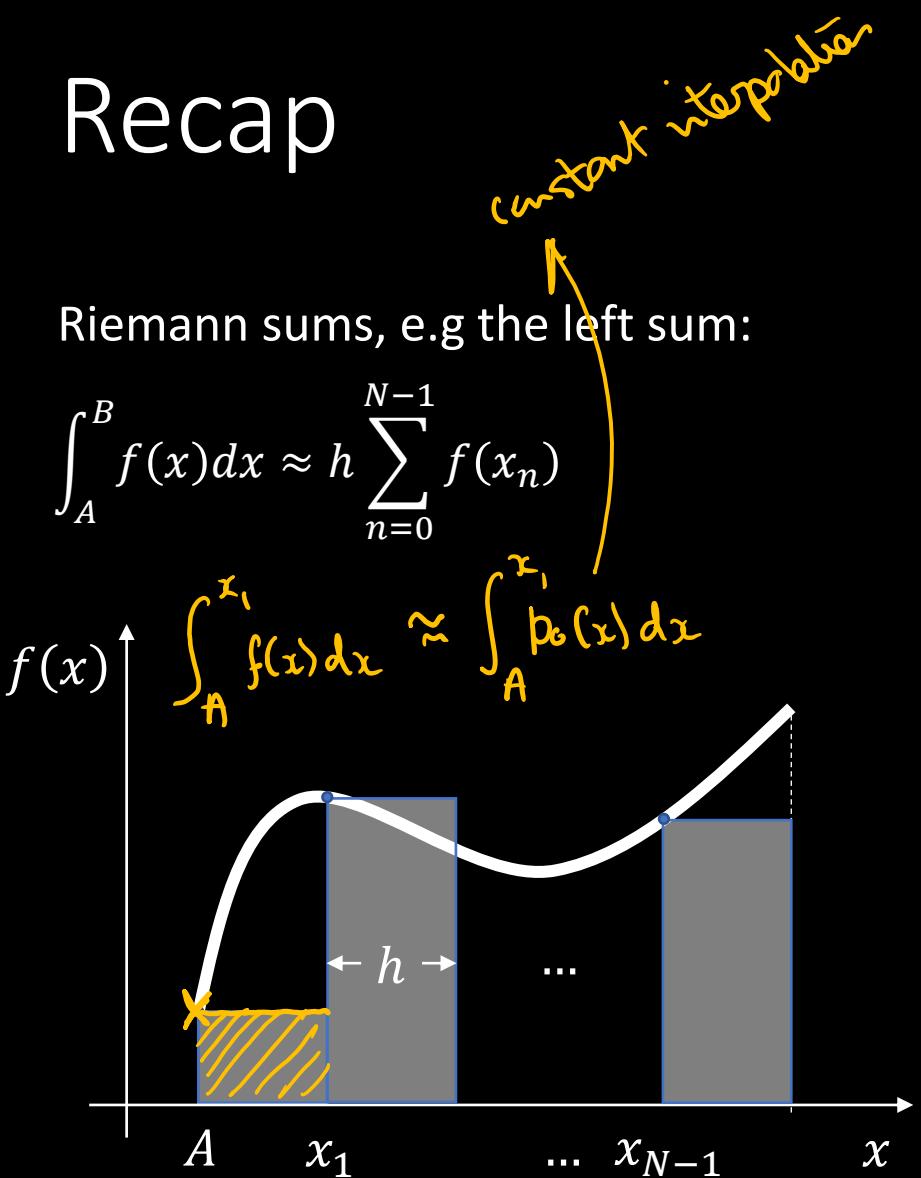
Numerical Integration Overview

Students should be able to:

- Divide an integral into appropriate subintervals and predict whether it will converge or diverge when evaluated numerically
- Numerically calculate integrals using different schemes:
 - Riemann Sums (derive & apply)
 - Trapezium Rule (derive & apply)
 - Simpson's Rule (derive & apply)
 - Adaptive Integration (apply)
 - Gauss Quadrature (apply)
- Estimate numerical integration error



Recap



Improve Accuracy using a Higher Order Interpolation Polynomial

To improve accuracy, a higher order polynomial can be used to interpolate between the points. Simpson's 1/3 Rule uses a quadratic interpolation polynomial:

$$\int_A^B f(x) dx \approx \int_A^B p_2(x) dx$$

Where:

$$p_2(x) = \sum_{j=0}^2 y_j \prod_{\substack{k=0 \\ k \neq j}}^2 \frac{(x - x_k)}{(x_j - x_k)}$$

Or expanded:

$$p_2(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Simpson's 1/3 Rule

With equal spacing $h = x_{j+1} - x_j$ (so $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$).

$$p_2(x) = y_0 \frac{(x - x_0 - h)(x - x_0 - 2h)}{2h^2} - y_1 \frac{(x - x_0)(x - x_0 - 2h)}{h^2} + y_2 \frac{(x - x_0)(x - x_0 - h)}{2h^2}$$

Integrating over single subinterval ($A = x_0, B = x_0 + 2h$):

$$\int_A^B f(x) dx \approx \int_{x_0}^{x_0+2h} p_2(x) dx$$

Substituting $t = x - x_0$ (for ease of integration):

$$\int_{x_0}^{x_0+2h} p_2(x) dx = \frac{1}{h^2} \int_0^{2h} \frac{y_0}{2}(t-h)(t-2h) + y_1 t(t-2h) + \frac{y_2}{2} t(t-h) dt$$

Integrating and evaluating between the limits:

$$\int_{x_0}^{x_2} f(x) dx \approx h \left(\frac{1}{3} y_0 + \frac{4}{3} y_1 + \frac{1}{3} y_2 \right) = \boxed{\underbrace{\frac{h}{3} \left(y_0 + 4y_1 + y_2 \right)}_{\text{Simpson's 1/3 Rule}}}$$

Compound Simpson's 1/3 Rule

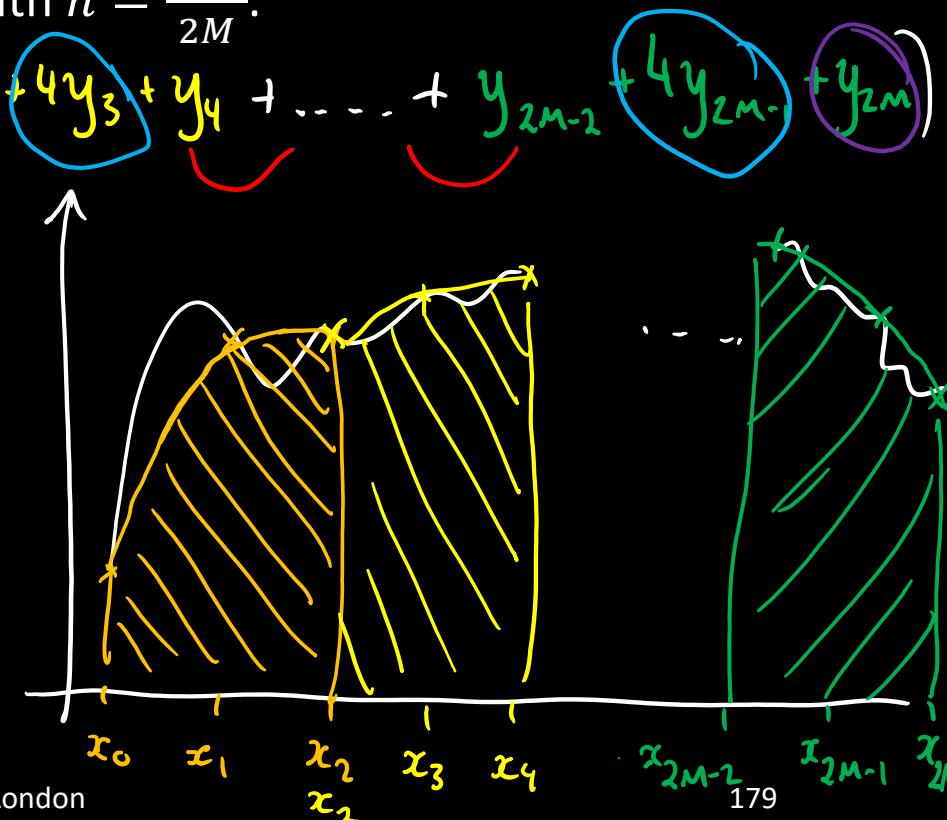
Each parabola integrated requires three known points and is integrated over a width $2h$.

The Compound Simpson's 1/3 Rule therefore requires an even number ($N = 2M$) of subintervals between A and B , where M is an integer and the subinterval width $h = \frac{B-A}{2M}$.

$$I \approx \frac{h}{3} (y_0 + 4y_1 + y_2 + y_3 + 4y_4 + \dots + y_{2M-2} + 4y_{2M-1} + y_{2M})$$

The compound rule is thus:

$$\int_A^B f(x) dx \approx \frac{h}{3} \left(y_0 + y_{2M} + 4 \sum_{\substack{i=1 \\ i \text{ odd}}}^{2M-1} y_i + 2 \sum_{\substack{i=2 \\ i \text{ even}}}^{2M-2} y_i \right)$$



Trapezium Error Calculation

- For each subinterval, the trapezium rule error is the area between the function and the linear interpolation polynomial:

$$\varepsilon = \int_{x_0}^{x_1} f(x)dx - \int_{x_0}^{x_1} p_1(x)dx = \int_{x_0}^{x_1} f(x) - p_1(x)dx$$

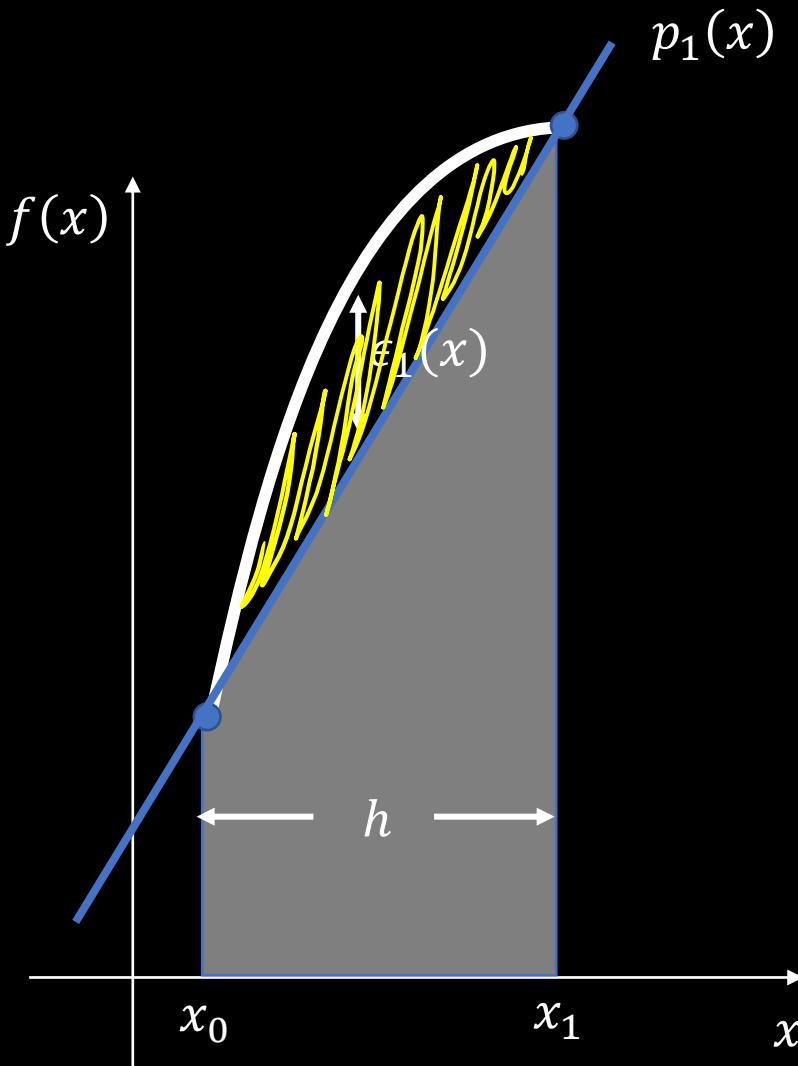
- Recalling that the error of an interpolation polynomial can be calculated exactly from:

$$\epsilon_n(x) = \frac{f^{n+1}(\xi)}{(n+1)!} \prod_{j=0}^n (x - x_j)$$

slide 110

- Therefore, for each subinterval the trapezium error is:

$$\varepsilon_{local} = \int_{x_0}^{x_1} \epsilon_1(x) dx = \int_{x_0}^{x_1} \frac{f''(\xi)}{2} (x - x_0)(x - x_1) dx$$



Trapezium Error

It is important to note that whilst ξ is commonly unknown, it will vary for each subinterval, and thus is a function of x . This makes the integration a challenge as:

$$\varepsilon_{local} = \int_{x_0}^{x_1} \frac{f^2(\xi(x))}{2} (x - x_0)(x - x_1) dx$$

To deal with this, we make the following substitutions: $x_1 = x_0 + h$, and $t = x - x_0$:

$$\varepsilon_{local} = \int_0^h \frac{f^2(\xi(t))}{2} t(t - h) dt$$

This change in variable makes it easy to spot that $t(t - h)$ does not change sign over the integration interval $[0, h]$. Therefore we can apply the Mean Value Theorem of Integral Calculus to find the solution:

$$\varepsilon_{local} = -\frac{h^3}{12} f^2(\xi) = -\frac{(B - A)^3}{12N^3} f^2(\xi)$$

As with for interpolation, as ξ is commonly unknown, practical application of this formula involves using error bounds (i.e. finding $\max_{t \in [0,h]} \{f^2(t)\}$ and using this instead of $f^2(\xi)$).

Compound Trapezium and Simpson's 1/3 Rule Error

If the trapezium rule is applied over N subintervals, there are N lots of this local error, therefore the total error is:

$$\varepsilon = N\varepsilon_{local} = -\frac{(B-A)^3}{12N^2}f^2(\xi) = -\frac{(B-A)h^2}{12}f^2(\xi)$$

$\varepsilon \propto h^2 \Rightarrow$ if we halve
' h ' we get $\frac{1}{4}$ of
the error

Using a similar approach, the error of the Compound Simpson's 1/3 Rule:

$$\varepsilon = -\frac{(B-A)^5}{180(2M)^4}f^4(\xi) = -\frac{(B-A)h^4}{180}f^4(\xi)$$

$\varepsilon \propto h^4 \Rightarrow$ if we halve ' h '
we get $\frac{1}{16}$ of error

Derivation: Talman LA, Simpson's Rule Is Exact for Quintics, 2006, American Mathematical Monthly 113(2):144-55.

$$\varepsilon_{trap} = -\frac{(B-A)h^2}{12} f^2(\xi) \quad \varepsilon_{simpson} = -\frac{(B-A)h^4}{180} f^4(\xi)$$

Error Estimation

slide 108

true value = $I_h + \varepsilon_h$

- If the data points have been obtained experimentally, and the $f(x)$ is not known, then the crude error estimate can be used.
- The more accurate numerical scheme used for comparison is that obtained by halving the step size h . Using the notation I_h to be the integral evaluated with step size h , and $I_{h/2}$ as the integral evaluated with step size $h/2$:

true value = true value
 $I_h + \varepsilon_h = I_{h/2} + \varepsilon_{h/2}$

- Rearranging:

$$\varepsilon_h - \varepsilon_{h/2} = I_{h/2} - I_h \Rightarrow \varepsilon_h = 4\varepsilon_{h/2}$$

- For the Trapezium Rule $\varepsilon \propto h^2$, and so $\varepsilon_{h/2} \approx \frac{1}{4}\varepsilon_h$ (approximately as ξ may be different for the two different step sizes). Therefore:

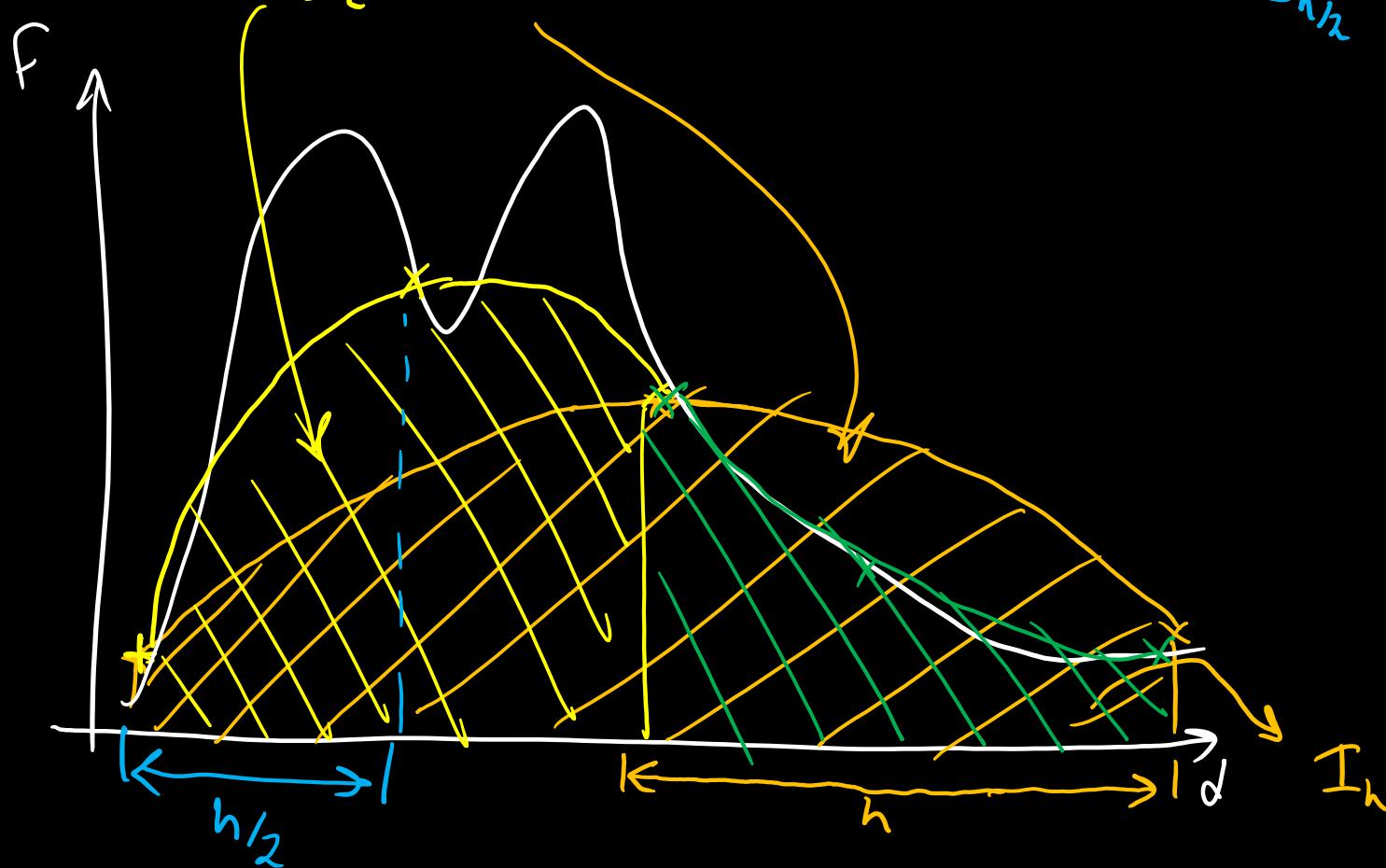
$$\varepsilon_{h/2} \approx \frac{1}{3}(I_{h/2} - I_h)$$

- For Simpson's 1/3 Rule $\varepsilon \propto h^4$, and so $\varepsilon_{h/2} \approx \frac{1}{16}\varepsilon_h$. Therefore:

$$\varepsilon_{h/2} \approx \frac{1}{15}(I_{h/2} - I_h)$$

Worked Example 16: Adaptive Integration

$$I_{h/2} = (I_{h/2})_L + (I_{h/2})_R$$



$$\epsilon_{h/2} \approx \frac{1}{15} (I_{h/2} - I_h)$$

- ↳ is this good enough?
- ↳ yes \Rightarrow stop, we have an answer
- ↳ no \Rightarrow find $I_{h/4}$ and repeat

Adaptive Integration

For computational efficiency, the step size can be varied based on the estimated error, thus reducing the number of calculations needed. The strategy is first define a tolerance, then:

1. Use Simpson's 1/3 Rule to integrate the data over the full interval width, and with two subintervals: i.e. with $h = (B - A)/2$, and $h_{1/2} = (B - A)/4$.
2. Estimate the error with $\varepsilon_{h/2} = \frac{1}{15}(I_{h/2} - I_h)$
3. If the estimated error is less than the tolerance, there is no need to further halve the step size for that subinterval. If it is not, halve the step size again and recalculate the error. For each subinterval, rather than compare the error for the complete integral against the tolerance for the whole integral, halve the tolerance as well and compare the error locally.
4. Repeat 3 until the error of each subinterval is less than the tolerance attributed to that subinterval.
5. Sum the results to find the integral over the full interval.

Degree of Precision for the Trapezium Rule

$$f(x) = mx + c \quad ; \quad f'(x) = m \quad ; \quad f''(x) = 0$$



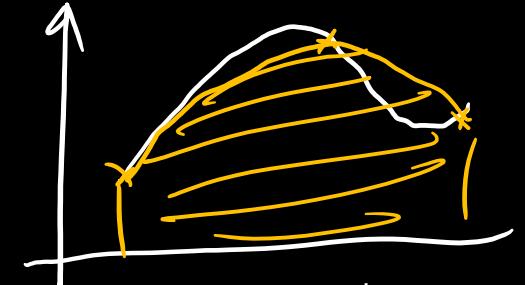
- Consider the error of the trapezium rule $\varepsilon = -\frac{(B-A)h^2}{12} f''(\xi)$: it is dependent on the second derivative of $f(x)$.
- Therefore, if $f(x)$ were a linear function, the error would always be zero (as the second derivative of a first order polynomial is zero).
- The Trapezium Rule has Degree of Precision = 1
- This makes intuitive sense: we have used a linear interpolation polynomial, which provides an exact fit for a linear function, and therefore it is expected that the error is zero.

Degree of Precision for Simpson's 1/3 Rule

- The error of the Simpson's 1/3 Rule is dependent of the 4th derivative of $f(x)$:

$$\varepsilon = -\frac{(B - A)h^4}{180} f^4(\xi)$$

- Therefore, the error for up to third order polynomials must be equal to zero (as they have zero 4th derivative).
- The Degree of Precision is 3.
- This is counter-intuitive. As we have used a quadratic interpolation polynomial, one would expect degree of precision 2 (perfect accuracy for quadratic functions). However we have seemingly gained accuracy, with Simpson's 1/3 Rule having zero error for cubic functions as well.
- The extra precision arises as a result of the properties of integrals of odd and even functions between symmetric limits, and the applicability of the Mean Value Theorem of Integral Calculus.
- The proof of this (e.g. *Talman LA 2006*) falls outside of the scope of this course – but the implications are highly relevant: with Degree of Precision 3, Simpson's 1/3 Rule provides sufficient accuracy for many practical problems.

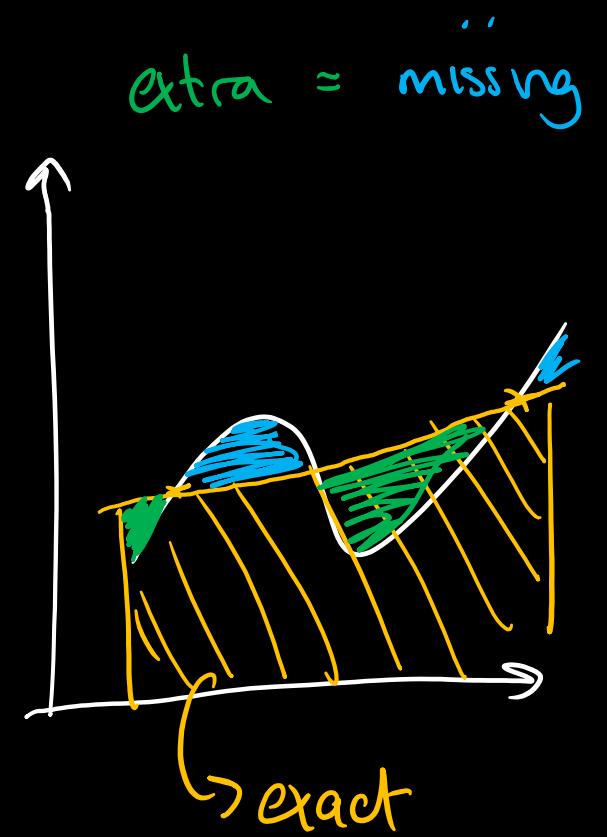
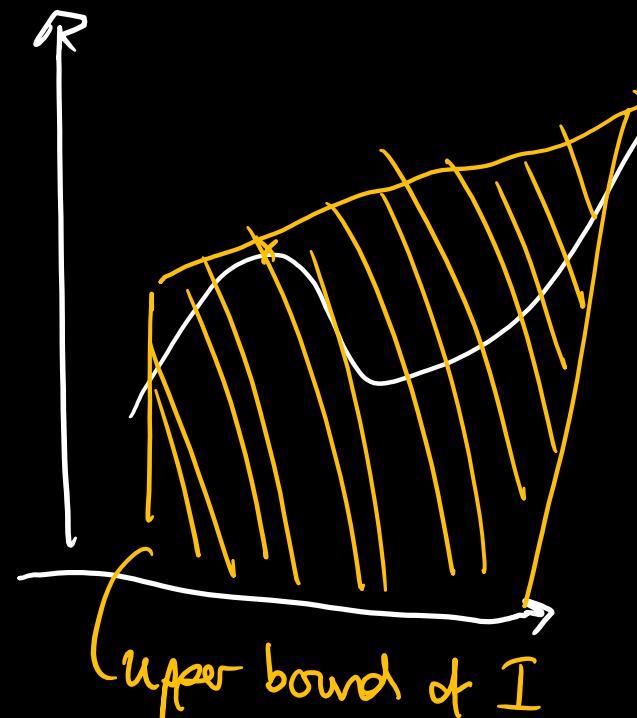
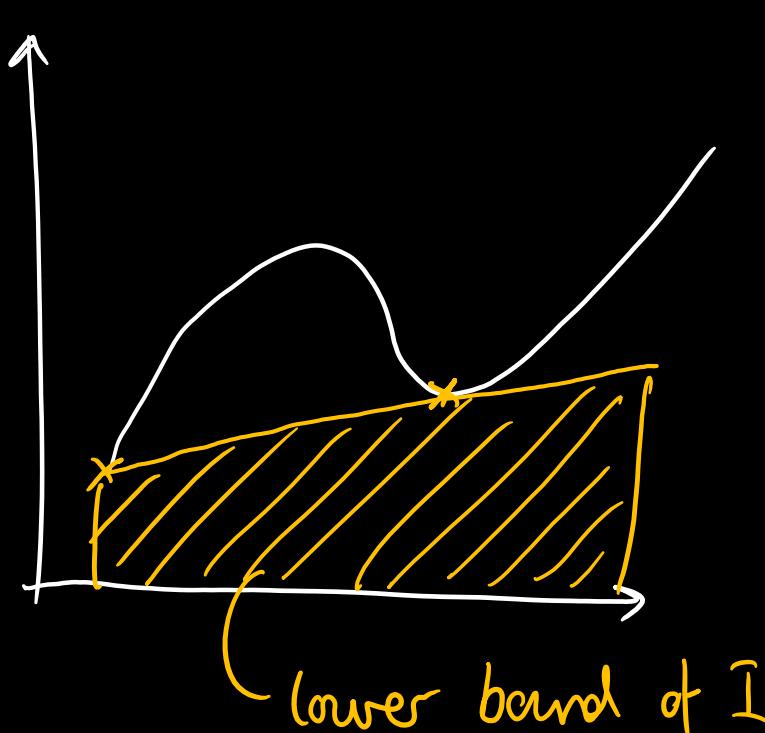


Maximum Degree of Precision

- All of our integration so far has relied on equidistant nodes. However, accuracy can be improved through careful choice of nodes.
- Gauss demonstrated that degree of precision $(2n - 1)$ can be achieved from n nodes if appropriate spacing and function weightings are used (compared to a maximum degree of precision of n if equidistant nodes are used).

we want maximise degree of precision for minimum nodes
↳ very very efficient computer codes

Gauss-Quadrature Diagram



Gauss-Legendre Quadrature (also known as Gauss Integration)

step 1) we convert x to t to get our axis from -1 to 1

- Using the substitution $x = \frac{1}{2}[A(1 - t) + B(t + 1)]$, a function $f(x)$ with $x \in [A, B]$ is mapped to $f(t)$ with $t \in [-1, 1]$. Noting that $dx = \frac{B-A}{2} dt$:

step 2) we look up the ideal node locations

$$\int_A^B f(x) dx = \frac{B-A}{2} \int_{-1}^1 f(t) dt$$

- The integral of $f(t)$ is then approximated with:

step 3) we integrate with the gauss formula

$$\int_{-1}^1 f(t) dt \approx \sum_{j=1}^n w_j f(t_j)$$

- Where w_j are predetermined weighting coefficients, dependent on n (the number of nodes t_j), but not dependent on $f(t)$
 - For interest: the locations of nodes t_j are found as the solution of the j th root of n th Legendre polynomial, and then the weightings can be found using Lagrange's interpolation polynomial. For further details see Dahlquist G. and Bjorck A., *Numerical Methods*, ISBN 0136273157
- Careful with the change in convention: for the Trapezium and Simpson's Rules we started labelling at node 0, and thus required $(n + 1)$ nodes. For Gauss Quadrature, notation commonly starts at node 1, therefore requiring n nodes.

Gauss-Legendre Quadrature Nodes and Weighting

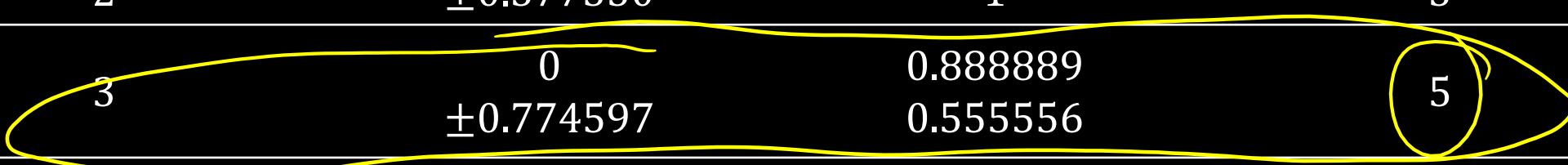
n	Nodes t_j	Weightings w_j	Degree of Precision
1	0	2	1
2	$\pm \frac{1}{\sqrt{3}}$	1	3
3	0, $\pm \sqrt{\frac{3}{5}}$	$\frac{8}{9}, \frac{5}{9}$	5
4	$\pm \sqrt{\frac{3}{7} - \frac{2}{7}\sqrt{\frac{6}{5}}}$, $\pm \sqrt{\frac{3}{7} + \frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{18 + \sqrt{30}}{36}, \frac{18 - \sqrt{30}}{36}$	7
5	0, $\pm \frac{1}{3}\sqrt{5 - 2\sqrt{\frac{10}{7}}}$, $\pm \frac{1}{3}\sqrt{5 + 2\sqrt{\frac{10}{7}}}$	$\frac{128}{225}, \frac{322 + 13\sqrt{70}}{900}, \frac{322 - 13\sqrt{70}}{900}$	9

number of nodes I need (circled 3) is highlighted in green.

polynomial equivalence (circled 5) is highlighted in green.

Gauss-Legendre Quadrature Nodes and Weighting

n	Nodes t_j	Weightings w_j	Degree of Precision
1	0	2	1
2	± 0.577350	1	3
3	0 ± 0.774597	0.888889 0.555556	5
4	± 0.339981 ± 0.861136	0.652145 0.347855	7
5	0 ± 0.538469 ± 0.906180	0.568889 0.478629 0.236927	9



Worked Example 17: Gauss Quadrature

Evaluate the integral of the following 5th order polynomial exactly using Gauss Integration with three nodes. Confirm the result by comparing the analytical solution:

$$I = \int_{1/5000}^1 02x^{05} + 2016x^4 + 81x^2 + 23x^3 + 12x^1 - 3x^0 \, dx$$



The function commemorates Leicester City Winning the Premier League: from 5000:1 odds at the start of the season, they won the league on 2nd May 2016. Their season totalled 81 points having recorded 23 wins, 12 draws and 3 losses.



Substitution: $x = \frac{1}{2}[A(1-t) + B(t+1)]$

$$\int_A^B f(x)dx \approx \frac{B-A}{2} \sum_{j=1}^n w_j f(t_j)$$

Worked Example 17

step 1) we convert $x \rightarrow t$

$$x = \frac{1}{2} \left[\frac{1}{5000} (1-t) + 1(t+1) \right] = \frac{4999t}{10000} + \frac{5001}{10000}$$

$$f(t) = 2\left(\frac{4999}{10000}t + \frac{5001}{10000}\right)^5 + 2016\left(\frac{4999}{10000}t + \frac{5001}{10000}\right)^4 + 81\left(\frac{4999}{10000}t + \frac{5001}{10000}\right)^2 \\ + 23\left(\frac{4999}{10000}t + \frac{5001}{10000}\right)^3 + 12\left(\frac{4999}{10000}t + \frac{5001}{10000}\right) - 3$$

step 2) we look up ideal nodes & weight

$$t_1 = -\sqrt{\frac{3}{5}} ; t_2 = 0 ; t_3 = \sqrt{\frac{3}{5}}$$

$$w_1 = 5/9 ; w_2 = 8/9 ; w_3 = 5/9$$

step 3) we integrate with the formula

$$\int_{-\frac{1}{5000}}^{\frac{1}{5000}} f(x) dx = \frac{1 - \frac{1}{5000}}{2} \left(\frac{5}{9} f(t_1) + \frac{8}{9} f(t_2) + \frac{5}{9} f(t_3) \right)$$

$$= \frac{4999}{10000} \left(\frac{5}{9} \times -0.2529... + \frac{8}{9} \times 152.2994... + \frac{5}{9} \times 1338.3123... \right) = 439.2839330931173$$

default precision
for matlab & python

Worked Example 17

Analytically:

$$I = \left[\frac{2}{6}x^6 + \frac{2016}{5}x^5 + \frac{81}{3}x^3 + \frac{23}{4}x^4 + \frac{12}{2}x^2 - 3x^1 \right]_{1/5000}^1$$

$$I = \frac{6863811454579958187567333}{5000^6}$$

$$I = 439.2839330931172980854171328246593475341796875$$

Which is exactly the same as the numerical result (with Python/MatLab default precision)

Summary

- Improve integration accuracy using higher order interpolation polynomials
- Estimate errors by integrating the interpolation error
- For experimental data, Simpson's 1/3 Rule provides excellent accuracy for minimal computational expense due to its “unexpected” 3rd degree accuracy
- Crude error estimated can be used to implement adaptive integration routines
- Gauss Quadrature provides the best possible accuracy for when integrating known functions.

Chapter 2: Numerical Differentiation | Methods and Errors



[Support Us](#) [Login](#) [Search](#)

[Programmes and Prizes](#)

[Policy and Resources](#)

[Education and Skills](#)

[News and Events](#)

[About Us](#)

[Programmes and prizes](#) > [Programmes](#) > [UK Grants and Prizes](#) > [Support for education](#) > Engineering Leaders Scholarship

Engineering Leaders Scholarship



Numerical Differentiation Overview

Students should be able to:

- Numerically calculate derivatives (of any order) using forward, backward and central finite difference methods.
- Develop more accurate numerical differentiation schemes using Lagrange interpolation
- Demonstrate the accuracy of a numerical differentiation schemes using Taylor Series
- Describe how floating point arithmetic and noise can affect the accuracy of numerical differentiation

Forward Finite Difference Approximation

Consider the definition of a derivative:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

This can be numerically approximated by simply not taking the limit as $h \rightarrow 0$, rather just using small h :

$$f'(x) \approx \frac{f(x + h) - f(x)}{h}$$

Or using our previous notation for discrete data:

$$y'_n \approx \frac{y_{n+1} - y_n}{h} = \frac{\Delta y_n}{h}$$

This is a forward difference approximation for a derivative.

on next slide

Forward Difference Accuracy

Taylor expansion with $a = x_n$ & $h = h$

$$f(x_n + h) = f(x_n) + hf'(x_n) + \frac{h^2}{2!}f''(x_n) + \frac{h^3}{3!}f'''(x_n) + \dots$$

$$y_{n+1} = y_n + hy_n' + \frac{h^2}{2!}y_n'' + \frac{h^3}{3!}y_n''' + \dots$$

Rearrange for y_n' :

$$y_n' = \frac{y_{n+1} - y_n}{h} - \frac{h}{2!}y_n'' - \frac{h^2}{3!}y_n''' - \dots$$

error = true - approximation

$$\epsilon = -\frac{h}{2!}y_n'' - \frac{h^2}{3!}y_n''' - \dots$$

If h is small, then h^2 is very small, $h \gg h^2$, so our error is dominated by the first term

$\epsilon = O(h)$, first order accurate

Backward Finite Difference Approximation

If we shift backward by the node spacing h :

$$f'(x) \approx \frac{f(x) - f(x - h)}{h}$$

Or using our previous notation for discrete data:

$$y'_n \approx \frac{y_n - y_{n-1}}{h} = \frac{\nabla y_n}{h}$$

This is a backward difference approximation for a derivative.

Backward Difference Accuracy

Using the Taylor expansion:

$$f(x_n - h) = y_{n-1} = y_n - hy'_n + \frac{h^2}{2!}y''_n - \frac{h^3}{3!}y'''_n \dots$$

Rearranging for the first derivative term y'_n :

$$y'_n = \frac{y_n - y_{n-1}}{h} + \frac{h}{2!}y''_n - \frac{h^2}{3!}y'''_n \dots$$

As with the forward finite difference approximation, the backward finite difference derivative approximation is also first order accurate:

$$y'_n = \frac{\nabla y_n}{h} + O(h)$$

*Backward Differentiation has same accuracy
as Forwards*

Central Finite Difference Approximation

If we shift backward by half the node spacing ($h/2$):

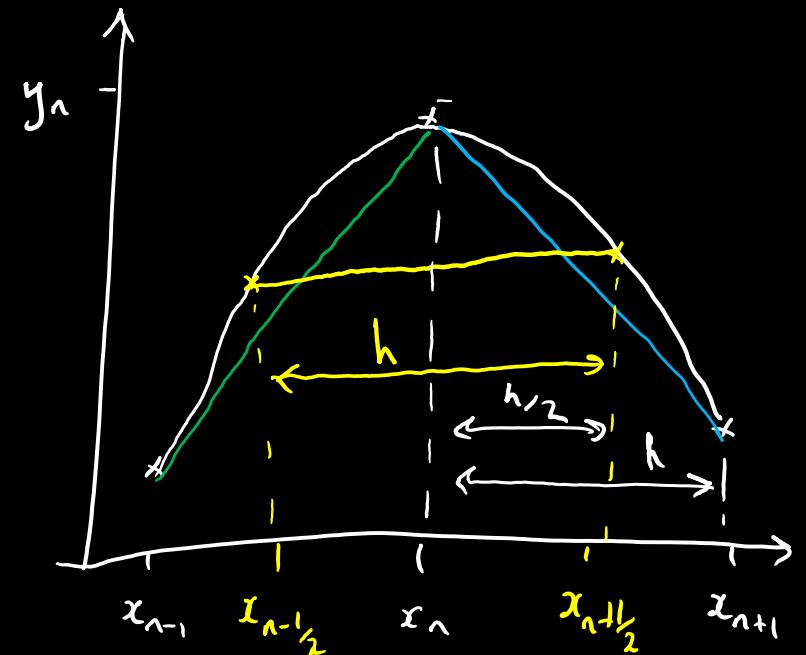
$$f'(x) \approx \frac{f\left(x + \frac{h}{2}\right) - f\left(x - \frac{h}{2}\right)}{h}$$

Or for discrete data:

$$y'_n \approx \frac{y_{n+1/2} - y_{n-1/2}}{h} = \frac{\delta y_n}{h}$$

This is a central difference approximation for a derivative.

 = Forwards
 = Backwards
 = Central



Central Difference Accuracy

Two Taylor expansions are needed:

$$f\left(x_n + \frac{h}{2}\right) = y_{n+1/2} = y_n + \frac{h}{2}y'_n + \underbrace{\left(\frac{h^2}{2^2 2!}y''_n\right)}_{\text{These cancel}} + \frac{h^3}{2^3 3!}y'''_n \dots \quad \#1$$

$$f\left(x_n - \frac{h}{2}\right) = y_{n-1/2} = y_n - \frac{h}{2}y'_n + \underbrace{\left(\frac{h^2}{2^2 2!}y''_n\right)}_{\text{These cancel}} - \frac{h^3}{2^3 3!}y'''_n \dots \quad \#2$$

$$(\#1) - (\#2):$$

$$\delta y_n = hy'_n + \frac{h^3}{2^2 3!}y'''_n + \dots \quad h^2 \gg h^3$$

Rearranging:

$$y'_n = \frac{\delta y_n}{h} - \frac{h^2}{2^2 3!}y'''_n = \frac{\delta y_n}{h} + O(h^2)$$

The central difference derivative approximation is second order accurate.

Practical Application of Central Finite Difference Approximation

The central difference approximation requires the function to be evaluated at the midpoint between the nodes. For experimental data, where nodal midpoints are not known, this is not possible.

Instead, the central difference is evaluated by taking the average of the central difference approximations for $y_{n+1/2}$ and $y_{n-1/2}$:

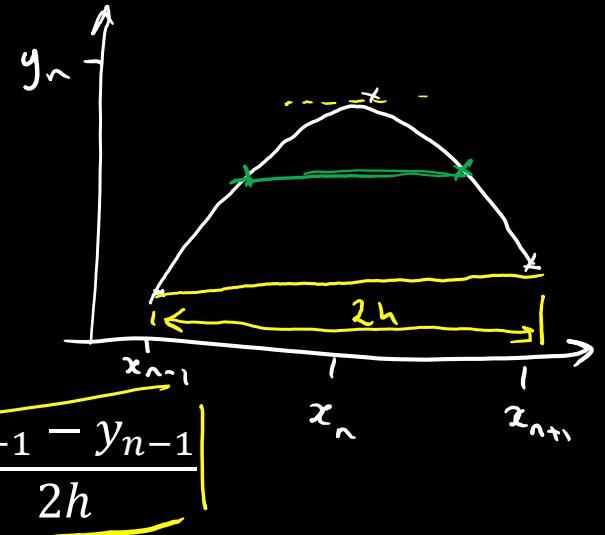
↳ this method is over the top here, but will work for higher order derivatives as well

$$y'_{n+1/2} \approx \frac{\delta y_{n+1/2}}{h} = \frac{y_{n+1} - y_n}{h}$$

$$y'_{n-1/2} \approx \frac{\delta y_{n-1/2}}{h} = \frac{y_n - y_{n-1}}{h}$$

Averaging:

$$y'_n \approx \frac{1}{2} \left(\frac{\delta y_{n+1/2}}{h} + \frac{\delta y_{n-1/2}}{h} \right) = \frac{(y_{n+1} - y_n) + (y_n - y_{n-1})}{2h} = \boxed{\frac{|y_{n+1} - y_{n-1}|}{2h}}$$



Worked Example 18: Numerical Differentiation Error

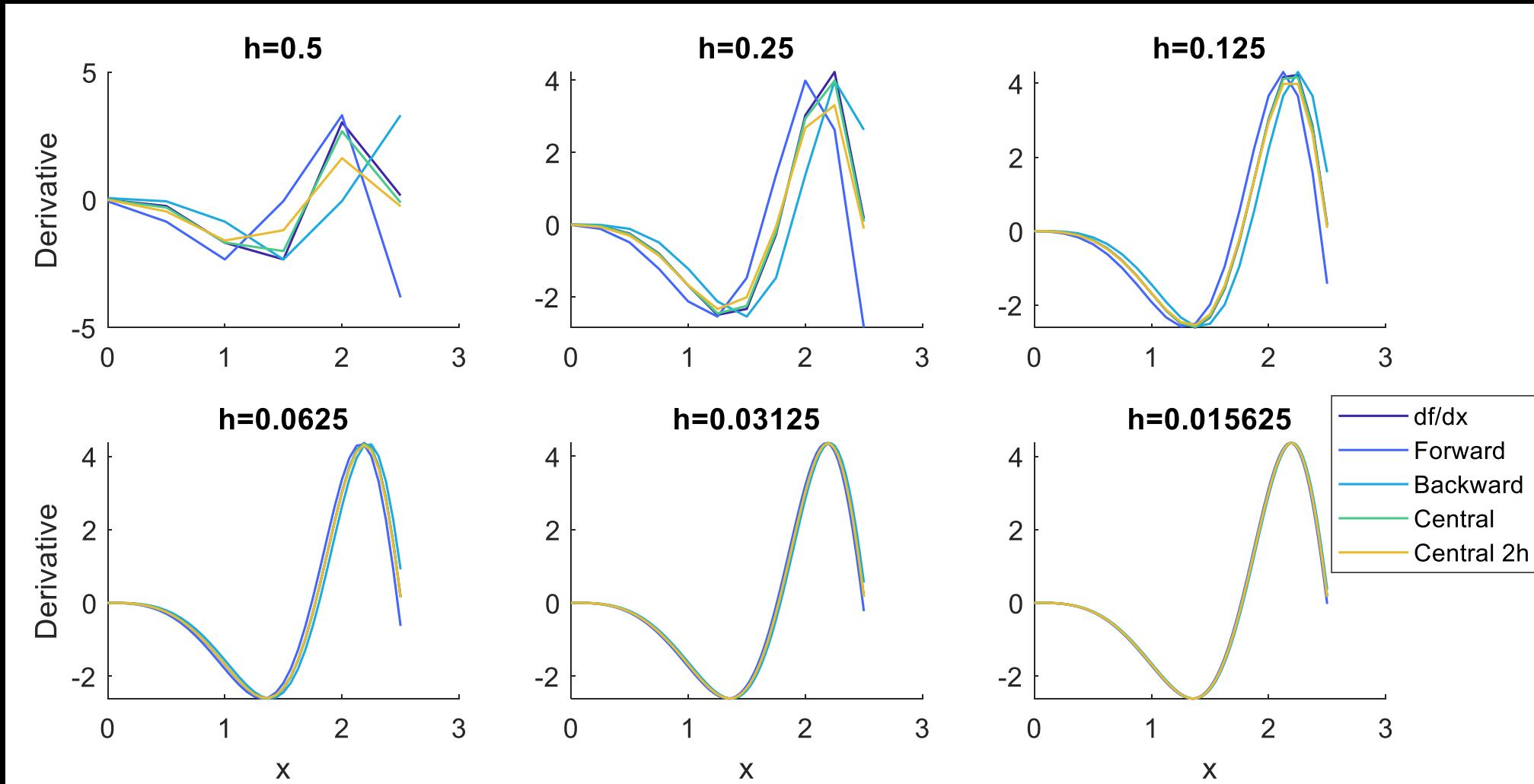
$$f''(x) = -2x \sin(x^2)$$

Estimate the derivative of $f(x) = \cos(x^2)$ for $0 \leq x \leq 0.25$ with $h = 0.5$ and compare to the exact value. How does the error changes as the node spacing is halved?

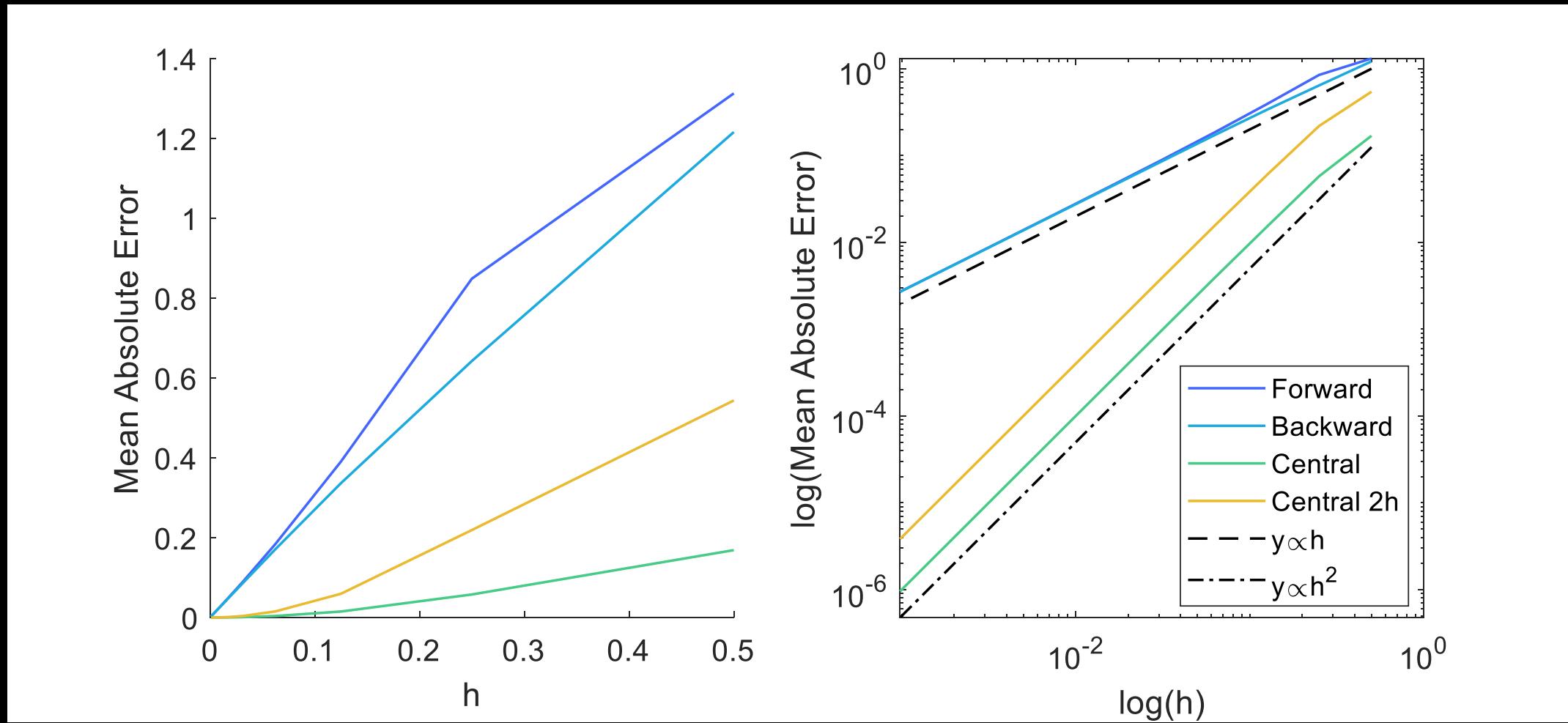
x	$f(x)$	$\frac{df}{dx}$	$\frac{\Delta y_n}{h}$	$\frac{\nabla y_n}{h}$	$\frac{\delta y_n}{h}$	$\frac{\delta_{2h} y_n}{2h}$
0	1	0	-0.0622	n/a	n/a	n/a
0.5	0.9689	-0.2474	-0.8572	-0.0622	-0.3042	-0.4597
1	0.5403	-1.6829	-2.3370	-0.8572	-1.6753	-1.5971
1.5	-0.6282	-2.3342	-0.0509	-2.3370	-2.0103	-1.1939
2	-0.6536	3.0272	3.3062	-0.0509	2.6798	1.6276
2.5	0.9994	0.1659	n/a	3.3062	n/a	n/a

Note that $\frac{\delta y_n}{h}$ was evaluated by calculating $f(x)$ at the nodal midpoints which aren't shown in the table

Worked Example 18: Numerical Differentiation Error



Worked Example 18: Numerical Differentiation Error



Higher order derivatives

Higher order derivatives are found by re-applying the finite difference approximation but with $f^n(x)$ rather than with $f(x)$.

The second forward derivative approximation is:

$$\begin{aligned}f''(x) &\approx \frac{f'(x+h) - f'(x)}{h} = \frac{\frac{f(x+2h) - f(x+h)}{h} - \frac{f(x+h) - f(x)}{h}}{h} \\&= \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}\end{aligned}$$

Alternative notation:

$$y_n'' = \frac{\Delta^2 y_n}{h^2} = \frac{\Delta y_{n+1} - \Delta y_n}{h^2} = \frac{(y_{n+2} - y_{n+1}) - (y_{n+1} - y_n)}{h^2} = \frac{y_{n+2} - 2y_{n+1} + y_n}{h^2}$$

K th order derivative approximation

Forward $y_n^k \approx \frac{\Delta^k y_n}{h^k}$

Backward $y_n^k \approx \frac{\nabla^k y_n}{h^k}$

Central $y_n^k \approx \frac{\delta^k y_n}{h^k}$

Where the difference operators can be calculated using binomial coefficients:

$$\Delta^k y_n = \sum_{i=0}^k (-1)^i \binom{k}{i} y_{n+(k-i)}$$

$$\nabla^k y_n = \sum_{i=0}^k (-1)^i \binom{k}{i} y_{n-i}$$

$$\delta^k y_n = \sum_{i=0}^k (-1)^i \binom{k}{i} y_{n+(k/2-i)}$$

Remember that:

$$\binom{k}{i} = \frac{k!}{i!(k-i)!}$$

K th order central finite difference approximations

$$\text{Central } y_n^k \approx \frac{\delta^k y_n}{h^k}$$

For odd order derivatives calculated with a central finite difference method (i.e. odd k), the approximation requires knowledge of the nodal midpoints. These midpoints are typically unknown when using experimental data. Therefore, as we did for the first order central difference, we average the k th order central differences evaluated at $x_n \pm \frac{h}{2}$.

Improving Accuracy

slide 105

Accuracy can be improved using Lagrange polynomials. Recall that, between x_n and x_{n+2} :

$$p_2(x) = y_n \frac{(x - x_{n+1})(x - x_{n+2})}{(x_n - x_{n+1})(x_n - x_{n+2})} + y_{n+1} \frac{(x - x_n)(x - x_{n+2})}{(x_{n+1} - x_n)(x_{n+1} - x_{n+2})} + y_{n+2} \frac{(x - x_n)(x - x_{n+1})}{(x_{n+2} - x_n)(x_{n+2} - x_{n+1})}$$

Rewriting for equal nodal spacing (i.e. $x_{n+1} = x_n + h$, $x_{n+2} = x_n + 2h$):

$$p_2(x) = y_n \frac{(x - x_n - h)(x - x_n - 2h)}{2h^2} - y_{n+1} \frac{(x - x_n)(x - x_n - 2h)}{h^2} + y_{n+2} \frac{(x - x_n)(x - x_n - h)}{2h^2}$$

Differentiating (using product rule):

$$p'_2(x) = y_n \frac{2x - 2x_n - 3h}{2h^2} - y_{n+1} \frac{2x - 2x_n - 2h}{h^2} + y_{n+2} \frac{2x - 2x_n - h}{2h^2}$$

Improving Accuracy

$$\underline{p'_2(x)} = \frac{1}{2h} \left(y_n \frac{2x - 2x_n - 3h}{h} - y_{n+1} \frac{4x - 4x_n - 4h}{h} + y_{n+2} \frac{2x - 2x_n - h}{h} \right)$$

We then evaluate $p'_2(x)$ at x_n , $x_n + h$ and $x_n + 2h$ to obtain “three-point formulas” approximations for y'_n , y'_{n+1} and y'_{n+2} :

$$x = x_n \Rightarrow p'_2(x_n) \rightarrow \text{Forwards More Accurate Method} \quad y'_n = \frac{1}{2h} (-3y_n + 4y_{n+1} - y_{n+2})$$

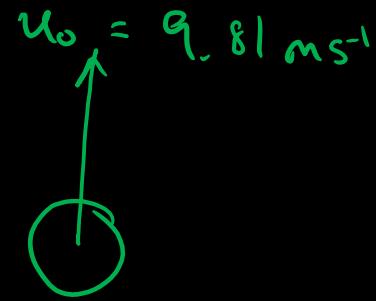
↑ 1 node shift: $y'_n = \frac{1}{2h} (y_{n+1} - y_{n-1})$

$$x = x_{n+1} \Rightarrow p'_2(x_{n+1}) \rightarrow \text{Central More Accurate Method} \quad y'_{n+1} = \frac{1}{2h} (-y_n + y_{n+2})$$

↓ Shift LHS & RHS by 2 nodes!

$$x = x_{n+2} \Rightarrow p'_2(x_{n+2}) \rightarrow \text{Backward formula} \quad y'_{n+2} = \frac{1}{2h} (y_n - 4y_{n+1} + 3y_{n+2}) \quad y'_n = \frac{1}{2h} (y_{n+2} - 4y_{n+1} + 3y_n)$$

An advantage of these formulae is improved accuracy without the need to know y_{n-1} or $y_{n-1/2}$. This is particularly relevant for time based data, starting at $t = 0$, as these points are not known.



Worked Example 19: Effects of Noise

A ball is thrown vertically upwards, with initial velocity 9.81 m/s and its time varying position is measured with an optical tracking system. Numerically calculate the ball's velocity and acceleration (with central difference methods), and its absement (with Simpson's Rule). Compare the numerical results to an analytical model of its trajectory. First do the calculations assuming zero measurement error. Then with a random measurement error with maximum amplitude of 1 mm.

$$\text{Calculate Absement} = \int y(t) dt = \frac{h}{3} \left[y_0 + y_{2m} + 4 \sum_{\substack{i=1 \\ i \text{ odd}}}^{2m-1} y_i + 2 \sum_{\substack{i=2 \\ i \text{ even}}}^{2m-2} y_i \right]$$

Measure Height = $y(t)$

$$\text{Calculate Velocity} = y'(t) \approx \frac{y_{n+1} - y_{n-1}}{2h}$$

$$\text{Calculate Acceleration} = y''(t) \approx \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2}$$

Worked Example 19: Effects of Noise

Demonstrated in lecture.

Take home:

- Numeric integration typically reduces (or eliminates) errors/noise
 - Integration smooths data
- Numeric differentiation amplifies errors/noise
 - Differentiation roughens data

Summary

- Forward and Backward divided difference derivative approximations are first order accurate (prove with Taylor Series)
- Central divided difference derivative approximations are second order accurate
- Higher order derivatives can be calculated recursively.
- Differentiate Lagrange interpolation polynomials to develop formulae with improved accuracy.
- Differentiation amplifies noise whilst integration smooths data.

Chapter 3: Numerical Solutions to ODES

Dr Richard J van Arkel

Numerical Methods for Solving ODEs: Overview

Students should be able to:

- Apply multi-stage explicit methods to initial value problems to numerically estimate the solutions of first order differential equations (ODEs) including: Forward Euler, Heun and Runge-Kutta 4th order.
- Demonstrate the accuracy of these techniques with Taylor series and through integration analogy.
- Derive multistep methods from interpolation polynomials including: Adams methods and Backward Differentiation Formula.
- Analyse the convergence of multistep methods (consistency and zero-stability).
- Apply implicit methods to initial value ODE problems including: Backward Euler and Trapezium Rule.
- Analyse the absolute stability of both explicit and implicit methods.
- Use matrices to solve systems of linear ODEs and analyse their eigenvalue stability.
- Identify and describe the consequences of a ‘stiff’ system of ODEs
- Express higher order ODE problems as a system of first order ODEs
- Understand stability in the complex plane
- Derive numerical schemes for non-linear problems

Chapter 3: Numerical ODEs | Forward Euler, Heun and RK4

Revision: Ordinary Differential Equations

Forced Mass Spring Damper:

$$m \frac{d^2x}{dt^2} + c \frac{dx}{dt} + kx = F(t)$$

Pendulum:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l} \sin \theta$$

Blasius Solution:

$$\frac{d^3f}{d\eta^3} + \frac{1}{2}f \frac{df}{d\eta} = 0$$

Radioactive Decay:

$$\frac{dN}{dt} = \lambda N \quad (\text{where } \lambda < 0)$$

Series RLC circuit:

$$\frac{d^2i}{dt^2} + \frac{R}{L} \frac{di}{dt} + \frac{1}{LC} i = 0$$

Revision: First Order Ordinary Differential Equations

These take the general form:

$$\frac{dy}{dt} = f(t, y)$$

First order $\frac{dy}{dt}$
Second order $\frac{d^2y}{dt^2}$

For example, for radioactive decay:

- Our ODE is $\frac{dy}{dt} = \lambda y$ and hence our $f(t, y) = \lambda y$

However, depending on the physics of the problem, the $f(t, y)$ could be anything. For example these made-up functions:

- If our ODE is $\frac{dy}{dt} = ye^{-t^2}$ then our $f(t, y) = ye^{-t^2}$
- If our ODE is $\frac{dp}{dx} = \tan(\sqrt{px^2})$ then our $f(p, x) = \tan(\sqrt{px^2})$

Revision: Initial Value Problems

- If we know the physics (our ODE) and a starting point (an initial condition), and we want to predict what happens in the future: we have an initial value problem.
- For example, for radioactive decay:
 - Our number of atoms, y , decreases with time: $y = f(t)$
 - The rate of decrease of atoms is proportional to the current number of atoms:
$$\frac{dy}{dt} = f(t, y) = \lambda y \quad \text{where } \lambda \text{ is a negative decay constant.}$$
 - We know the initial number of undecayed atoms: $y(0) = Y_0 = \text{constant}$
 - We want to know how the number of atoms decreases with time: we need to solve our ODE.
 - We will use this as our base example because:
 - We understand the physics, allowing us to focus on the numerical methods.
 - We know the analytical solution allowing comparison to an exact value: $y = Y_0 e^{\lambda t}$
- These problems are usually time based, but can be position based (e.g. a cantilevered beam). For the following slides, I will use time, but every time I write or say time, know that it could be a distance (or any other parameter) instead.

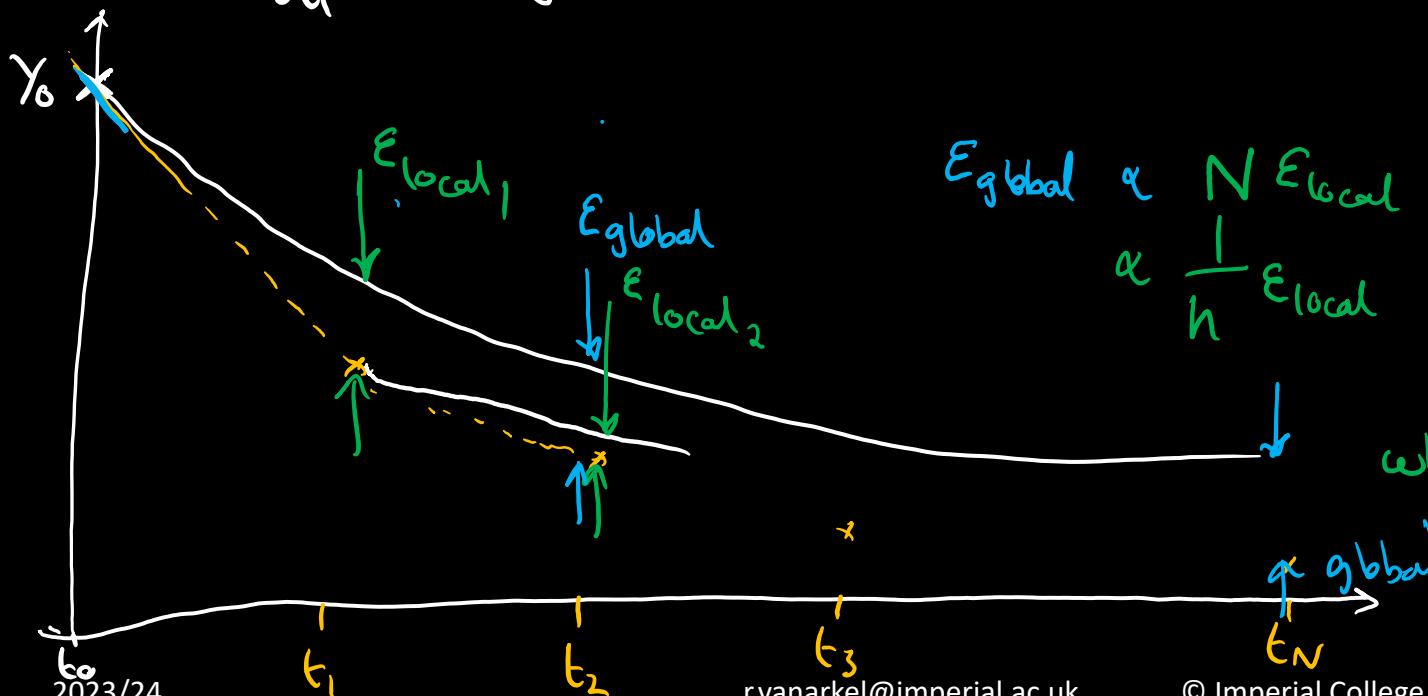
An Example Initial Value Problems: Radioactive Decay

We know our physics: $\frac{dy}{dt} = \text{some known function}$

& we know our initial point

We want to find all subsequent y values

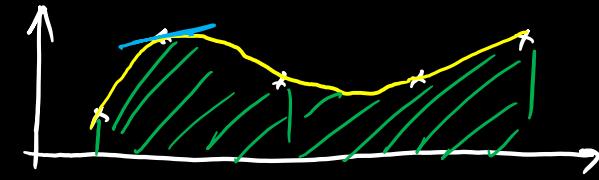
$$\frac{dy}{dt} = f(t, y)$$



$$\begin{aligned} \epsilon_{\text{global}} &\propto N \epsilon_{\text{local}} \\ &\propto \frac{1}{h} \epsilon_{\text{local}} \end{aligned}$$

where $N = \frac{\text{total time}}{h}$

Previously we have known all $y(t)$ nodes



Solving Initial Value Problems Numerically: Forward Euler Method

↗ slide 202

Consider forwards numerical differentiation:

$$y'_n \approx \frac{y_{n+1} - y_n}{h}$$

Rearranging:

$$y_{n+1} \approx y_n + hy'_n$$

Where:

- $h = \Delta t = t_{n+1} - t_n$ = the node spacing/step size (also known as time step for time based problems)
- $y_n = y(t_n)$ = the current (and hence known) value of y
- $y_{n+1} = y(t_{n+1}) = y(t_n + h) = y(t_n + \Delta t)$ = the next value of y
- $y'_n = \frac{dy(t_n)}{dt}$ = the current value of the derivative of y = a function of y and t

From our first order ODE, we know:

$$\left. \begin{aligned} y'_n &= \frac{dy(t_n)}{dt} = f(t_n, y_n) \\ y_{n+1} &\approx y_n + hf(t_n, y_n) \end{aligned} \right\} \xrightarrow{\text{Forwards Euler Method}}$$

Therefore:

Accuracy of the Forward Euler Method

To calculate the error, we compare the forward Euler Method to the Taylor expansion of y_{n+1} :

$$y_{n+1|exact} = y(t_n + h) = y_n + hy'_n + \frac{h^2}{2!}y''_n + \frac{h^3}{3!}y'''_n \dots = y_n + hf(t, y) + \frac{h^2}{2!}y''_n + \frac{h^3}{3!}y'''_n \dots$$

Therefore the error for a single step (known as the local error) is:

$$\varepsilon_{local} = y_{n+1|exact} - y_{n+1|Euler} = \frac{h^2}{2!}y''_n + \frac{h^3}{3!}y'''_n \dots$$

If the step size is small, then $h^2 \gg h^3$, thus the local truncation error:

$$\varepsilon_{local} \approx \frac{h^2}{2}y''_n = O(h^2)$$

If the problem is solved for a total time τ , with a constant step size, then the total number of steps, N , is:

$$N = \frac{\tau}{h}$$

Therefore our global truncation error is:

$$\varepsilon_{global} \propto N\varepsilon_{local} \approx \frac{\tau h}{2}y''_n = O(h)$$

The Forward Euler Method is *first order accurate*.

Worked Example 20: Forward Euler for Radioactive Decay

If there are 100 atoms initially of a radioactive element characterised by decay constant $\lambda = -0.1$, estimate the number of atoms that remain after 50 s using the forward Euler method with a time step of 5 s. Compare to the analytical result.

Our ODE: $\frac{dy}{dt} = f(t, y) = -0.1y$

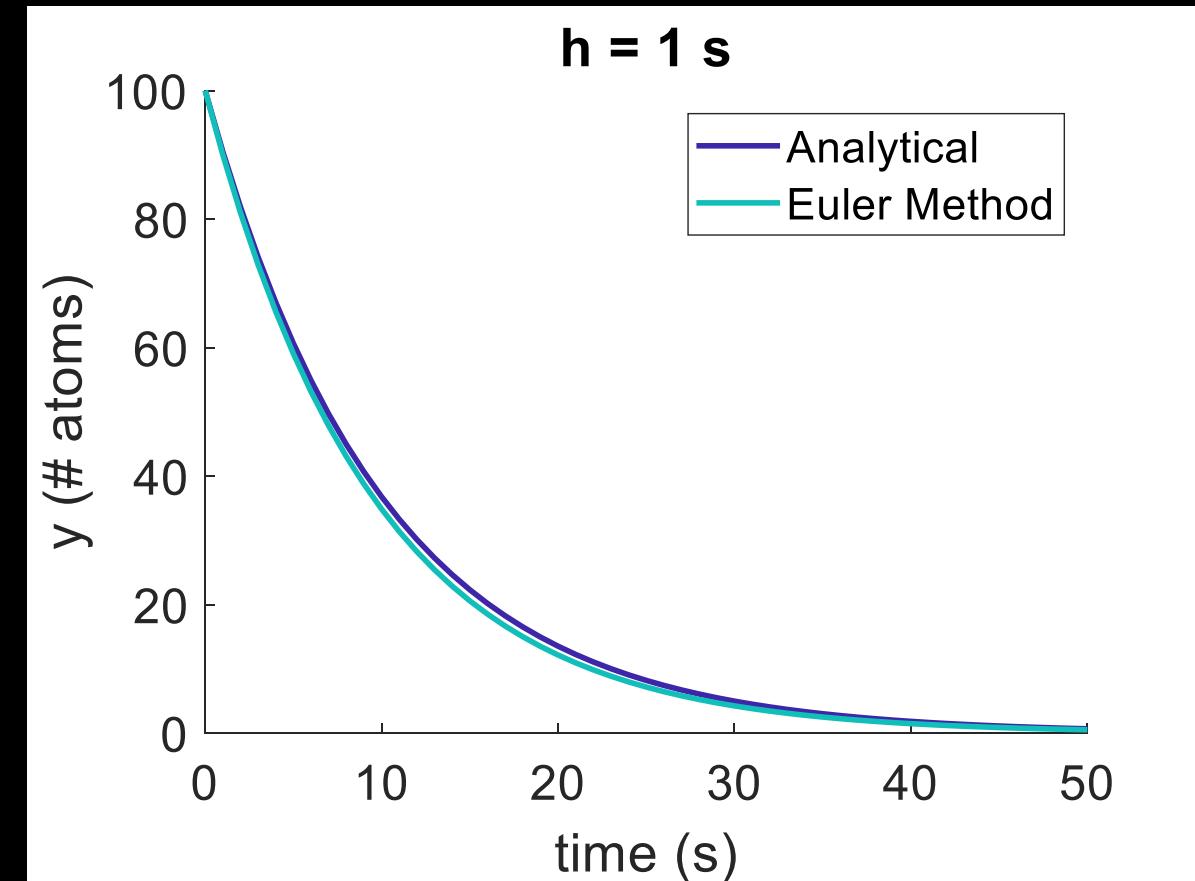
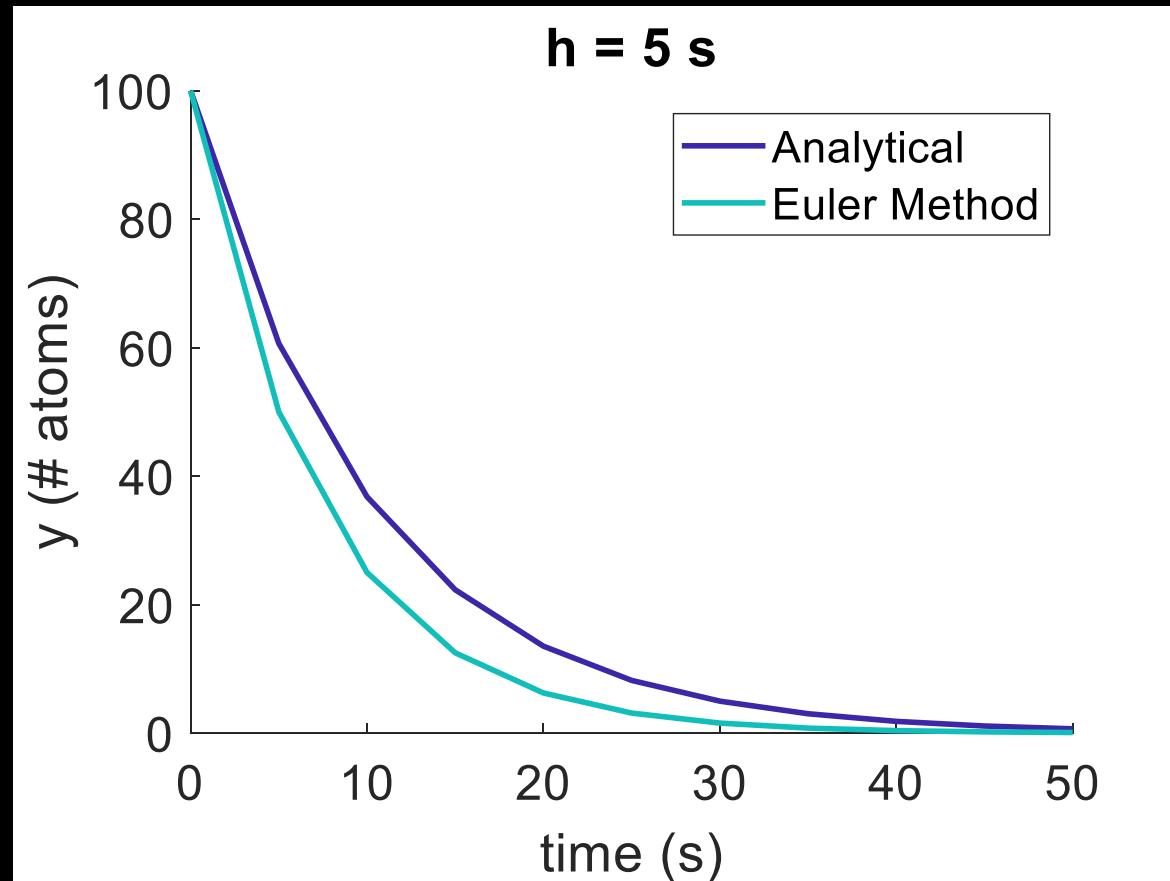
Euler method: $y_{n+1} \approx y_n + hf(t_n, y_n) = y_n + h(-0.1y_n) = y_n(1 - 0.1h) = 0.5y_n$

Therefore:

n	0	1	2	3	4	5	6	7	8	9	10
t_n	0	5	10	15	20	25	30	35	40	45	50
y_n	100	50	25	12.5	6.25	3.125	1.5625	0.7813	0.3906	0.1953	0.0977
y_{n+1}	50	25	12.5	6.25	3.125	1.5625	0.7813	0.3906	0.1953	0.0977	

$$y_1 = 0.5^1 \times y_0$$
$$y_{n+1} = 0.5^{n+1} y_0$$

Worked Example 20: Forward Euler for Radioactive Decay



Improving Accuracy with Heun's Method (a.k.a. Improved Euler)

Heun's method is known as a predictor-corrector method.

- First we predict a value for y at the next time step using Euler:

temporary value $\rightarrow y_{n+1}^* = y_n + hf(t_n, y_n)$

- Then we use this predicted value to find a more accurate estimate for y_{n+1} :

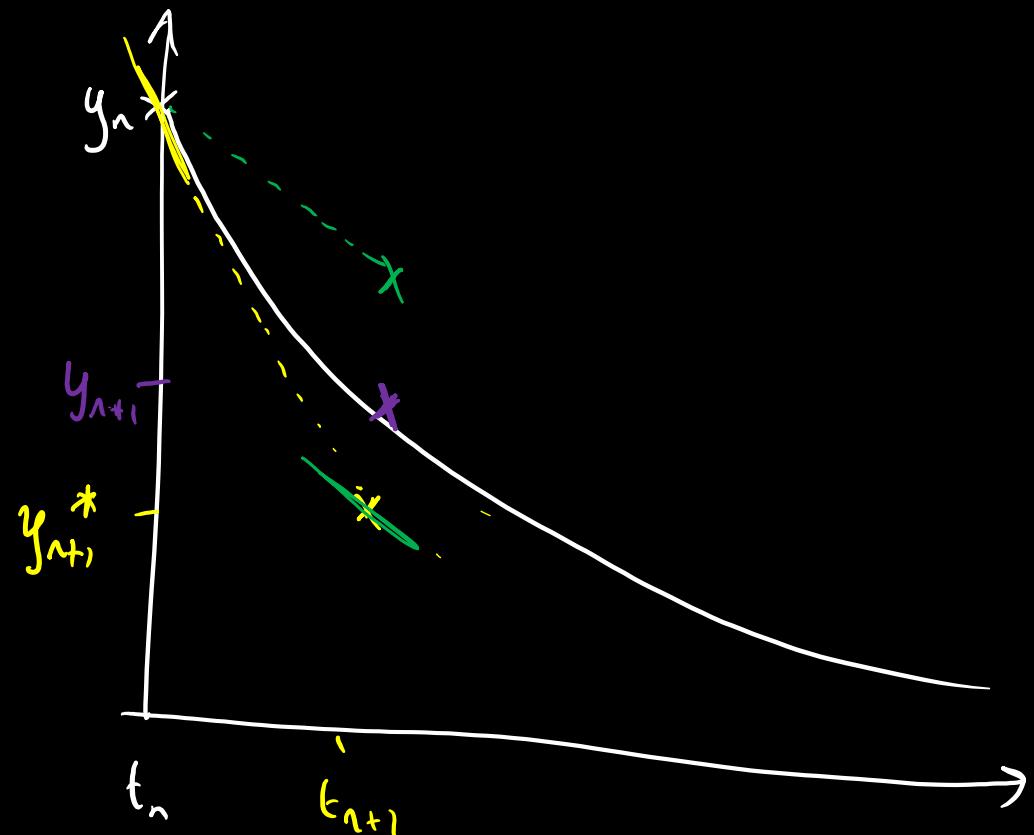
result \downarrow $y_{n+1} = y_n + \frac{h}{2} [f(t_n, y_n) + f(t_{n+1}, y_{n+1}^*)]$

- This is also sometimes written as:

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf(t_n + h, y_n + k_1)$$

$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$



Accuracy of Heun's Method

To calculate the error, we again compare to the Taylor expansion of y_{n+1} :

$$y_{n+1|exact} = y(t_n + h) = y_n + hy'_n + \frac{h^2}{2!} y''_n + \frac{h^3}{3!} y'''_n + \dots$$

we want to do true - approximation, and when do this, we will need expression for y'_n, y''_n, y'''_n

$$y' = f(t, y(t)) ; y'' = \frac{d}{dt}(f(t, y(t))) = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} \frac{dy}{dt} = f_t + f_y f \leftarrow \text{Chain Rule for Total Derivative}$$

$$y''' = \frac{d}{dt}(f_t + f_y f) = \frac{d}{dt}(f_t) + f_y \frac{d}{dt}(f) + f \frac{d}{dt}(f_y) \leftarrow \text{product rule.}$$

$$\begin{aligned} \frac{\partial^2 f}{\partial y \partial t} &= \frac{\partial^2 f}{\partial t \partial y} \\ &= f_{tt} + f_{ty} \frac{dy}{dt} + f_y(f_t + f_y f) + f(f_{yt} + f_{yy} \frac{dy}{dt}) \leftarrow \text{Chain Rule Total Deriv.} \\ &= f_{tt} + 2f_{ty}f + f_y f_t + f_y^2 f + \cancel{f f_{yt}} + \cancel{f_{yy} f^2} \end{aligned}$$

Therefore:

$$y_{n+1|exact} = y_n + \underbrace{hf_n}_{y'_n} + \frac{h^2}{2!} \underbrace{(f_{t_n} + f_y f_n)}_{y''_n} + \frac{h^3}{3!} \underbrace{(f_{tt_n} + 2f_{ty_n} f_n + f_y f_{t_n} + f_y^2 f_n + f_{yy} f_n^2)}_{y'''_n} + \dots$$

Accuracy of Heun's Method Continued

$$\text{Heun: } y_{n+1|Heun} = y_n + \frac{h}{2} f_n + \frac{h}{2} f(t_n + h, y_n + hf_n)$$

Taylor expansion with two variables ($a = t_n, b = y_n, h = h, k = hf_n$):

$$\begin{aligned} f(t_n + h, y_n + hf_n) &= f(t_n, y_n) + [hf_t + hf_n f_y]_n + \frac{1}{2} [h^2 f_{tt} + 2h^2 f_n f_{ty} + h^2 f_n^2 f_{yy}]_n \\ &= f_n + h(f_{t_n} + f_n f_{y_n}) + \frac{h^2}{2}(f_{tt_n} + 2f_n f_{ty_n} + f_n^2 f_{yy_n}) + \dots \end{aligned}$$

Substituting into the Heun expression:

$$y_{n+1|Heun} = y_n + \underbrace{\frac{h}{2} f_n + \frac{h}{2} f_n}_{\text{wavy bracket}} + \underbrace{\frac{h^2}{2} (f_{t_n} + f_n f_{y_n})}_{\text{wavy bracket}} + \frac{h^3}{4} (f_{tt_n} + 2f_n f_{ty_n} + f_n^2 f_{yy_n}) + \dots$$

Combining:

$$\varepsilon_{local} = y_{n+1|exact} - y_{n+1|Heun} = O(h^3)$$

Only partially cancels

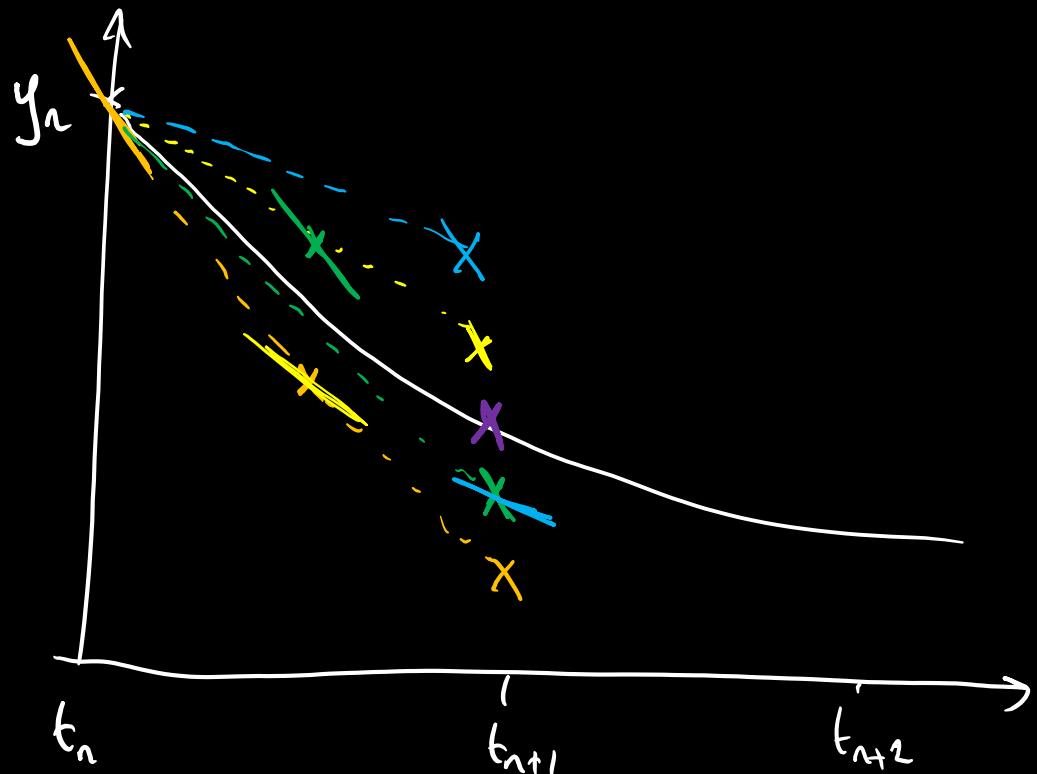
Therefore: $\varepsilon_{global} = O(h^2)$

Improving Accuracy Further: Runge-Kutta Methods

Heun's method is a 2nd order Runge-Kutta method. The most famous Runge-Kutta Method is the 4th order method (RK4):

$$\begin{aligned}k_1 &= h f(t_n, y_n) \\k_2 &= h f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\k_3 &= h f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right) \\k_4 &= h f(t_n + h, y_n + k_3)\end{aligned}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$



RK4: Analogy to Simpson's Rule

The difference between our next value of y and our current value of y is equal to the integral of the gradient:

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} f(t, y(t)) dt$$

$$\frac{dy}{dt} = f$$

Integrate both sides

$$\int_{t_n}^{t_{n+1}} \frac{dy}{dt} dt = \int_{t_n}^{t_{n+1}} f dt$$

*Integration with Left Riemann Sum
with Trapezium Rule Applied to y^* → Fwd Euler*

Applying Simpson's Rule, noting that when we did numerical calculus our h was half the integration interval for Simpson's Rule, whereas here h spans the full time step and hence is double the value of that we used for numerical calculus:

$$y_{n+1} - y_n = \frac{h}{6} \left[f(t_n, y(t_n)) + 4f\left(t_n + \frac{h}{2}, y\left(t_n + \frac{h}{2}\right)\right) + f(t_n + h, y(t_n + h)) \right]$$

$$y_{n+1} = y_n + \frac{1}{6} [hf(t_n, y_n) + 4hf\left(t_{n+1/2}, y_{n+1/2}\right) + hf(t_{n+1}, y_{n+1})]$$

We effectively have the gradient evaluated at both ends of the interval, and 4 times the gradient in the middle. In RK4, k_1 is the slope evaluated at the left edge, k_4 at the right edge, whilst k_2 and k_3 are effectively slopes evaluated at the midpoint. Hence if k_2 and k_3 are averaged:

$$y_{n+1} = y_n + \frac{1}{6} \left(k_1 + \frac{4(k_2 + k_3)}{2} + k_4 \right) = \underbrace{y_n + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)}$$

Accuracy of RK4

Local Error of Simpson's 1/3 Rule:

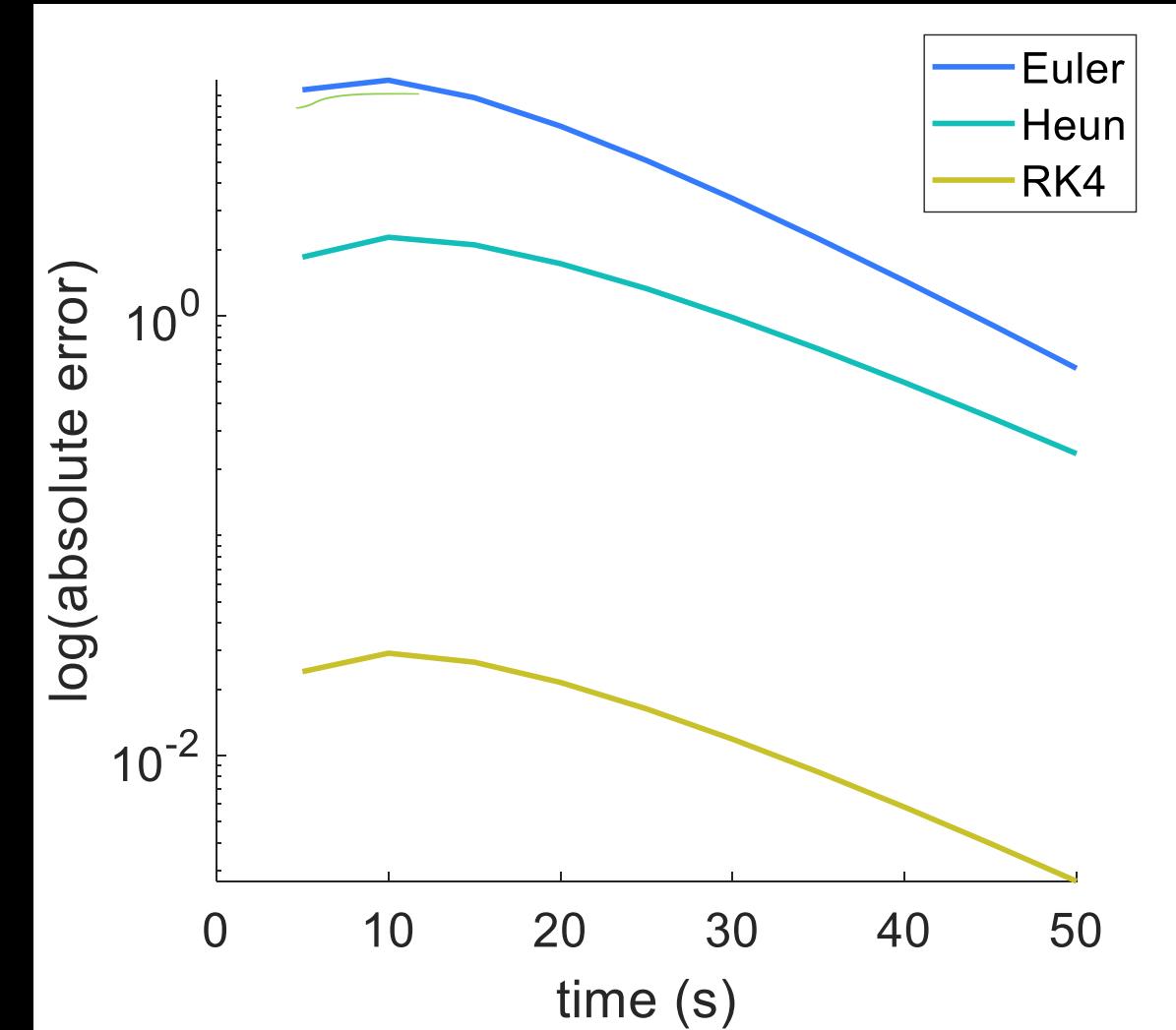
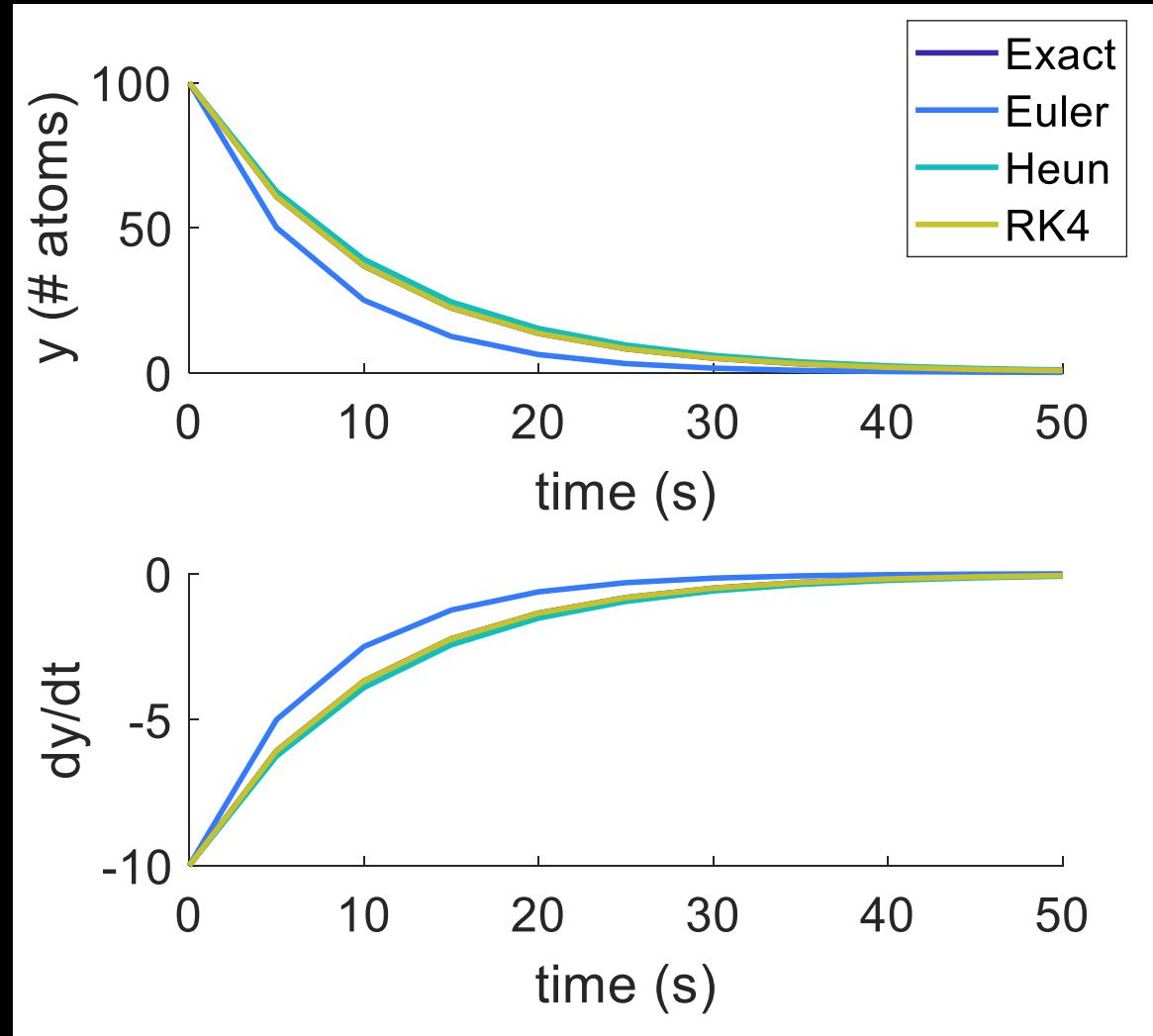
$$\varepsilon_{local} = -\frac{h^5}{180} f^4(\xi) = O(h^5)$$

We are effectively applying Simpson's Rule $N \propto \frac{1}{h}$ times for the global ODE solution. Therefore:

$$\varepsilon_{global} = N \varepsilon_{local} = O(h^4)$$

RK4 is 4th order accurate (hence why it is called the 4th order method).

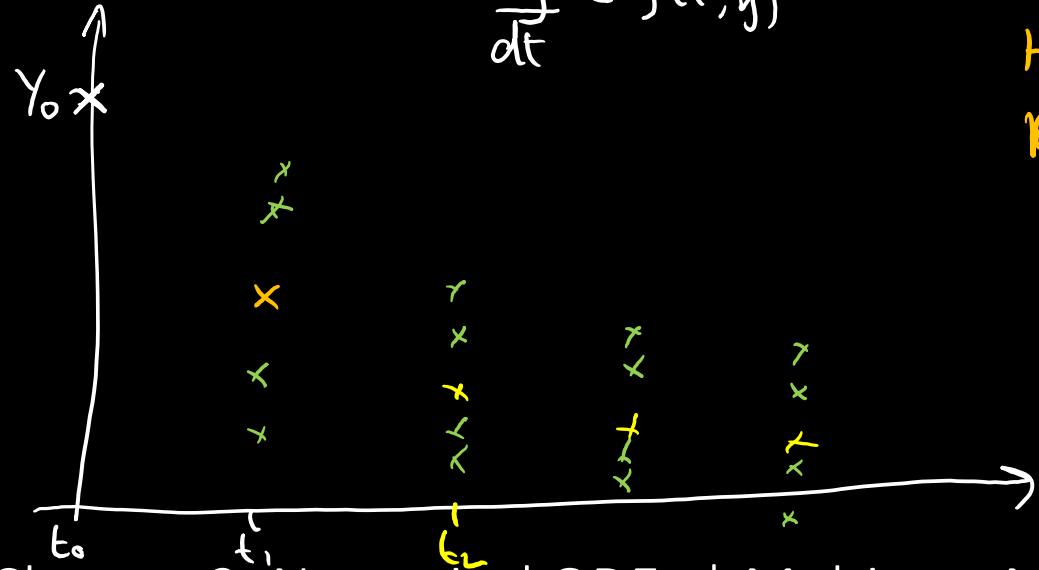
Worked Example 21: Comparing Forward Euler, Heun and RK4



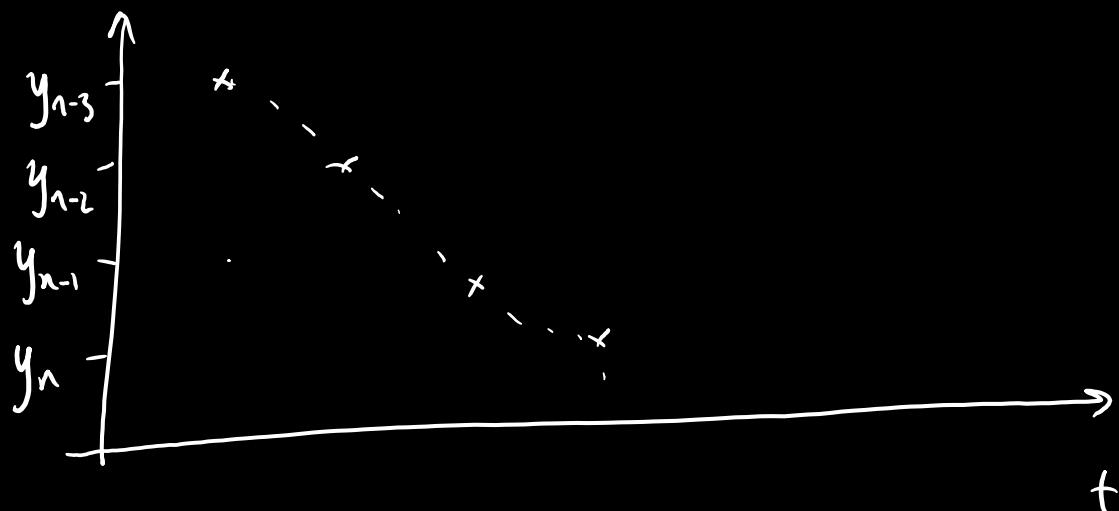
Runge-Kutta-Fehlberg Methods

- An explicit method similar to RK4, however for the RKF method 6 constants are calculated.
- The constants are used to evaluate both a 4th order and a 5th order estimate.
- Their difference gives an error estimate which is used to determine the step size.
- This method is implemented in many maths software packages and is very popular for practical applications as it automatically determines the step size:
 - Python: `scipy.integrate.solve_ivp(f)`
 - MatLab: `ode45`

Method	Function Evaluations	Local Error	Global Error
Forward Euler	1	$O(h^2)$	$O(h)$
Heun	2	$O(h^3)$	$O(h^2)$
RK4	4	$O(h^5)$	$O(h^4)$
RKF	6	$O(h^6)$	$O(h^5)$



Chapter 3: Numerical ODEs | Multistep Methods & Convergence



Multistep Methods

- All of the methods we covered last week used just a single previous value of y .
 - Pro: these methods are self-starting (requiring only a single initial condition).
 - Con: they do not maximise efficiency as the predictor-corrector methodology requires computation of values that are not recorded in the final solution.
- Multistep methods improve accuracy by using previous values of y (e.g. y_{n-1}, y_{n-2} etc) and $f(t, y)$ (e.g. f_{n-1}, f_{n-2} etc). The general form for a linear multistep method is (where α_i and β_i are constants):

$$\sum_{i=0}^s \alpha_i y_{n+1-i} = h \sum_{i=0}^s \beta_i f_{n+1-i}$$

Worked Example 22: The 4th Order Adams Bashford Multistep method

One method to establish a multistep method is to use numerical integration techniques to solve our ODEs. Consider a first order ODE:

$$y' = f(t, y)$$

Integrating with respect to t :

$$\int_{t_n}^{t_{n+1}} y' dt = \int_{t_n}^{t_{n+1}} f(t, y) dt$$

Integration of the LHS is trivial $\int_{t_n}^{t_{n+1}} \frac{dy}{dt} dt = y_{n+1} - y_n$, whereas integration of the RHS can be challenging, particularly for non-linear functions. Therefore, we choose to approximate our $f(t, y)$ with an interpolation polynomial, which is easily integrated. Hence our integrated ODE becomes:

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} p_m(t) dt$$

Hence our next value of y is:

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} p_m(t) dt$$

Worked Example 22: The 4th Order Adams Bashford Multistep method

If we know four values for y , we can implement of 4th order multistep method, i.e. we know:

assume we know
4 existing points

$$\left. \begin{array}{l} t_n, y_n \text{ and hence } f_n = f(t_n, y_n) \\ t_{n-1}, y_{n-1} \text{ and hence } f_{n-1} = f(t_{n-1}, y_{n-1}) \\ t_{n-2}, y_{n-2} \text{ and hence } f_{n-2} = f(t_{n-2}, y_{n-2}) \\ t_{n-3}, y_{n-3} \text{ and hence } f_{n-3} = f(t_{n-3}, y_{n-3}) \end{array} \right\}$$

so we also the gradient
at these 4 point as
we have $\frac{dy}{dt} = f$

We thus have 4 nodes pairs $[t_k, f_k]$ which we can use to find our interpolation polynomial for $f(t, y)$. As we are basing our interpolation of previous values, and we have a constant timestep h , it makes sense to use the equal spacing Newton Backward Divided Difference Interpolation method:

Goal is to find y_{n+1}

$$p_3(t) = \sum_{k=0}^3 (-1)^k \binom{-r}{k} \nabla^k f_n$$

$$\text{Where } r = \frac{t-t_n}{h}$$

$$p_3(t) = \sum_{k=0}^3 (-1)^k \binom{-r}{k} \nabla^k f_n \quad \text{where } r = \frac{t - t_n}{h}$$

Worked Example 22: The 4th Order Adams Bashford Multistep method #1

$$\int_{t_n}^{t_{n+1}} \frac{dy}{dt} dt = \int_{t_n}^{t_{n+1}} f(t, y) dt$$

LHS integrates trivially, the RHS is hard when it is a non-linear function, so we substitute a cubic polynomial

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} p_3(t) dt$$

$$= \int_{t_n}^{t_{n+1}} \left(f_n + r \nabla f_n + \frac{r(r+1)}{2!} \nabla^2 f_n + \underbrace{\frac{r(r+1)(r+2)}{3!} \nabla^3 f_n}_{\text{error term}} \right) dt$$

From #1 $\Rightarrow \frac{dr}{dt} = \frac{1}{h} \Rightarrow dt = h dr$; when $t = t_n, r = 0$; when $t = t_{n+1} = t_n + h, r = 1$

using this substitution

$$y_{n+1} - y_n = \int_0^1 \left(f_n + r \nabla f_n + \frac{(r^2+r)}{2!} \nabla^2 f_n + \underbrace{\frac{(r^3+3r^2+2r)}{3!} \nabla^3 f_n}_{\text{error term}} \right) h dr$$

Integrating:

$$y_{n+1} = y_n + h \left(f_n + \frac{1}{2} \nabla f_n + \frac{5}{12} \nabla^2 f_n + \frac{3}{8} \nabla^3 f_n \right)$$

Worked Example 22: The 4th Order Adams Bashford Multistep method

Our difference table is:

all known

ADV \rightarrow computational efficiency for same order accuracy as RIC4
 DIS ADV \rightarrow not self-starting

Node j	t_j	f_j	∇f_j	$\nabla^2 f_j$	$\nabla^3 f_j$
$n-2$	t_{n-3}	f_{n-3}			
$n-1$	t_{n-2}	f_{n-2}	$f_{n-2} - f_{n-3}$		
n	t_{n-1}	f_{n-1}	$f_{n-1} - f_{n-2}$	$f_{n-1} - 2f_{n-2} + f_{n-3}$	
$n+1$	t_n	f_n	$f_n - f_{n-1}$	$f_n - 2f_{n-1} + f_{n-2}$	$f_n - 3f_{n-1} + 3f_{n-2} - f_{n-3}$

Hence:

$$y_{n+1} = y_n + h \left(f_n + \frac{1}{2}(f_n - f_{n-1}) + \frac{5}{12}(f_n - 2f_{n-1} + f_{n-2}) + \frac{3}{8}(f_n - 3f_{n-1} + 3f_{n-2} - f_{n-3}) \right)$$

$$y_{n+1} = y_n + \frac{h}{24} (55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})$$

This method is 4th order accurate: the local truncation error is $O(h^5)$ and the global error is $O(h^4)$.

Worked example 23: A highly accurate yet useless method

Aim of the WE, derive the most accurate possible two-step explicit method.

- From slide 238: Multistep methods improve accuracy by using previous values of y (e.g. y_{n-1}, y_{n-2} etc) and $f(t, y)$ (e.g. f_{n-1}, f_{n-2} etc). The general form for a linear multistep method is (where α_i and β_i are constants):

$$\sum_{i=0}^s \alpha_i y_{n+1-i} = h \sum_{i=0}^s \beta_i f_{n+1-i}$$

- When $s = 2$, we get the general form of a two-step method:

$$\alpha_0 y_{n+1} + \alpha_1 y_n + \alpha_2 y_{n-1} = h(\beta_0 f_{n+1} + \beta_1 f_n + \beta_2 f_{n-1})$$

- Explicit implies $\beta_0 = 0$:

$$\alpha_0 y_{n+1} + \alpha_1 y_n + \alpha_2 y_{n-1} = h(\beta_1 f_n + \beta_2 f_{n-1})$$

- Noting that all the alphas and betas are arbitrary constants at this stage, we can eliminate the need to find one of the constants if we choose to combine α_0 into the other constants to get the form:

$$y_{n+1} + \alpha_1 y_n + \alpha_2 y_{n-1} = h(\beta_1 f_n + \beta_2 f_{n-1})$$

- Thus, to make my explicit method as accurate as possible we need to find ideal values for $\alpha_1, \alpha_2, \beta_1, \beta_2$

Worked example 23: A highly accurate yet useless method

$$y_{n+1} \approx -\alpha_1 y_n - \alpha_2 y_{n-1} + h(\beta_1 f_n + \beta_2 f_{n-1})$$

Noting that $f_n = y'_n$ (our classic form for a general first order ODE), and making Taylor series expansions for all the $n - 1$ terms to get y_{n+1} as a function of y_n , y'_n , etc:

$$\begin{aligned} y_{n+1} &\approx -\alpha_1 y_n \dots \\ &- \alpha_2 \left(y_n - hy'_n + \frac{h^2}{2} y''_n - \frac{h^3}{6} y'''_n + \frac{h^4}{24} y''''_n + \dots \right) \dots \\ &+ h\beta_1 (y'_n) \dots \\ &+ h\beta_2 \left(y'_n - hy''_n + \frac{h^2}{2} y'''_n - \frac{h^3}{6} y''''_n + \frac{h^4}{24} y'''''_n + \dots \right) \end{aligned}$$

Taylor of y_{n-1}

Taylor of $f_{n-1} = y'_{n-1}$

Compare to exact Taylor expansion of y_{n+1}

$$y_{n+1|exact} = y_n + hy'_n + \frac{h^2}{2} y''_n + \frac{h^3}{6} y'''_n + \frac{h^4}{24} y''''_n + \dots$$

Equating coefficients between exact and approx.:

$$y_n: -\alpha_1 - \alpha_2 = 1$$

$$hy'_n: \alpha_2 + \beta_1 + \beta_2 = 1$$

$$h^2 y''_n: -\frac{1}{2}\alpha_2 - \beta_2 = \frac{1}{2}$$

$$h^3 y'''_n: \frac{1}{6}\alpha_2 + \frac{1}{2}\beta_2 = \frac{1}{6}$$

We have four constants, so we could solve the first four equations simultaneously so that the approx. equals the exact up to the y''''_n term. The resulting method would have local error $O(h^4)$ and so be third order accurate.

Worked example 23: A highly accurate yet useless method

- Simultaneous equations + computer = matrices:

$$\begin{bmatrix} -1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & -\frac{1}{2} & 0 & -1 \\ 0 & \frac{1}{6} & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \frac{1}{2} \\ \frac{1}{6} \end{bmatrix}$$

- Solving:

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & -\frac{1}{2} & 0 & -1 \\ 0 & \frac{1}{6} & 0 & \frac{1}{2} \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \\ \frac{1}{2} \\ \frac{1}{6} \end{bmatrix} = \begin{bmatrix} 4 \\ -5 \\ 4 \\ 2 \end{bmatrix}$$

- And so our new fancy-pants super-duper accurate two step method is:

$$y_{n+1} \approx -4y_n + 5y_{n-1} + h(4f_n + 2f_{n-1}) \quad \leftarrow O(h^4) \text{ local error}$$

Worked example 23: A highly accurate yet useless method

$$y_{n+1} \approx -4y_n + 5y_{n-1} + h(4f_n + 2f_{n-1})$$

“Wow, that method sure is fancy pants”. High fives all round...

- But is it useful?

Convergence

- There are many different numerical methods for estimating solutions of ODEs.
 - Reducing computing time for a given accuracy is a huge driver for advancement
 - When we use a numerical method, we must analyse whether it is *convergent*.
 - I.e. is the numerical estimate representative of the solution and not dominated by errors?
- A method is convergent if the error $\rightarrow 0$, as step size $\rightarrow 0$. I.e.:
$$\max_{n=[0,\frac{\tau}{h}]} |y_{n|exact} - y_{n|method}| \rightarrow 0 \quad \text{as} \quad h \rightarrow 0$$
- Convergence is achieved if the method is *consistent* and *stable*:

Consistency and stability

- The numerical method is **consistent** when the discretised equation tends to the differential equation as $h \rightarrow 0$.
 - A method is consistent if:
$$\lim_{h \rightarrow 0} \left(\frac{\varepsilon_{local}}{h} \right) = 0$$
 - I.e. we are testing whether the local truncation error $\rightarrow 0$ fast enough, as $h \rightarrow 0$, such that the global error could also tend to zero.
- The numerical method is **stable** if the solution remains bounded (errors do not grow). The exact definition of stability depends on context. Two common concepts:
 - Zero stability: the stability of a method in general (i.e. if you derive a new method, you test if it is zero-stable to analyse if it could ever be useful).
 - Absolute stability: the stability of a method when practically applied (i.e. you want analyse the conditions for which a method will be stable when applied to a specific problem).

Is a Newly Derived Method Convergent?

The Dahlquist Equivalence Theorem states that a linear multistep method is convergent if, and only if, it is consistent and zero-stable.

- Thus, once we derive a method, we check if it is **consistent** by verifying that:

$$\lim_{h \rightarrow 0} \left(\frac{\varepsilon_{local}}{h} \right) = 0$$

- Then we will analyse if it is **zero-stable**

- For zero-stability, the solution must remain bounded as $h \rightarrow 0$. Thus, we for a general multistep method we require:

$$\sum_{i=0}^s \alpha_i y_{n+1-i} = 0$$

- This is tested by assuming a solution $y_n = z^n y_0$ and substituting into the above to find a characteristic polynomial $p(z)$. (Note z can be complex).
 - The method is then zero-stable if the roots of $p(z)$ lie within a unit disk, i.e. $|z| \leq 1$,
 - (if $|z| = 1$, it is also necessary to show that there are no repeated roots on the unit circle).

Worked Example 24: Analysing convergence for WE22 and WE23

The Adams-Basford 4th Order method:

$$y_{n+1} - y_n = \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})$$

For consistency we want the local error to vanish fast enough that the global error might tend to zero as well. We test:

$$\lim_{h \rightarrow 0} \left(\frac{\epsilon_{local}}{h} \right) = \lim_{h \rightarrow 0} \left(\frac{O(h^5)}{h} \right) = \lim_{h \rightarrow 0} (O(h^4)) = 0$$

i.e. h on the numerator
✓ yes it is consistent

Does the solution remain bounded as $h \rightarrow 0$? I.e. is it zero-stable?

To test, set $h=0$ & apply test equation $y_n = z^n y_0$

$$h=0 \Rightarrow y_{n+1} - y_n = 0$$

$$y_n = z^n y_0 \Rightarrow z^{n+1} y_0 - z^n y_0 = 0$$

$$y_0 = \text{const} \therefore z^{n+1} - z^n = 0$$

✓ yes
zero-

which has roots $z=0, z=1$
stable
which ≤ 1

Worked Example 24: Analysing convergence for WE22 and WE23

The super accurate yet useless method:

$$y_{n+1} \approx -4y_n + 5y_{n-1} + h(4f_n + 2f_{n-1})$$

- First, check consistency (i.e. does the local error decay sufficiently fast as $h \rightarrow 0$ that there is hope that the global error will be small as well?)

$$\lim_{h \rightarrow 0} \left(\frac{\varepsilon_{local}}{h} \right) = \lim_{h \rightarrow 0} \left(\frac{O(h^4)}{h} \right) = 0$$

- Yep, of course it's consistent as we optimised the accuracy to get the highest order accuracy possible for a two-step explicit method.

Worked Example 24: Analysing convergence for WE22 and WE23

$$y_{n+1} \approx -4y_n + 5y_{n-1} + h(4f_n + 2f_{n-1})$$

- Is it zero stable? Will a solution remain bounded as $h \rightarrow 0$?
- To see this, we test what happens when the step size is zero each time we apply our method.

When $h = 0$, our method is:

$$y_{n+1} = -4y_n + 5y_{n-1}$$

Applying the test equation: $y_n = z^n y_0$

$$z^{n+1} y_0 = -4z^n y_0 + 5z^{n-1} y_0$$

Worked Example 24: Analysing convergence for WE22 and WE23

- Solving to find the zero-step size amplification factor, z :

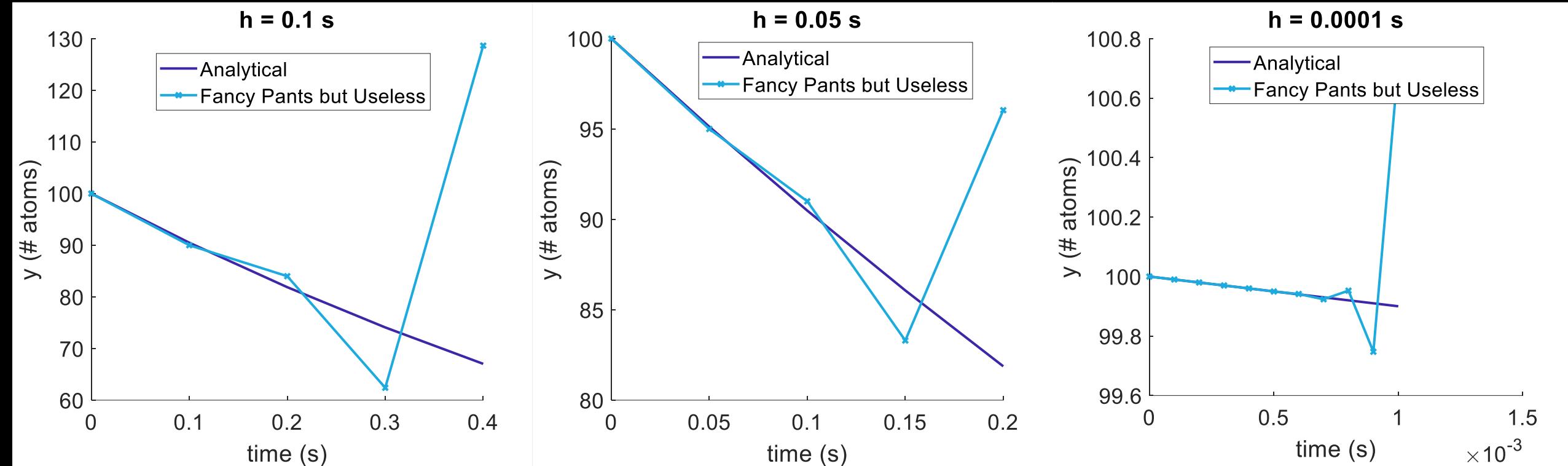
$$\begin{aligned} z^{n+1}y_0 + 4z^n y_0 - 5z^{n-1}y_0 &= 0 \\ z^{n+1} + 4z^n - 5z^{n-1} &= 0 \\ (z^2 + 4z - 5)z^{n-1} &= 0 \\ (z + 5)(z - 1)z^{n-1} &= 0 \end{aligned}$$

- Which has roots at $\underbrace{z = -5}$, $z = 1$ and $z = 0$.
- We need $|z| \leq 1$. But the root $z = -5$ has magnitude bigger than one, meaning it will become unbounded as the number of steps increases. I.e. The method itself is not stable.
- Hence, even though it was derived in a fancy pants way, it's useless.

Worked Example 24: Analysing convergence for WE22 and WE23

For example, if we apply this non-zero stable method to radioactive decay, when $\lambda = -0.1$ and $y_0 = 100$ and the ODE is $f(t, y) = \lambda y$, then the solution with our fancy pants method is rubbish.

It becomes unstable no matter what the step size. As $h \rightarrow 0$ it becomes unstable after shorter times:



Useful when revising: Notational equivalence between amplification factor and zero-stability.

$$y_{n+1} = -4y_n + 5y_{n-1} + h(4f_n + 2f_{n-1})$$

If we wanted to find the amplification factor for this method applied to a problem, we could do the following.

- First, assume that an amplification factor can be found by assuming that $y_{n+1} = A^{n+1}y_0$. If we substitute this in, we get:

$$A^{n+1}y_0 = -4A^n y_0 + 5A^{n-1}y_0 + h(4f_n + 2f_{n-1})$$

- Then, we would substitute in our ODE physics, (e.g. $f_n = \lambda y_n$ for radioactive decay). But for zero stability, we are testing how the method behaves when the step size tends to and becomes zero.
- All our f_n terms disappear when $h = 0$ (and hence the physics of the problem don't matter when we evaluate zero stability, we are just analysing if the numerical method itself is convergent):

$$A^{n+1}y_0 = -4A^n y_0 + 5A^{n-1}y_0$$

- This is the same as our notation on slide 331 except with the letter A instead of z:

$$z^{n+1}y_0 = -4z^n y_0 + 5z^{n-1}y_0$$

- If you have become familiar with the amplification factor, and what it means for stability, this may help you rationalise why we test zero stability by checking if $|z| \leq 1$.

Lecture swap:

Thursday 3pm ME2 Materials <-> Friday 9am ME2 Maths

Chapter 3: Numerical ODEs | Absolute Stability

Worked Example 20: Revisited

Our ODE: $\frac{dy}{dt} = f(t, y) = -0.1y$

Euler method: $y_{n+1} \approx y_n + hf(t_n, y_n)$

$$y_{n+1} = (1 - 0.1h)y_n$$

- What happens as we increase h ?

Absolute Stability: the stability of a method applied to a problem

Consider the process for solving a linear ODE, such as radioactive decay, our next value is based on our previous value multiplied by some amplification factor A , which is a function of λh :

$$y_{n+1} = A(\lambda h)y_n$$

The value for y_n was also calculated in the same way: $y_n = Ay_{n-1}$

Therefore:

$$y_{n+1} = A^2y_{n-1}$$

Continuing this logic we can reason that: $y_{n+1} = A^{n+1}Y_0$

Where Y_0 is our initial condition, and A depends on the numerical method: Euler, Heun, RK4 etc.

Now consider our truncation and rounding errors for our first calculated value:

$$y_1 = AY_0 = y_{exact}(t_1) + \epsilon_1$$

Substituting in:

$$y_2 = Ay_{exact}(t_1) + Ae_1$$
$$y_{n+1} = A^n(y_{exact}(t_1) + \epsilon_1) = A^n y_{exact}(t_1) + A^n \epsilon_1$$

i.e. The local errors evolve in the same way as the solution.

- If $|A| < |1|$ then the magnitude of the errors decreases and the method is stable
- If $|A| > |1|$ then the magnitude of the errors increases and so the method is unstable

Worked Example 25: Forward Euler Absolute Stability for Radioactive Decay

Forward Euler: $y_{n+1} = y_n + h f_n$

For radioactive decay:

$$f(t, y) = \lambda y$$

Forward Euler for radioactive decay: $y_{n+1} = y_n + h(\lambda y_n) = (\underbrace{1 + \lambda h}_{A}) y_n$

For stability we need $|A| < 1$

$$|1 + \lambda h| < 1$$

$$1 + \lambda h < 1$$

$$\lambda h < 0$$

Divide by λ (noting that it is negative)

$$h > 0$$

$$-1 - \lambda h < 1$$

$$-\lambda h < 2$$

$(-1) = +ve$:

$$h < -\frac{2}{\lambda}$$

$$0 < h < -\frac{2}{\lambda}$$

The forward Euler method for radioactive decay is conditionally stable.

E.g. for worked example 20, $\lambda = -0.1$ and therefore stable for $0 < h < 20$

Worked Example 26: Heun Method Absolute Stability for Radioactive Decay

$$f = \lambda y$$

Applying the Heun method to Radioactive Decay:

$$k_1 = h\lambda y_n$$

$$k_2 = h(\lambda(y_n + k_1)) = \lambda h(y_n + h\lambda y_n)$$

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{2}(h\lambda y_n + \lambda h(y_n + h\lambda y_n)) \\ &= (\frac{1}{2}(\lambda h)^2 + \lambda h + 1)y_n \end{aligned}$$

A

For stability we need $|A| < 1$

$$|\frac{1}{2}(\lambda h)^2 + \lambda h + 1| < 1$$

$$\begin{aligned} \frac{1}{2}(\lambda h)^2 + \lambda h + 1 &< 1 \\ (\lambda h)^2 + 2\lambda h &< 0 \end{aligned}$$

$$\begin{aligned} \lambda h &< 0 & (\lambda h)^2 &< -2\lambda h \\ h > 0 & \text{Divide by } (\lambda h) & \lambda h &< -2 \\ & \text{is } -\infty & h &< -2/\lambda \end{aligned}$$

$$\begin{aligned} 0 < h &< -2/\lambda \end{aligned}$$

Heun Method:

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf(t_n + h, y_n + k_1)$$

$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$

$$\frac{1}{2}(\lambda h)^2 + \lambda h + 1 > -1$$

$$(\lambda h)^2 + 2\lambda h + 4 > 0$$

Complete the square

$$(\lambda h + 1)^2 + 3 > 0$$

$$(\lambda h + 1)^2 > -3$$

$$(\lambda h + 1) \leq \pm \sqrt{3}$$

$$\lambda h \leq -1 \pm \sqrt{3}$$

=PENDING DOOM

Worked Example 27: RK4 Absolute Stability for Radioactive Decay

Following the same steps for RK4: $y_{n+1} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + y_n =$

$$\begin{aligned} & \frac{1}{6} \left(\cancel{\lambda h y_n} + 2 \left(\lambda h \left(y_n + \frac{1}{2} \cancel{\lambda h y_n} \right) \right) + 2 \left(\lambda h \left(y_n + \frac{1}{2} \lambda h \left(y_n + \frac{1}{2} \cancel{\lambda h y_n} \right) \right) \right) \right. \\ & \quad \left. + \lambda h \left(y_n + \lambda h \left(y_n + \lambda h \left(y_n + \frac{1}{2} \lambda h \left(y_n + \frac{1}{2} \lambda h y_n \right) \right) \right) \right) \right) + y_n \end{aligned}$$

Simplifying: ROOT FINDING

$$y_{n+1} = \left(\frac{1}{24} \lambda^4 h^4 + \frac{1}{6} \lambda^3 h^3 + \frac{1}{2} \lambda^2 h^2 + \lambda h + 1 \right) y_n$$

Stable when: (will cover with
Matrix next year)

$$\left| \frac{1}{24} \lambda^4 h^4 + \frac{1}{6} \lambda^3 h^3 + \frac{1}{2} \lambda^2 h^2 + \lambda h + 1 \right| < 1$$

$$-2.7852 < \lambda h < 0 \text{ and so } 0 < h < -\frac{2.7852}{\lambda}$$

$$\text{With, } \lambda = -0.1 \Rightarrow 0 < h < 27.852$$

Implicit Methods for Solving ODEs

So far our ODE solvers have been explicit

- Forward Euler, Heun, RK4, AB4
- The next value is calculate purely from known values
- The methods are unstable/conditionally stable

We can also solve ODEs with implicit methods

- These methods are harder to implement as we have to evaluate our gradient function with values that have yet to be calculated
- Hence, they require solving of an equation (which may be non-linear)
- This extra effort however typically ensures we have unconditionally stability

An Implicit Method: the Backward Euler method

Consider backwards numerical differentiation:

$$y'_n \approx \frac{y_n - y_{n-1}}{h}$$

Rearranging and shifting by 1 index:

$$y_{n+1} \approx y_n + hy'_{n+1}$$

From our first order ODE, we know:

$$y'_{n+1} = f(t_{n+1}, y_{n+1})$$

Therefore:

$$y_{n+1} \approx y_n + hf(t_{n+1}, y_{n+1})$$

We are evaluating our gradient/function at the future value and therefore need to rearrange and solve our equation for y_{n+1} . This step is typically trivial for linear equations, but can be challenging for non-linear equation or large systems of linear equations.

You will demonstrate the accuracy is $O(h)$ in your tutorial

Worked Example 28: Backward Euler for Radioactive Decay

Develop forward and backward Euler method numerical schemes for calculating the number of atoms as a radioactive element decays. For what timesteps are these methods stable?

Radioactive Decay ODE: $\frac{dy}{dt} = f(t, y) = \lambda y$ where by definition λ is a negative constant

Forward

$$y_{n+1} \approx y_n + hf(t_n, y_n) = y_n + h(\lambda y_n)$$

$$y_{n+1} = (1 + \lambda h)y_n$$

Stable when: $|1 + \lambda h| < |1|$

$$-1 < 1 + \lambda h < 1$$

Conditionally stable: $0 < h < -\frac{2}{\lambda}$

Backward

$$y_{n+1} = y_n + h f_{n+1}$$

$$= y_n + h (\lambda y_{n+1})$$

Solve for y_{n+1} :

$$y_{n+1} - \lambda h y_{n+1} = y_n$$

$$y_{n+1} (1 - \lambda h) = y_n$$

$$y_{n+1} = \frac{1}{(1 - \lambda h)} y_n$$

$$|A| < 1 \Rightarrow \left| \frac{1}{1 - \lambda h} \right| < 1$$

$$|1| < |1 - \lambda h|$$

$$1 < 1 - \lambda h$$

$$1 - \lambda h > 1$$

$$-\lambda h > 0$$

$$h > 0$$

unconditionally
stable

$$-1 > 1 - \lambda h$$

$$1 - \lambda h < -1$$

$$-\lambda h < -2$$

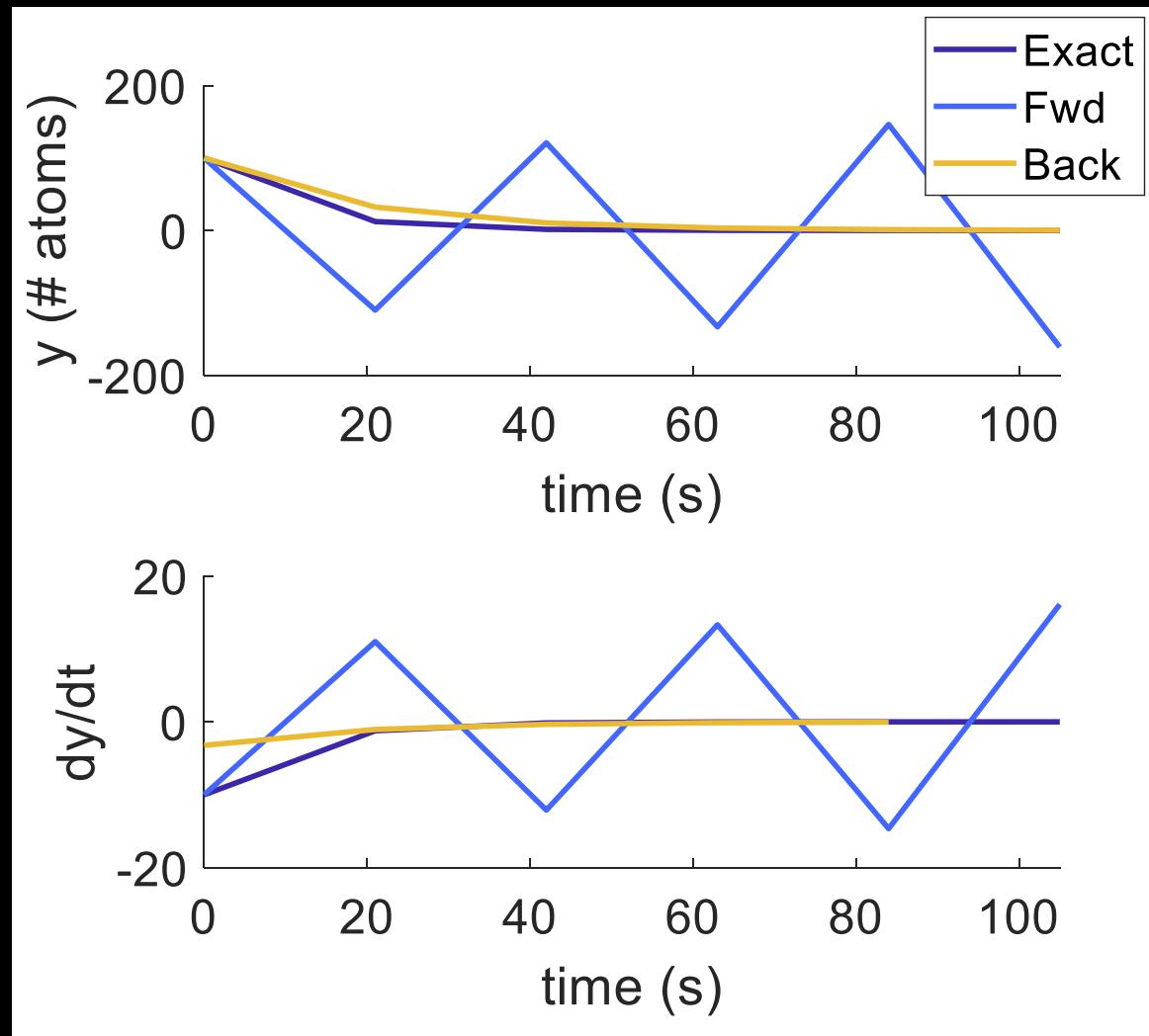
$$h < \frac{2}{\lambda}$$

nonsense

Worked Example 28: Backward Euler for Radioactive Decay

With $h = 21$:

- Forward Euler Unstable
- Backward Euler Stable



Implicit Multistep Methods: Adams-Moulton

- Our Adams-Bashford method is explicit: it is based purely on previous known values of y .
- However, before we can apply our multistep method, we need to get started
 - For our 4th order method, we need the first 4 values.
- To do so, we use another explicit method, such as RK4 or Heun or a lower order Adams method.
- It is also possible to derive implicit multistep methods.
 - These are known as Adams-Moulton methods.
 - To derive these, we use the same methodology, but rather apply a Newton Equal Spacing Backward Divided Difference interpolation from the $n + 1$ value

Implicit Multistep Methods: Backward Differentiation Formulae

BDF methods follow a different philosophy to Adams methods: instead of using our polynomial to approximate the integral of our gradient function, we use it to approximate the derivative. We previously derived a higher order backwards derivative approximation:

$$y'_{n+2} = \frac{1}{2h} (y_n - 4y_{n+1} + 3y_{n+2})$$

Shifted one index:

$$y'_{n+1} = \frac{1}{2h} (y_{n-1} - 4y_n + 3y_{n+1}) \quad \#1$$

Our ODE:

$$y' = f(t, y)$$

Evaluated at the $n + 1$ node:

$$y'_{n+1} = f(t_{n+1}, y_{n+1}) \quad \#2$$

Combining #1 and #2 and rearranging we get the formula for BDF2:

$$y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1} + \frac{2h}{3}f(t_{n+1}, y_{n+1})$$

Overview the Multistep Methods

Adams-Basford (AB)

- Explicit (easier to solve)
- $f(t, y)$ evaluated at $t_n, t_{n-1}, t_{n-2} \dots$ (as required, depending on the order) to calculate y_{n+1}

Adams-Moulton (AM)

- Implicit (improved absolute stability)
- $f(t, y)$ evaluated at $t_{n+1}, t_n, t_{n-1} \dots$ and equation is solved to find y_{n+1} , or...
- Often combined with AB as a predictor-corrector method: AB used to predict y_{n+1} , and then this predicted value is used to evaluate f_{n+1} in an AM method. Combining advantages of AB and AM.

Backward Differentiation Formulae (BDF)

- Implicit (more stable)
- $f(t, y)$ evaluated at t_{n+1} only, combined with previous values y and solved to find y_{n+1}
- Only BDF1-6 are zero-stable

Chapter 3: Numerical ODEs | Systems of ODEs and Stiffness

Systems of ODEs

- All of our numerical methods for solving ODEs can also be applied to a system of coupled ODEs by applying the solver to each variable.
- For linear systems of ODEs – this can be conveniently captured through vectors and matrices (which can be readily programmed into a computer).

$a = \text{rabbits}$; $b = \text{foxes}$

$$\frac{da}{dt} = f(a, b, t)$$

$$\frac{db}{dt} = g(a, b, t)$$

Worked Example 29: Stiff Systems of ODEs

Solve the following system of coupled ODEs with forward and backward Euler methods where k is a constant. Start with $k = 10$ and $h = 0.1$, and explore what happens as k and h are varied.

$$\frac{dy_1}{dt} = \underline{-y_1 - y_2} = f(t, y_1, y_2) \leftarrow \text{row 1}$$

$$\frac{dy_2}{dt} = \underline{y_1 - ky_2} = g(t, y_1, y_2) \leftarrow \text{row 2}$$

Forward Euler:

$$\begin{aligned} y_{1,n+1} &= y_{1,n} + h f_n \\ &= y_{1,n} + h (-y_{1,n} - y_{2,n}) \end{aligned}$$

$$\begin{aligned} y_{2,n+1} &= y_{2,n} + h g_n \\ &= y_{2,n} + h (y_{1,n} - k y_{2,n}) \end{aligned}$$

→ no longer radioactive Decay
It is just made

ODE: $y' = M y$

Fwd Euler $y_{n+1} = y_n + h f_n$

$$= y_n + h M y_n$$

$$= (\underline{I} + h M) y_n \rightarrow A$$

$$\begin{aligned} \frac{d}{dt} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} &= \begin{pmatrix} -1 & -1 \\ 1 & -k \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \\ \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}_{n+1} &= \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}_n + h \begin{pmatrix} -1 & -1 \\ 1 & -k \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}_n \\ &= \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + h \begin{pmatrix} -1 & -1 \\ 1 & -k \end{pmatrix} \right] \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}_n \\ &= \begin{pmatrix} 1-h & -h \\ h & 1-keh \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}_n \end{aligned}$$

Worked Example 29: Stiff Systems of ODEs

$$\text{ODE: } \dot{\underline{y}} = \underline{f} = M \underline{y}$$

Backward Euler:

$$\begin{aligned}\underline{y}_{n+1} &= \underline{y}_n + h \underline{f}_{n+1} \\ &= \underline{y}_n + h M \underline{y}_{n+1}\end{aligned}$$

Solve for \underline{y}_{n+1} :

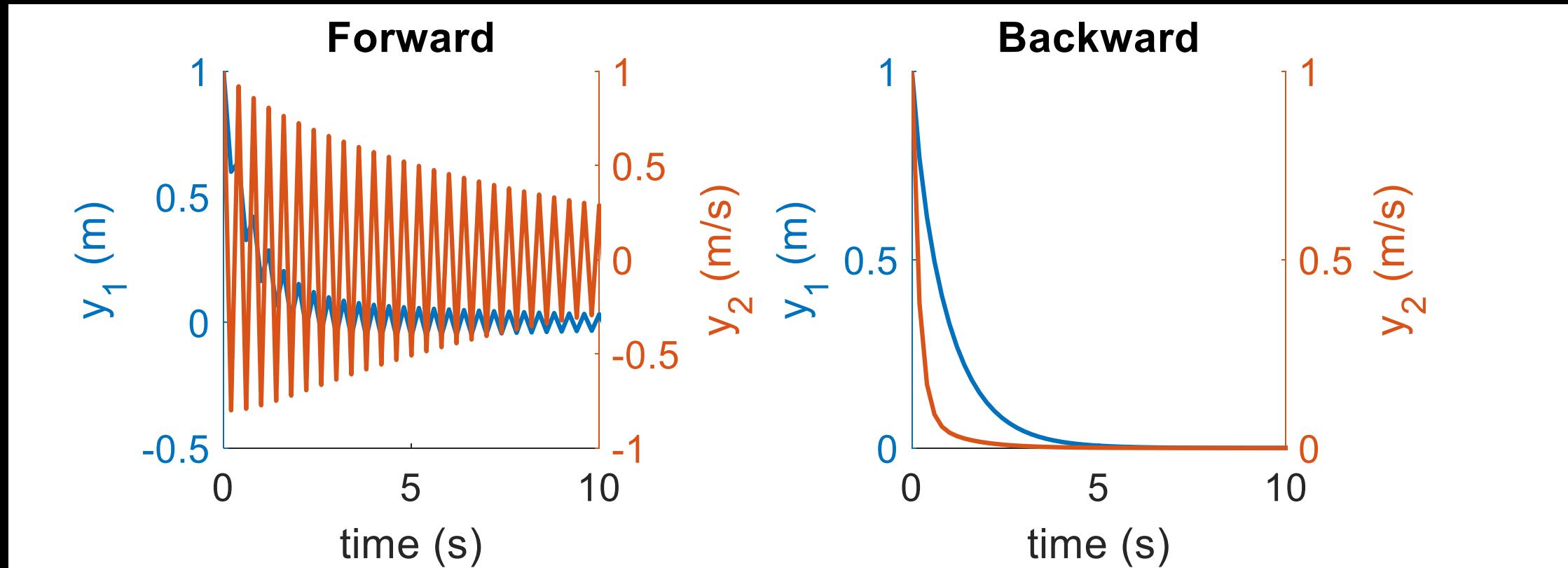
$$\underline{y}_{n+1} - h M \underline{y}_{n+1} = \underline{y}_n$$

$$(\underline{I} - h M) \underline{y}_{n+1} = \underline{y}_n$$

Pre-multiply by the inverse of the matrix $(\underline{I} - h M)$:

$$\underline{y}_{n+1} = \underbrace{(\underline{I} - h M)^{-1}}_A \underline{y}_n$$

Worked Example 29: Stiff Systems of ODEs



Worked Example 29: Stiff Systems of ODEs

With large k , the system is described as stiff.

Explicit methods:

- The rapid rate of change of one parameter requires a small step size whilst the slow rate of change of the other means that the system must also be solved over a long time period.
- This is very computationally expensive.

Implicit methods:

- There are no restrictions on stability and hence they can tolerate a large step size.
- However, linear systems require matrix inversion (and the computational expense of this increases rapidly for increasing numbers of equations) and non-linear systems require solution of non-linear equations.
- The benefit gained from increased step size needs to be weighed up against the disadvantage of the need to solve an equation.
- Accuracy is still affected by the large step size with errors of $O(h)$

Stability of Linear Systems of ODEs

Our stability analysis for a single ODE compared $|A|$ to 1

Linear systems of ODEs can be written in a matrix format such that $\mathbf{y}_{n+1} = \mathbf{A}\mathbf{y}_n$ where, as before, the matrix \mathbf{A} depends on the ODE and the numerical method used. For example in WE29:

- Forward Euler: $\mathbf{A} = (\mathbf{I} + h\mathbf{M})$
- Backward Euler: $\mathbf{A} = (\mathbf{I} - h\mathbf{M})^{-1}$

However, the \mathbf{M} is coupled* and so we cannot yet apply our stability analysis. We need to find a way to decouple matrix \mathbf{M}

*coupled ODEs means that the rate of change of y_1 depends on both y_1 and y_2 , etc.

Eigenvalue Stability for Numerical Solutions to ODEs

Recall from first year (section 18.6) and second year vector calculus that we can diagonalise a matrix with its eigenvectors, \mathbf{v}_i , and eigenvalues, λ_i :

Where: $\Lambda = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \ddots \end{pmatrix}$ and $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \dots) = \begin{pmatrix} v_{11} & v_{21} & \dots \\ v_{12} & v_{22} & \vdots \\ \vdots & \vdots & \ddots \end{pmatrix}$ with $v_{i,1}$ the first component of eigenvector \mathbf{v}_i , $v_{i,2}$ the second component of eigenvector \mathbf{v}_i and so on.

Therefore we can re-write our ODE $\mathbf{y}' = \mathbf{M}\mathbf{y}$ as:

Pre-multiplying by \mathbf{V}^{-1} :

Defining a new variable $\mathbf{w} = \mathbf{V}^{-1}\mathbf{y}$:

Our forward Euler becomes:

$$\mathbf{y}' = \mathbf{V}\Lambda\mathbf{V}^{-1}\mathbf{y}$$

$$\mathbf{V}^{-1}\mathbf{y}' = \Lambda\mathbf{V}^{-1}\mathbf{y}$$

$$\mathbf{w}' = \Lambda\mathbf{w}$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n + h\Lambda\mathbf{w}_n$$

$$\mathbf{w}_{n+1} = (\mathbf{I} + h\Lambda)\mathbf{w}_n$$

We have successfully decoupled our system of ODEs as both \mathbf{I} and Λ are diagonal.

Eigenvalue Stability for Numerical Solutions to ODEs

For example, if we had a system of 3 ODEs then:

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}_{n+1} = \left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + h \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \right) \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}_n = \begin{pmatrix} 1 + \lambda_1 h & 0 & 0 \\ 0 & 1 + \lambda_2 h & 0 \\ 0 & 0 & 1 + \lambda_3 h \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}_n$$

And so:

$$w_{i,n+1} = (1 + \lambda_i h)w_i$$

Our w_i are decoupled, and so our stability analysis can be applied to each w_i in turn.

- For the full system of ODEs to be stable, each w_i must be stable as $\mathbf{y} = \mathbf{V}\mathbf{w}$
 - In words: \mathbf{y} is a linear function of \mathbf{w} . Therefore, if one element of \mathbf{w} ‘blows up’, then \mathbf{y} must also ‘blow up’, but if \mathbf{w} is stable then \mathbf{y} will also be stable.
- Therefore for stability each $|1 + \lambda_i h| < 1$
- We need to analyse the stability of our numerical method for each eigenvalue λ_i .

Worked Example 30: Stability of WE29

What is the largest h that guarantees stability of our forward and backward Euler numerical methods for solving the following system of ODEs:

$$y'_1 = -y_1 - y_2 \quad \text{and} \quad \underline{\underline{M}} \quad y'_2 = y_1 - ky_2$$

With $k = 10$:

step 1) we find the eigenvalues

$$\underline{\underline{y'}} = \begin{pmatrix} -1 & -1 \\ 1 & -10 \end{pmatrix} \underline{\underline{y}}$$

step 2) we analyse stability for each ω (by analysing the eigenvalues)

$$|\underline{\underline{M}} - \lambda \underline{\underline{I}}| = 0$$

$$\begin{vmatrix} -1-\lambda & -1 \\ 1 & -10-\lambda \end{vmatrix} = (-1-\lambda)(-10-\lambda) - (-1 \times 1) = \lambda^2 + 11\lambda + 11 = 0$$

quadratic formula:

$$\lambda = -\frac{11}{2} \pm \frac{1}{2} \sqrt{11^2 - 44}$$

$$= -\frac{11}{2} \pm \frac{1}{2} \sqrt{77}$$

$$= -1.113, -9.887$$

Worked Example 30: Stability of WE29

Stability analyses $|A| < 1$:

slide 259

Forward Euler $|1 + \lambda h| < 1$

Eigenvalue $\lambda = -1.113$

$$0 < h < -\frac{2}{\lambda}$$

$$0 < h < -\frac{2}{-1.113}$$

$$0 < h < 1.7977$$

stricter

so overall my system is stable

when $0 < h < 0.2023$

Backward Euler $|1 - \lambda h| > 1$

Eigenvalue $\lambda = -1.113$

$$|1 + 1.113h| > 1$$

✓ true for
all $h > 0$

slide 264

Eigenvalue $\lambda = -9.887$

$$|1 + 9.887h| > 1$$

✓ true for all
 $h > 0$

unconditionally stable

Spectral Condition Number: Implicit versus Explicit Methods Trade-off

Both Explicit and Implicit method have pros and cons:

- Explicit methods:
 - Pro: Easy to implement
 - Con: Conditional stability/unstable
- Implicit methods
 - Pro: Absolute stability for large h
 - Con: Requires solution of an equation each time step (which can be non-linear).

The Spectral Condition Number, S , is a way for us to determine whether we should decrease the step size and use an explicit method, or to solve the equation and use an implicit method:

$$S = \frac{\max|\lambda_i|}{\min|\lambda_i|}$$

As a rule of thumb, when $S > 1000$ problems are considered stiff, and implicit methods tend to be more computationally efficient than explicit methods.

Solution of Stiff ODEs in Matlab/Python

- Many software packages have an inbuilt implicit solver for stiff systems of ODEs.
- This solver is based on the Backward Differentiation Formulae (or a variable step size equivalent).
 - MatLab: `ode15s`
 - Python: `scipy.integrate.solve_ivp(f, method='BDF')`
- As a rule of thumb, when $S > 1000$
 - Use `ode15s` or `scipy.integrate.solve_ivp(f, method='BDF')`
- When $S < 1000$
 - Use `ode45` or `scipy.integrate.solve_ivp(f)`
- Or be a lazy engineer: try `ode45/scipy.integrate.solve_ivp(f)` and if it doesn't work try `ode15s/scipy.integrate.solve_ivp(f, method='BDF')`. After all, modern computers are very fast...

Chapter 3: Numerical ODEs | Higher Order & Oscillating ODEs

Higher Order ODEs: System of first order ODEs

For example the equation of motion for a mass spring damper:

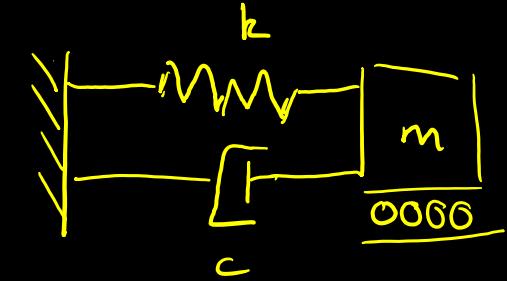
$$m \frac{d^2y}{dt^2} + c \frac{dy}{dt} + ky = 0 \quad \#1$$

We can rewrite this as a system of first order ODEs by defining:

$$\begin{aligned} y_1 &= y &= \text{position} \\ y_2 &= \frac{dy}{dt} = \frac{dy_1}{dt} &= \text{velocity} \end{aligned} \quad \#2$$

Therefore our equation of motion (#1) becomes:

$$m \frac{dy_2}{dt} + cy_2 + ky_1 = 0 \quad \#4$$



Our second order ODE has become a system of first order ODEs:

From #3: $\frac{dy_1}{dt} = y_2$

From #4: $\frac{dy_2}{dt} = -\frac{c}{m}y_2 - \frac{k}{m}y_1$

Note that: the highest index of y needed is equal to the order of the equation. Therefore if we had a third order term, we would also need a $y_3 = \frac{d^2y}{dt^2} = \frac{dy_2}{dt}$

Higher Order ODEs: System of first order ODEs

Rearranging #3 and #4:

$$\begin{aligned}\frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= -\frac{c}{m}y_2 - \frac{k}{m}y_1\end{aligned}$$

The two ODEs needed to be solved simultaneously as they are coupled. Linear simultaneous → matrices:

$$\frac{d}{dt} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \quad \begin{matrix} \textcolor{green}{y'} = f \\ f = M y \end{matrix}$$

This first order ODE is of the form $\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) = \mathbf{M}\mathbf{y}$. Therefore it can be solved with the our first order ODE numerical methods, e.g. the forward Euler method becomes:

$$\mathbf{y}_{n+1} \approx \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n) = (\mathbf{I} + h\mathbf{M})\mathbf{y}_n$$

Heun and RK methods also can be applied to vectors and matrices.

$$\frac{d}{dt} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

Worked Example 31: Numerical Solution Mass-Spring system

Consider a 1 kg mass on frictionless rollers attached to a horizontal spring with stiffness 10 N/m. The mass is pulled until the spring has been extended 0.2 m and then it is released. Numerically estimate the position of the mass after 6 s using forward Euler method using a time step of 0.01 s. What happens if a Heun, RK4 or backward Euler method is used instead?

$$\dot{\mathbf{y}}' = \begin{pmatrix} 0 & 1 \\ -10 & 0 \end{pmatrix} \mathbf{y}$$

$$\mathbf{y}_0 = \begin{pmatrix} 0.2 \\ 0 \end{pmatrix}$$

Fwd Euler:

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h \mathbf{f}_n \\ &= \mathbf{y}_n + h \begin{pmatrix} 0 & 1 \\ -10 & 0 \end{pmatrix} \mathbf{y}_n \end{aligned}$$

$$h = 0.01 \text{ seconds}$$

Worked Example 31: Numerical Solution Mass-Spring system

$$\begin{aligned} \text{bold } \downarrow & \quad y_0 = \begin{pmatrix} 0.2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & hM \\ -0.1 & 0 \end{pmatrix} \begin{pmatrix} 0.2 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -0.1 \times 0.2 + 0.01 \times 0 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -0.02 \end{pmatrix} = \begin{pmatrix} 0.2 \\ -0.02 \end{pmatrix} \\ y_1 = y_2 & = \begin{pmatrix} 0.2 \\ -0.02 \end{pmatrix} + \begin{pmatrix} 0 & 0.01 \\ -0.1 & 0 \end{pmatrix} \begin{pmatrix} 0.2 \\ -0.02 \end{pmatrix} = \begin{pmatrix} 0.2 \\ -0.02 \end{pmatrix} + \begin{pmatrix} -0.0002 \\ -0.02 \end{pmatrix} = \begin{pmatrix} 0.1998 \\ -0.04 \end{pmatrix} \\ y_3 = y_3 & = \begin{pmatrix} 0.1998 \\ -0.04 \end{pmatrix} + \begin{pmatrix} 0 & 0.01 \\ -0.1 & 0 \end{pmatrix} \begin{pmatrix} 0.1998 \\ -0.04 \end{pmatrix} = \begin{pmatrix} 0.1998 \\ -0.04 \end{pmatrix} + \begin{pmatrix} -0.0004 \\ -0.01998 \end{pmatrix} = \begin{pmatrix} 0.1994 \\ -0.05998 \end{pmatrix} \end{aligned}$$

y
 \downarrow
 new
 position
 & velocity
 after
 0.01 s

And so on. Careful with notation:

vector y_3 at time t_3

$$y_3 = y_3 = \begin{pmatrix} y_{1,3} \\ y_{2,3} \end{pmatrix}$$

$y_1(t_3)$ = position at time 3
 $y_2(t_3)$ = velocity at time 3

Some people prefer to write

$$y^n = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}^n = \begin{pmatrix} y_1^n \\ y_2^n \end{pmatrix}$$

Rest of problem demonstrated in lecture.

Worked Example 31: Stability Analysis

$$\underline{M} = \begin{pmatrix} 0 & 1 \\ -10 & 0 \end{pmatrix}$$

Stability → need to find eigenvalues

$$|\underline{M} - \lambda \underline{I}| = 0$$

$$\begin{vmatrix} -\lambda & 1 \\ -10 & -\lambda \end{vmatrix} = 0 = (-\lambda)(-\lambda) - (-10)(1)$$
$$= \lambda^2 + 10$$

$$\Rightarrow \lambda^2 = -10$$

$$\lambda = \pm \sqrt{10}i$$

Analogy to SHM

$$y'' = -\omega^2 y$$

$$\text{Try } y = e^{\lambda t}$$

$$\text{Find } \lambda = \pm \omega i$$

$$\Rightarrow y = e^{\pm i\omega t}$$

$$e^{i\omega t} = \cos \omega t + i \sin \omega t$$

$$= c_1 \cos \omega t + c_2 \sin \omega t$$

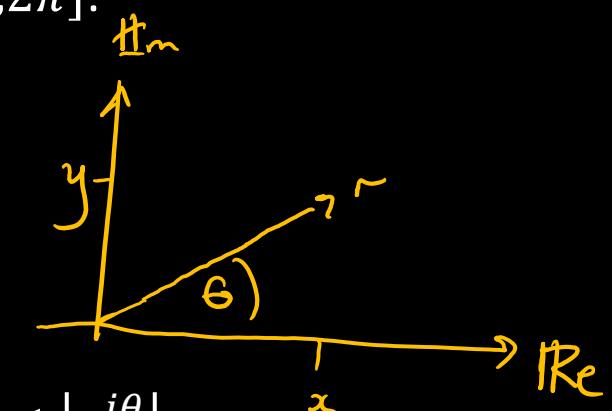
Complex Stability

- For all our examples previously we have only considered the real values of 1.
- But for oscillatory systems, we encounter complex numbers – (indeed this is part of our solution for second order ODEs, and will be part of our solution set for PDEs too).
- If we consider include complex numbers in our analysis:

$$1 = |e^{i\theta}| = |\cos \theta + i \sin \theta| \quad \text{where } \theta = [0, 2\pi].$$

$$z = x + iy$$

$$|z| = |x + iy| = \sqrt{x^2 + y^2}$$



Therefore, our stability analysis for our amplification factor becomes $|A| < |e^{i\theta}|$

Complex Stability for the Forward Euler Method

$$\text{For } wE3I : \quad \lambda = \pm\sqrt{10}i \\ \lambda h = \pm\sqrt{10}ih$$

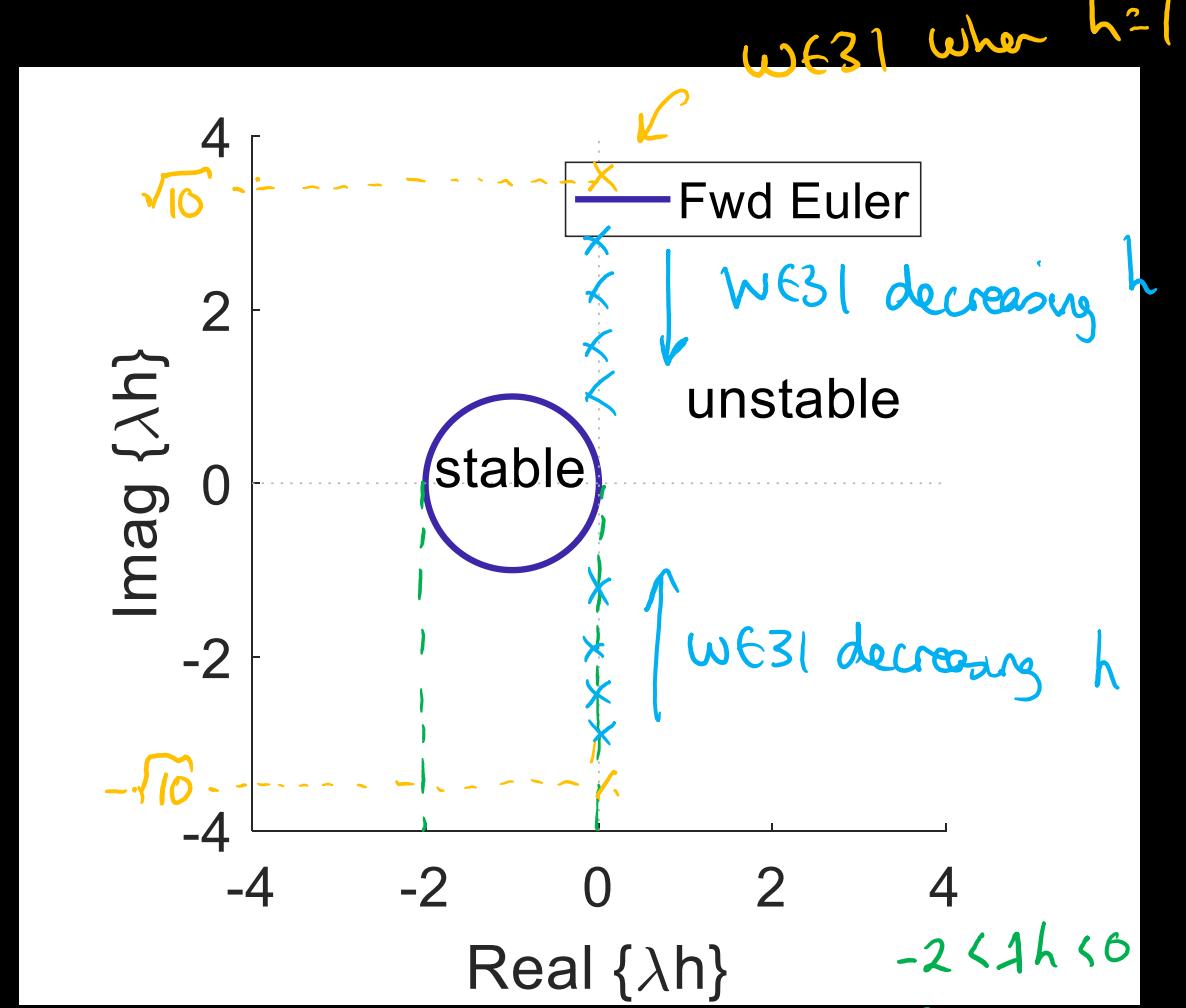
For the forward Euler $A = (1 + \lambda h)$, therefore:

$$|1 + \lambda h| < |e^{i\theta}| \text{ for } \theta = [0, 2\pi]$$

And hence our stability is defined by:

$$\lambda h < e^{i\theta} - 1$$

which is a circle of radius 1, centred at -1 on the real axis.



Stability Regions in the Complex Plane: Explicit Methods

Stability boundaries:

$$\text{Forward Euler: } 1 + \lambda h = e^{i\theta}$$

$$\text{Heun: } 1 + \lambda h + \frac{1}{2}\lambda^2 h^2 = e^{i\theta}$$

$$\text{RK4: } 1 + \lambda h + \frac{1}{2}\lambda^2 h^2 + \frac{1}{6}\lambda^3 h^3 + \frac{1}{24}\lambda^4 h^4 = e^{i\theta}$$

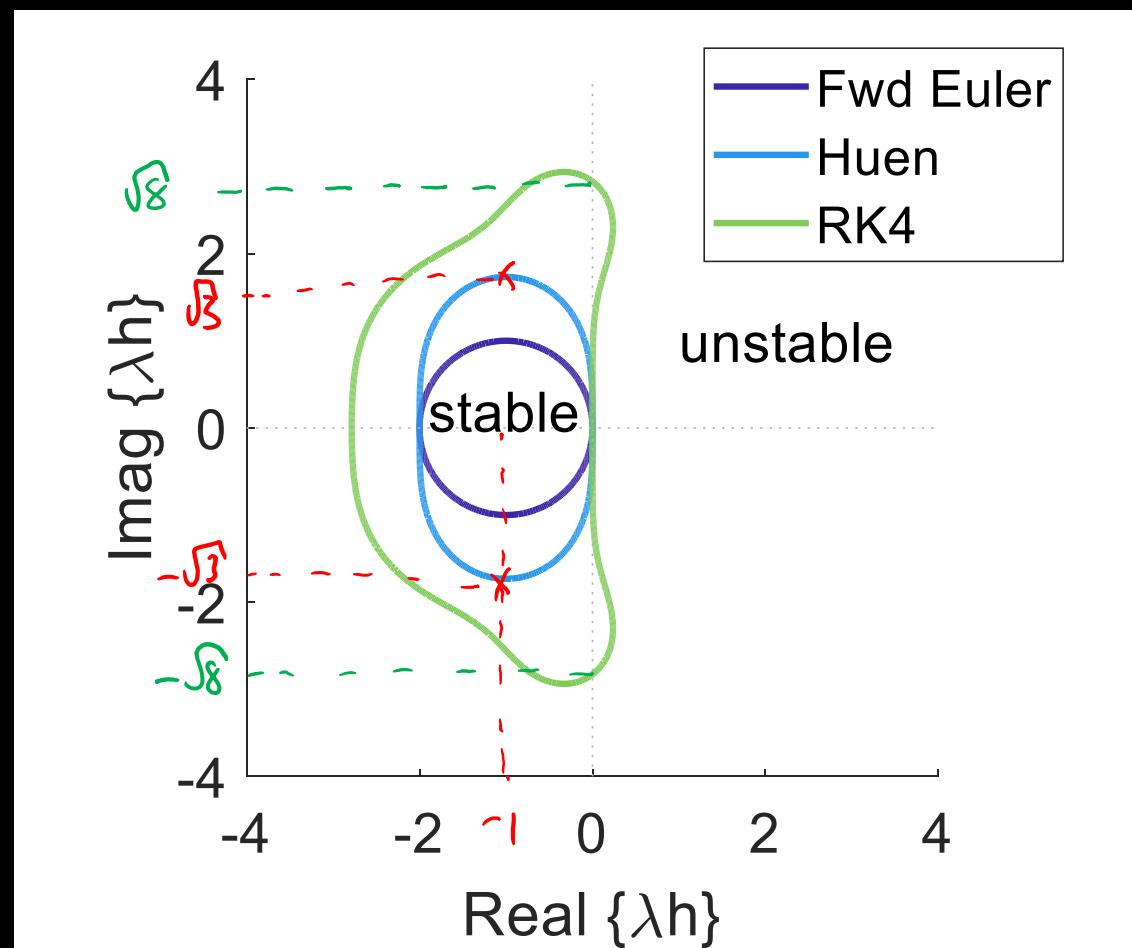
For ω_631

$$\lambda = \sqrt{10}i$$

$$\lambda h = \sqrt{10}ih$$

$$\text{RK4 stable } \sqrt{8}i > \sqrt{10}ih$$

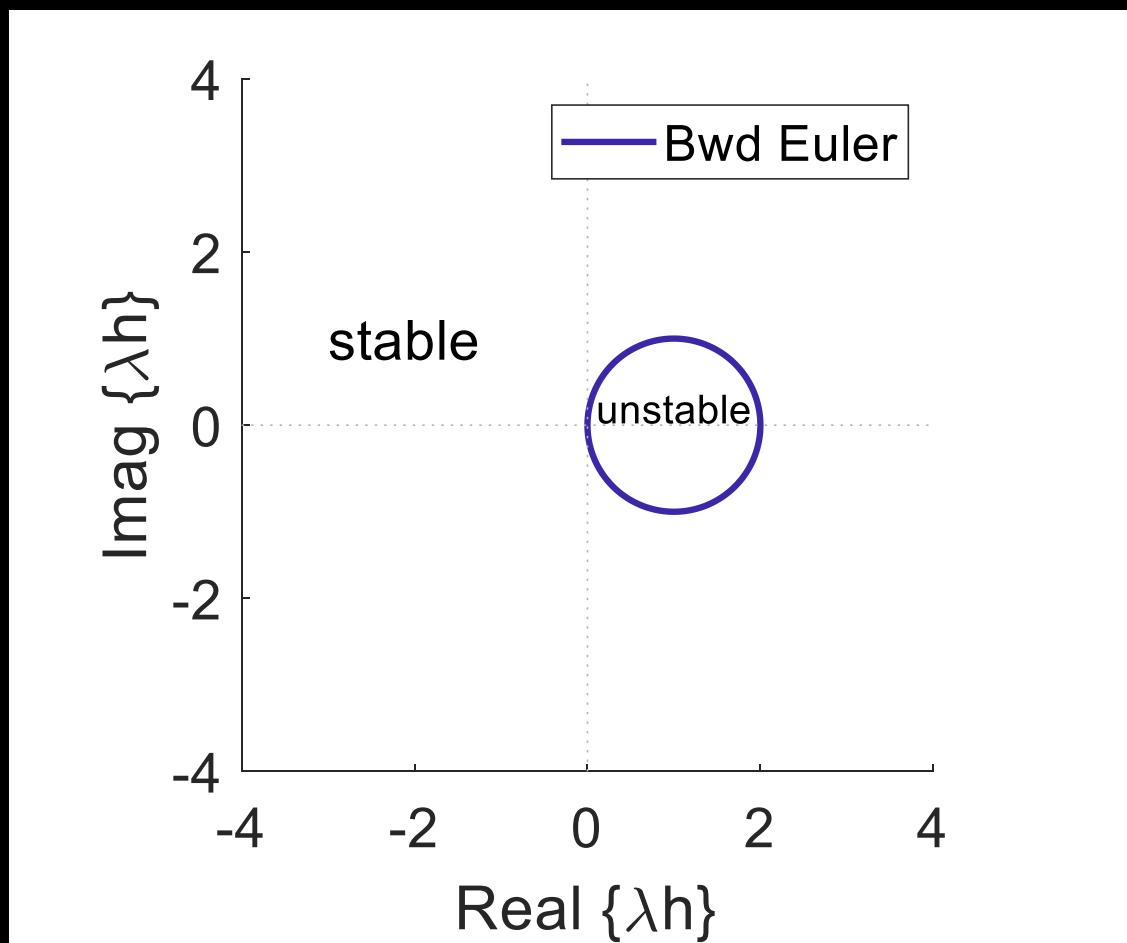
$$h < \sqrt{8}/\sqrt{10}$$



Stability Regions in the Complex Plane: Implicit Method

Stability boundaries:

Backward Euler: $1 - \lambda h = e^{i\theta}$

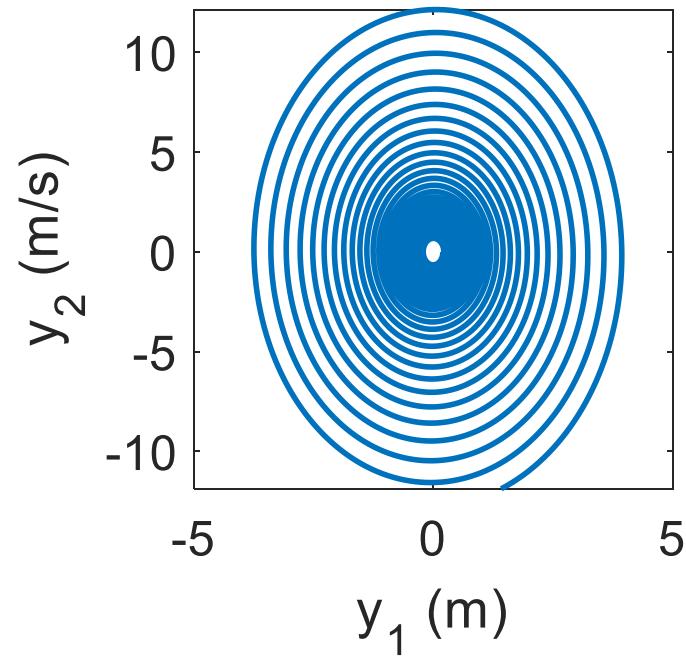
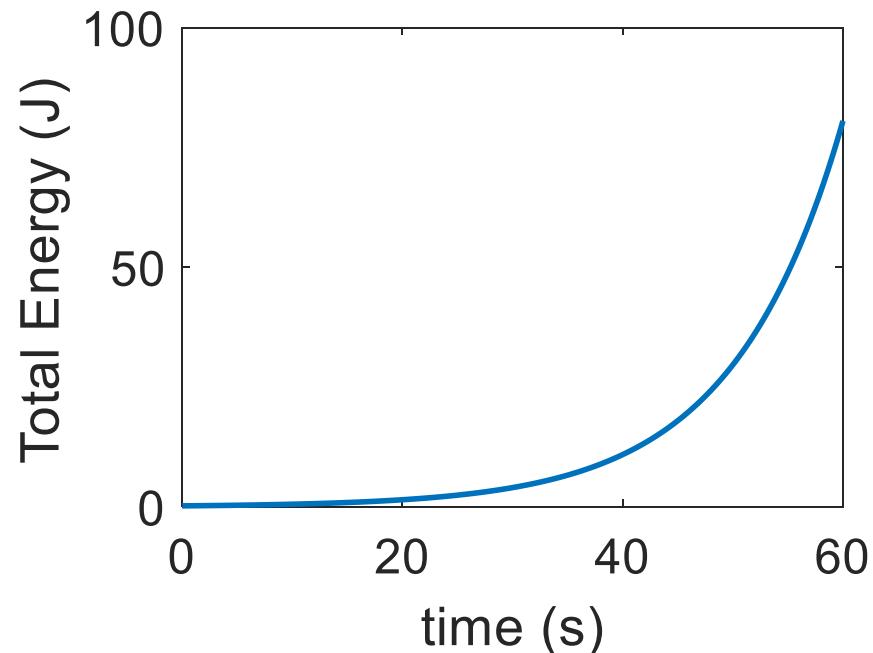


Chapter 3: Numerical ODEs | Semi-Implicit Methods & Non-Linear ODEs

Worked Example 29: Revisited: Total Energy

Our increasing error means we are violating the conservation of energy. The total energy in a mass-spring system (the Hamiltonian) is the sum of the kinetic and potential energies:

$$E = \frac{1}{2}my_2^2 + \frac{1}{2}ky_1^2$$



Conservation of Energy: Euler-Cromer Method

mass spring weq

- For Hamiltonian systems (where the dynamics are governed by conversion of kinetic to potential energy and vice versa) Euler-Cromer methods are useful.
 - E.g. our mass-spring system – we have a coupled system of ODEs.
- The Euler-Cromer method is semi-implicit – one variable is calculated explicitly (e.g. the velocity) and one implicitly (e.g. the position). For example for the mass-spring system:

Explicit velocity

$$y_{2,n+1} = y_{2,n} - h \frac{k}{m} y_{1,n} \quad (\text{Forward Euler on velocity})$$

Implicit position

$$y_{1,n+1} = y_{1,n} + h y_{2,n+1} \quad (\text{Backward Euler on position})$$

- Furthermore, as the method is only semi-implicit, it is simple to program, and does not require solution of an equation/matrix inversion
- There are other methods that achieve similar results including the: leapfrog, Euler-Richardson and the Verlet methods.
- A useful overview is in Cromer's paper: *Cromer, A, 1981. American Journal of Physics 49(5): 455-9.*

Worked Example 32: A Non-Linear ODE – the Pendulum

A 1 kg mass is hung at the end of a 1 m of ~~string~~ ^{beam}. Calculate how the angle of a pendulum varies when its initial angle is θ_0 rad and its initial velocity is $\dot{\theta}_0$ rad/s.

Our equation of motion is: At school small angle approximation
$$\frac{d^2\theta}{dt^2} = -\frac{g}{l} \sin \theta$$
 $\sin \theta \approx \theta$ if θ small \Rightarrow analytical solution

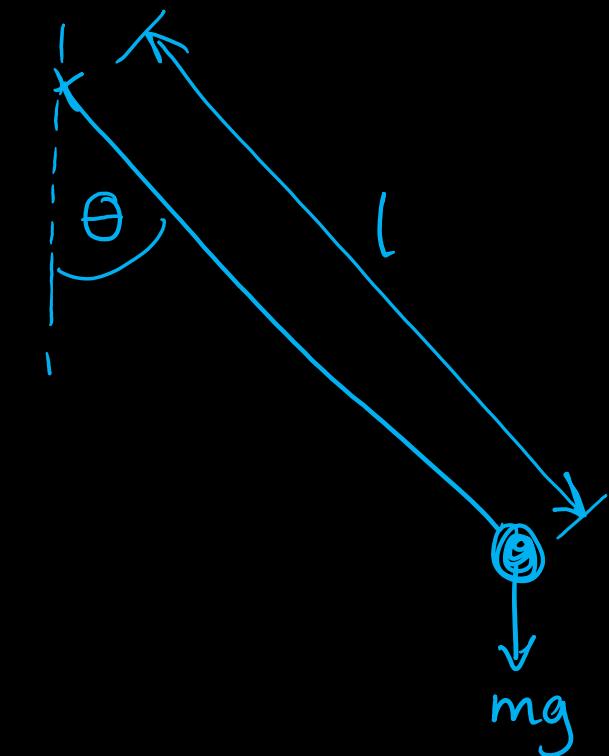
Large $\theta \Rightarrow$ numerical method

Second order ODE \rightarrow system of 1st order ODEs

Define $\theta_1 = \theta$ and $\frac{d\theta_1}{dt} = \theta_2 = f(t, \theta_1, \theta_2)$

$$\frac{d\theta_2}{dt} = -\frac{g}{l} \sin \theta_1 = p(t, \theta_1, \theta_2)$$

And our initial conditions are $\theta_1(0) = \theta_0$ and $\theta_2(0) = \dot{\theta}_0$



Worked Example 32: A Non-Linear ODE – the Pendulum

$$\begin{aligned} \Theta_{1,n+1} &= \Theta_{1,n} + h f_n \\ \Theta_{2,n+1} &= \Theta_{2,n} + h p_n \end{aligned} \quad \Rightarrow \quad y_{n+1} = y_n + h f_n$$

Applying explicit methods is trivial, e.g. the Forward Euler is:

$$\begin{aligned} \Theta_{1,n+1} &= \Theta_{1,n} + h (\Theta_{2,n}) \\ \Theta_{2,n+1} &= \Theta_{2,n} + h \left(-\frac{g}{l} \sin(\Theta_{1,n}) \right) = \Theta_{2,n} - \frac{gh}{l} \sin(\Theta_{1,n}) \end{aligned} \quad \Rightarrow \quad y_{n+1} = y_n + h f_{n+1}$$

Applying the backward Euler is more challenging:

$$\Theta_{1,n+1} = \Theta_{1,n} + h f_{n+1} = \Theta_{1,n} + h \Theta_{2,n+1} \quad \#1$$

$$\Theta_{2,n+1} = \Theta_{2,n} + h p_{n+1} = \Theta_{2,n} + h \left(-\frac{g}{l} \sin(\Theta_{1,n+1}) \right) \quad \#2$$

Sub #2 into #1

$$\Theta_{1,n+1} = \Theta_{1,n} + h \Theta_{2,n} - \frac{gh^2}{l} \sin(\Theta_{1,n+1})$$

Collect $\Theta_{1,n+1}$ terms on LHS

$$\Theta_{1,n+1} + \frac{gh^2}{l} \sin(\Theta_{1,n+1}) = \Theta_{1,n} + h \Theta_{2,n}$$

we solve this with root finding
(next term Monica)

Worked Example 32: A Non-Linear ODE – the Pendulum

Euler-Cromer, our semi-implicit method is also simple to apply. We can do either:

Explicit position $\theta_{1,n+1} = \theta_{1,n} + h f_n = \theta_{1,n} + h \dot{\theta}_{2,n}$ { Fwd Euler }

Implicit velocity $\theta_{2,n+1} = \theta_{2,n} + h p_{n+1} = \theta_{2,n} - \frac{gh}{l} \sin(\theta_{1,n+1})$ { Bwd Euler }

Or:

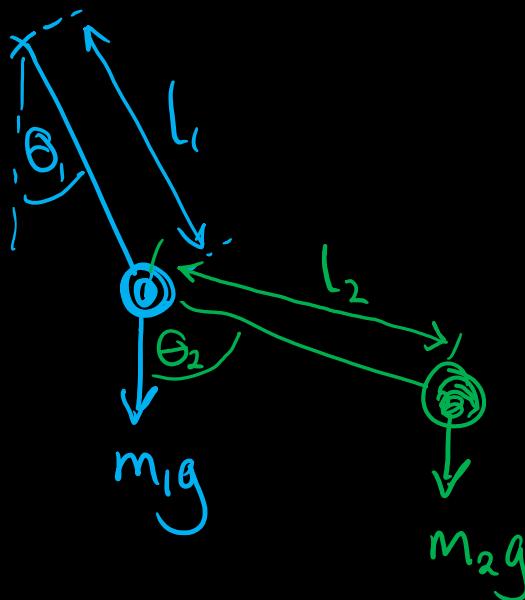
Explicit velocity $\dot{\theta}_{2,n+1} = \dot{\theta}_{2,n} - \frac{gh}{l} \sin(\theta_{1,n})$

Implicit position $\theta_{1,n+1} = \theta_{1,n} + h \dot{\theta}_{2,n+1}$

The solution of these methods will be demonstrated in the lecture.

Worked Example 33: the Double Pendulum

Consider a mass m_1 hung from the end of a some string with length l_1 . A second mass m_2 is then hung from the first mass with some string of length l_2 . This double pendulum is displaced, calculate how it moves.



The solution will be demonstrated computationally in the lecture.

[Double Pendulum -- from Eric Weisstein's World of Physics \(wolfram.com\)](#)

[Double Pendulum - YouTube](#)

Worked Example 33: the Double Pendulum

The equations of motion for a double pendulum are:

$$\begin{aligned}\frac{d^2\theta_1}{dt^2} = \ddot{\theta}_1 &= \frac{m_2 g \sin \theta_2 \cos(\theta_1 - \theta_2) - m_2 \sin(\theta_1 - \theta_2) (l_1 \dot{\theta}_1^2 \cos(\theta_1 - \theta_2) + l_2 \dot{\theta}_2^2) - (m_1 + m_2) g \sin \theta_1}{l_1(m_1 + m_2 \sin^2(\theta_1 - \theta_2))} \\ \frac{d^2\theta_2}{dt^2} = \ddot{\theta}_2 &= \frac{(m_1 + m_2)(l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + g \sin \theta_1 \cos(\theta_1 - \theta_2) - g \sin \theta_2) + m_2 l_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) \cos(\theta_1 - \theta_2)}{l_2(m_1 + m_2 \sin^2(\theta_1 - \theta_2))}\end{aligned}$$

We define $y_1 = \theta_1$, $y_2 = \dot{\theta}_1$, $y_3 = \theta_2$ and $y_4 = \dot{\theta}_2$ to develop a system of four ODEs for our numerical method:

$$y'_1 = y_2$$

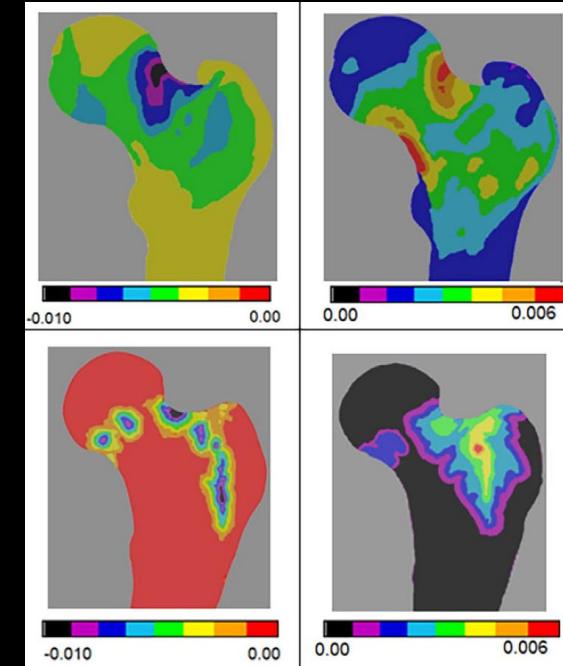
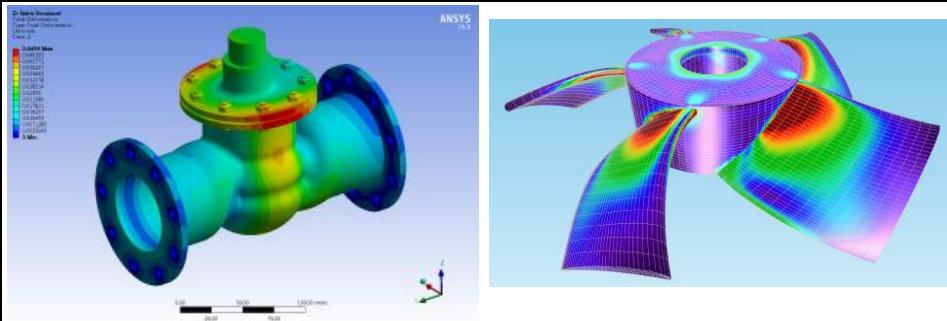
$$y'_2 = \frac{m_2 g \sin y_3 \cos(y_1 - y_3) - m_2 \sin(y_1 - y_3) (l_1 y_2^2 \cos(y_1 - y_3) + l_2 y_4^2) - (m_1 + m_2) g \sin y_1}{l_1(m_1 + m_2 \sin^2(y_1 - y_3))}$$

$$y'_3 = y_4$$

$$y'_4 = \frac{(m_1 + m_2)(l_1 y_2^2 \sin(y_1 - y_3) + g \sin y_1 \cos(y_1 - y_3) - g \sin y_3) + m_2 l_2 y_4^2 \sin(y_1 - y_3) \cos(y_1 - y_3)}{l_2(m_1 + m_2 \sin^2(y_1 - y_3))}$$

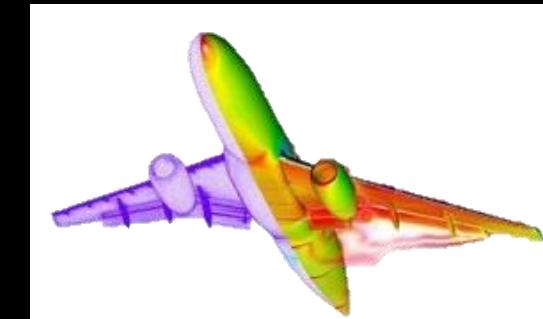
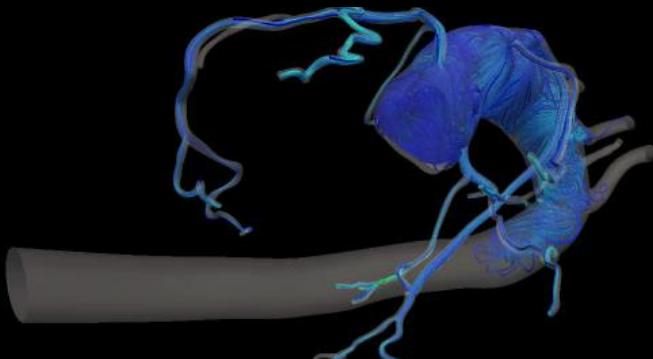
What's the point?

- Finite Element Analyses: stresses in solid bodies



Ridzwan, van Arkel,
Hansen et al. 2018

- Computational Fluid Dynamics: Solutions of Navier-Stokes equations



Chapter 3: Numerical ODEs | Boundary Value Problems

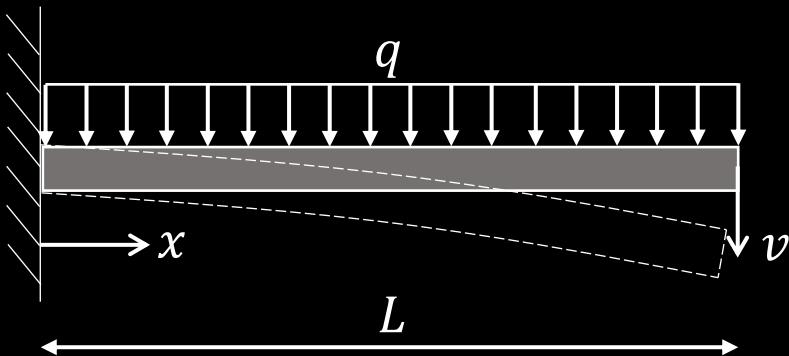
Finite Difference Methods for ODEs & PDEs: Learning Outcomes

Students should be able to:

- Distinguish between initial value problems (IVPs) and boundary value problems (BVPs)
- Solve boundary value ODE problems using the shooting method, or finite difference methods
- CALCULATORS IN RADIANS

Initial Value Problems Summary

- So far we have been solving initial value problems: we know a start point and we want to predict what happens as time progresses
 - E.g. radioactive decay, the double pendulum
 - Can also apply to position problems, e.g. cantilevered beam

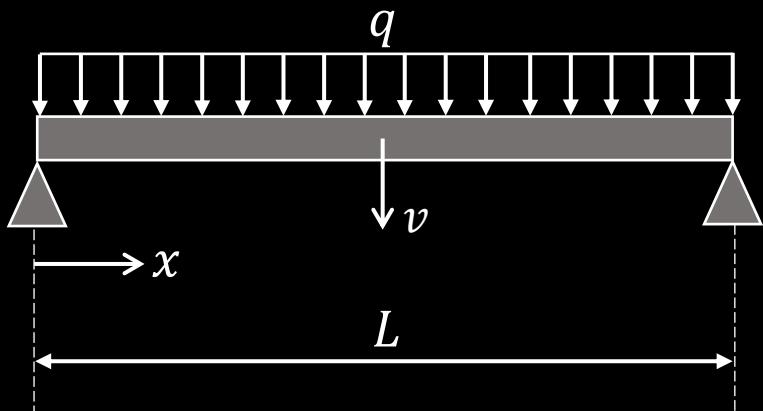


$$EI \frac{d^2v}{dx^2} = \frac{q}{2}(L - x)^2$$

Initial conditions (all known conditions at $x = 0$):
 $v(0) = 0$
 $\frac{dv}{dx}(0) = 0$

Boundary Value Problems: Beams

- We know conditions at both ends.
 - Often relevant for space/position based problems
 - Can be relevant for time problems if we know both the start and end points



$$EI \frac{d^2v}{dx^2} = \frac{q}{2}x(x - L)$$

Boundary conditions:
 $v(0) = 0$
 $v(L) = 0$

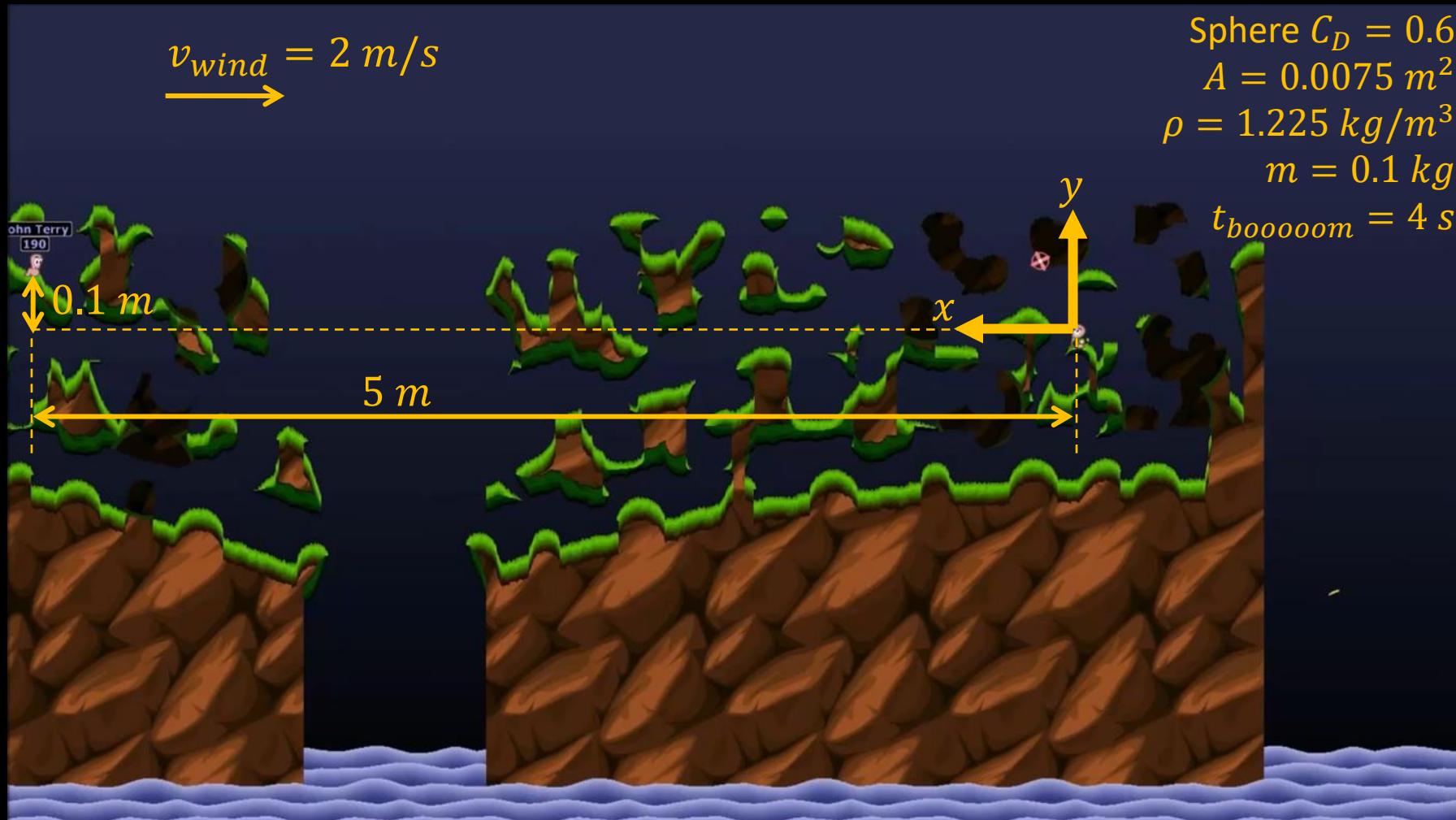
Boundary Value Problems: Ballistics



Solving Boundary Value Problems: The Shooting Method

- We have initial/boundary conditions at more than one time/location
- We can solve this by:
 - Estimating unknown initial values
 - Solve as an initial value problem
 - Compare the boundary value obtained to the actual boundary
 - Iterate until achieve desired boundary value
- Pro: easy to implement, even for non-linear problems
- Con: no guarantee of convergence

Worked Example 34: Shooting Method



Worked Example 34: Shooting Method

Let's consider just our x-direction ODE:

$$m \frac{d^2x}{dt^2} = -\frac{1}{2} \rho A C_D \left(\frac{dx}{dt} + v_{wind} \right)^2$$

$$ma = F_{\text{drag}}$$

Boundary conditions:

$$x(0) = 0, \quad x(4) = 5$$

Solution:

- 1) Rewrite as a system of first-order ODEs
- 2) Apply a numerical method for initial value problems (e.g. Forward Euler, RK4, etc) to calculate the value at the far boundary using an initial guess for x_2
- 3) Make an educated guess to try and improve the result
- 4) Interpolate to find a solution \leftarrow quicker & more elegant but not essential)

$$y' = f$$

$$y_{n+1} = y_n + hf_n$$

Worked Example 34: Shooting Method

- 1) Defining $x_1 = x$, $x_2 = \frac{dx}{dt}$ then our System of ODEs becomes:

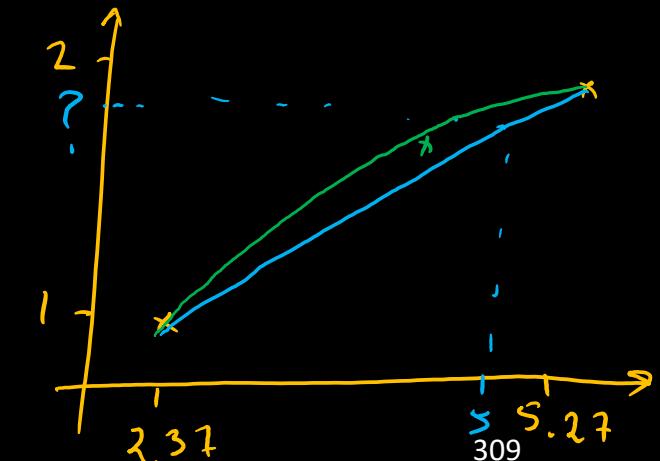
$$\frac{d}{dt} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_2 \\ -\frac{\rho A C_D}{2m} (x_2 + v_{wind})^2 \end{pmatrix}$$

- 2) Applying Forward Euler with time step $h = 0.01$ s:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_{n+1} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_n + 0.01 \left(\begin{pmatrix} x_{2,n} \\ -\frac{\rho A C_D}{2m} (x_{2,n} + v_{wind})^2 \end{pmatrix} \right)_n$$
$$x_1(0) = 0, \quad x_2(0) = \text{our guess} = 1 \text{ m/s}$$

- 3) Make some more “educated” guesses:

Initial velocity $x_2(0)$	Distance Travelled $x_1(4)$
1	2.372336986021419
2	5.266643184770463
1.8	4.705944807673327



Worked Example 34: Shooting Method

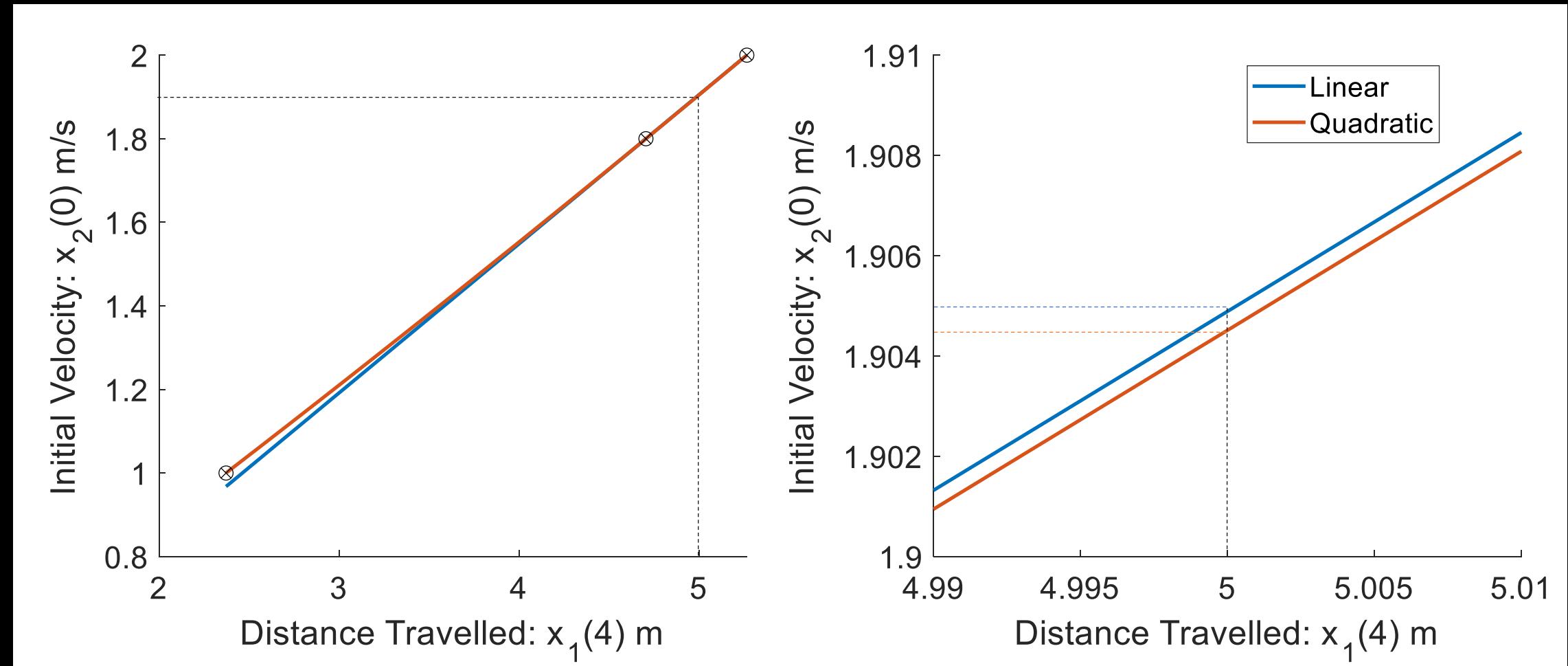
4) Interpolate to find the result

- We have bounded our desired result, and have three known points.
- If we interpolate $x_2(0)$ with respect to $x_1(4)$ we can find what our initial condition needs to be.
- Using our Non-Equal Spacing Newton Forward Interpolation Script (WE8):

Interpolation Method	Interpolated Initial Velocity: $x_2(0)$	Shooting Method Result: $x_1(4)$
Linear between 4.70... and 5.26...	1.904888904387084	5.001079553762604
Quadratic with 2.37..., 4.70... and 5.26...	1.904512857503790	5.000025712811980

We could apply the same method to find our initial velocity in the y direction, and combine to estimate our launch angle and velocity.

Worked Example 34: Plots of the interpolations

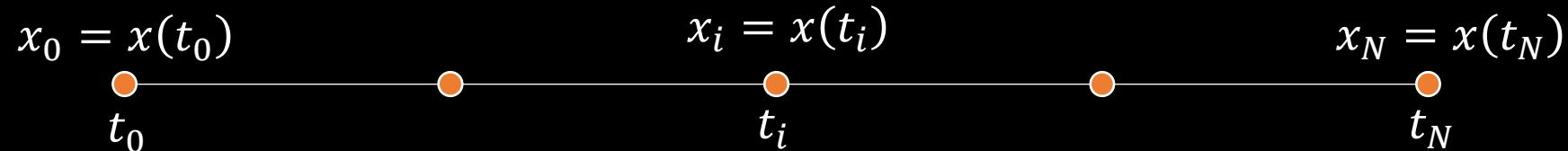


Solving Boundary Value Problems: Finite Difference Method

We can also solve the problems by converting the boundary value problem into a system of algebraic equations by replacing derivatives with finite difference approximations.

Consider an ODE of the form: $x'' = f(t, x, x')$ with boundary conditions $x(t_0) = \alpha$ and $x(t_N) = \beta$.

- 1) We divide the interval from $x = a$ to b into N subintervals with a spacing h :



- 2) We use finite approximations to replace the derivatives, e.g central difference approximations:

$$x'(t_i) = \frac{x_{i+1} - x_{i-1}}{2h}, \quad x''(t_i) = \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2}$$

{ slide 211

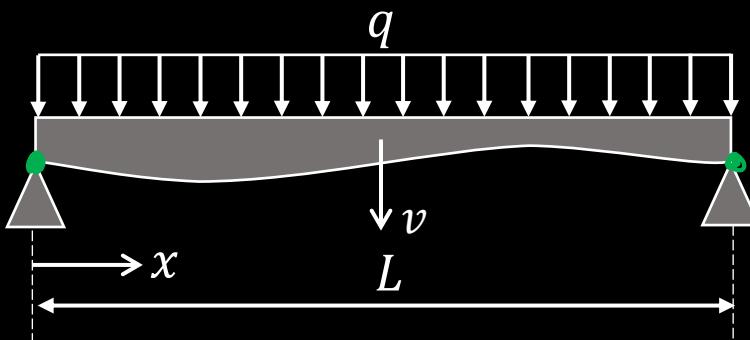
- 3) Solve the resulting algebraic equations to obtain a solution for the boundary value problem.

$$\frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} - f\left(t, x, \frac{x_{i+1} - x_{i-1}}{2h}\right) = 0$$

Solution can be direct (with matrices) or by iteration (Jacobi or Gauss-Seidel)

Worked Example 35: Finite Difference Method for a Beam BVP

Find the deflection of a beam with a non-constant cross-section using the finite difference method with 4 subintervals.



$$EI(x) \frac{d^2v}{dx^2} = \frac{q}{2}x(x - L)$$

Boundary conditions:

$$v(0) = 0, \quad v(L) = 0$$

$$E = 200 \text{ GPa}, \quad I(x) = I_0 \left(2 + \sin \left(\frac{2\pi x}{L} \right) \right) \text{ m}^4 \text{ where } I_0 = 8.34 \times 10^{-6} \text{ m}^4, \quad q = 160 \text{ N/m}, \quad L = 10 \text{ m}$$

step 1) Draw out the problem

$$v_0 = 0$$

•

$$x_0 = 0 \text{ m}$$

$$h = 2.5 \text{ m}$$

$$v_1 = ?$$

$$x_1 = 2.5 \text{ m}$$

$$v_i$$

$$v_2 = ?$$

$$x_2 = 5 \text{ m}$$

$$v_3 = ?$$

$$x_3 = 7.5 \text{ m}$$

$$v_4 = 0$$

•

$$x_4 = 10 \text{ m}$$

$$I(x) = I_0 \left(2 + \sin\left(\frac{2\pi x}{L}\right) \right) \quad EI(x) \frac{d^2v}{dx^2} = \frac{q}{2} x(x - L)$$

Worked Example 35: Finite Difference - Direct Method

Step 2) we substitute derivative terms for finite difference approx. (num diff.)

Choose to use central difference : $\frac{d^2v}{dx^2} = \frac{v_{i+1} - 2v_i + v_{i-1}}{h^2}$ } slide 211

Writing out our ODE: $EI(x_i) \left(\frac{v_{i+1} - 2v_i + v_{i-1}}{h^2} \right) = \frac{q}{2} x_i(x_i - L) g(x_i)$

Unknowns on LHS, known RHS:

$$v_{i+1} - 2v_i + v_{i-1} = \frac{h^2 q x_i (x_i - L)}{2EI_0 \left(2 + \sin\left(\frac{2\pi x_i}{L}\right) \right)}$$

$i=0$; $v_0 = 0$ (Boundary Condition)

$$i=1; (v_2) - 2(v_1) + v_0 = g(x_1)$$

$$i=2; v_3 - 2v_2 + v_1 = g(x_2)$$

$$i=3; v_4 - 2v_3 + v_2 = g(x_3)$$

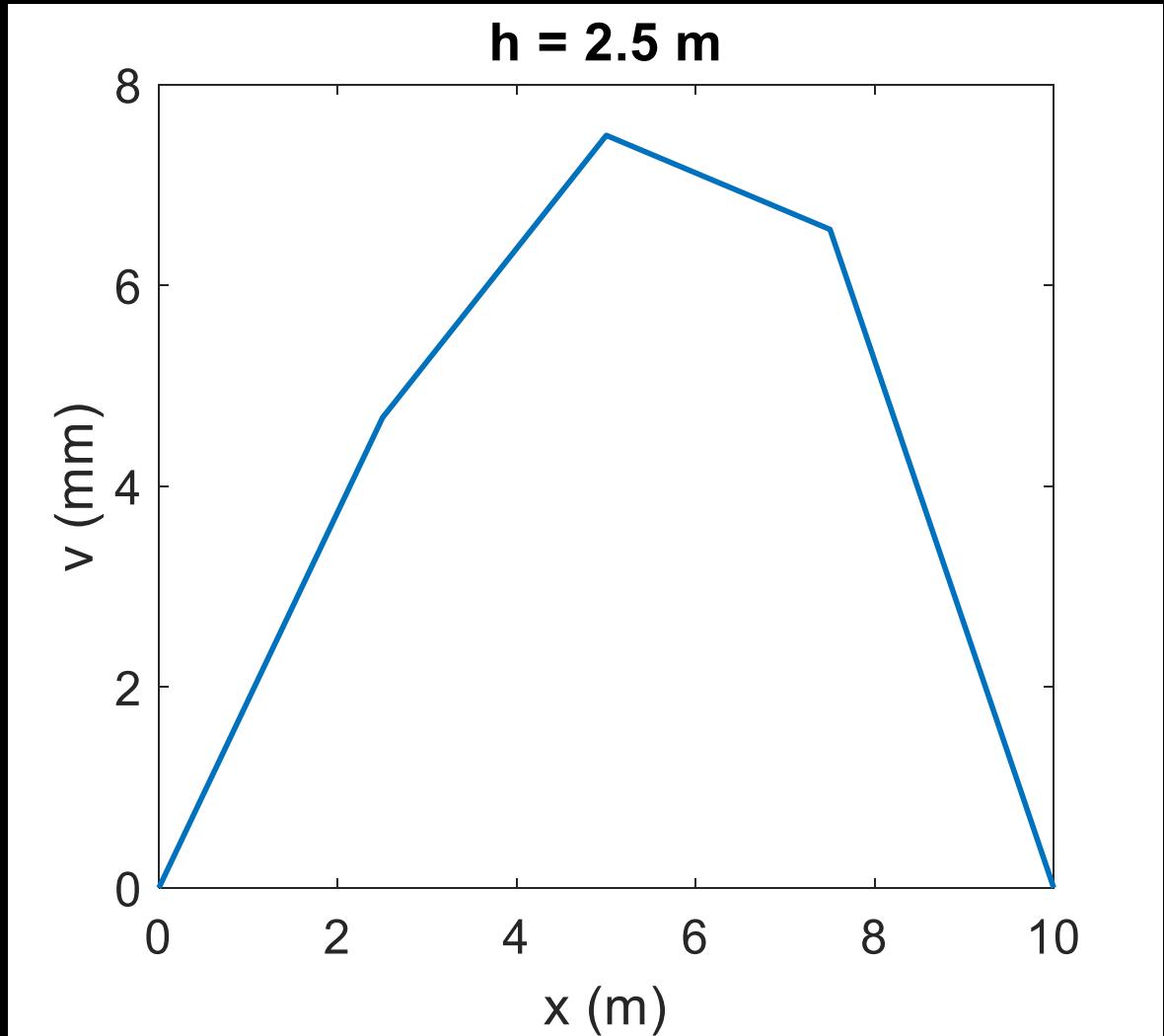
$$i=4; v_4 = 0 \quad (\text{B.C.})$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} 0 \\ g(x_1) \\ g(x_2) \\ g(x_3) \\ 0 \end{pmatrix}$$

Worked Example 35: Finite Difference - Direct Method

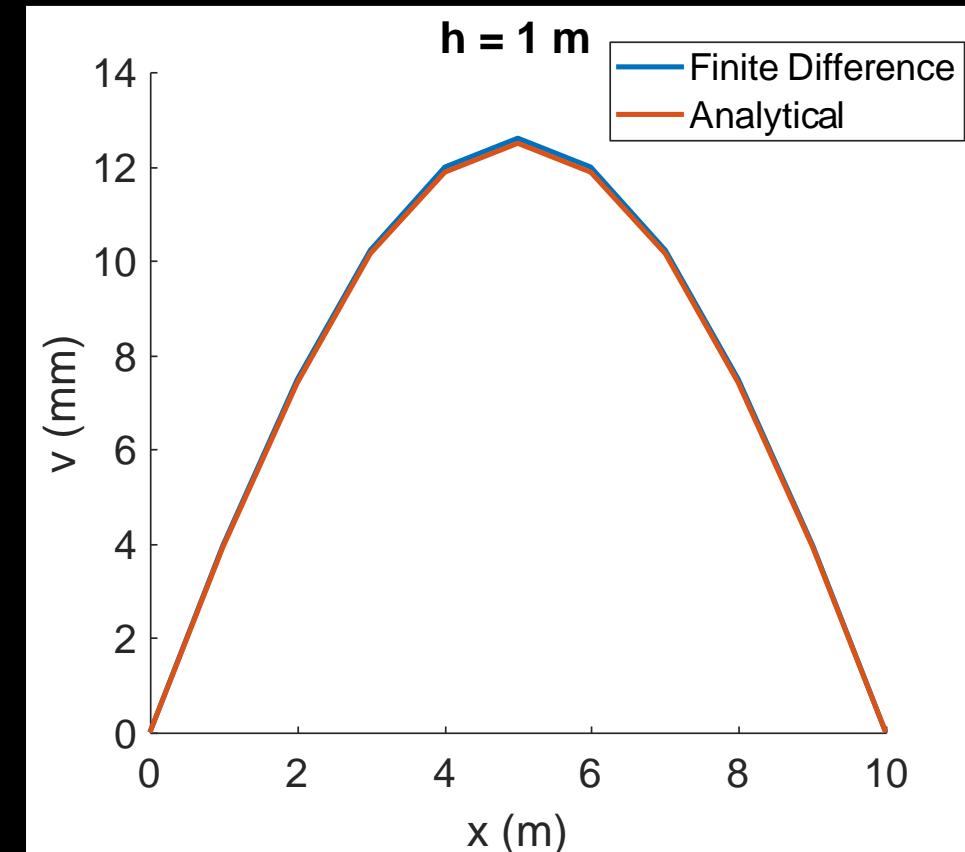
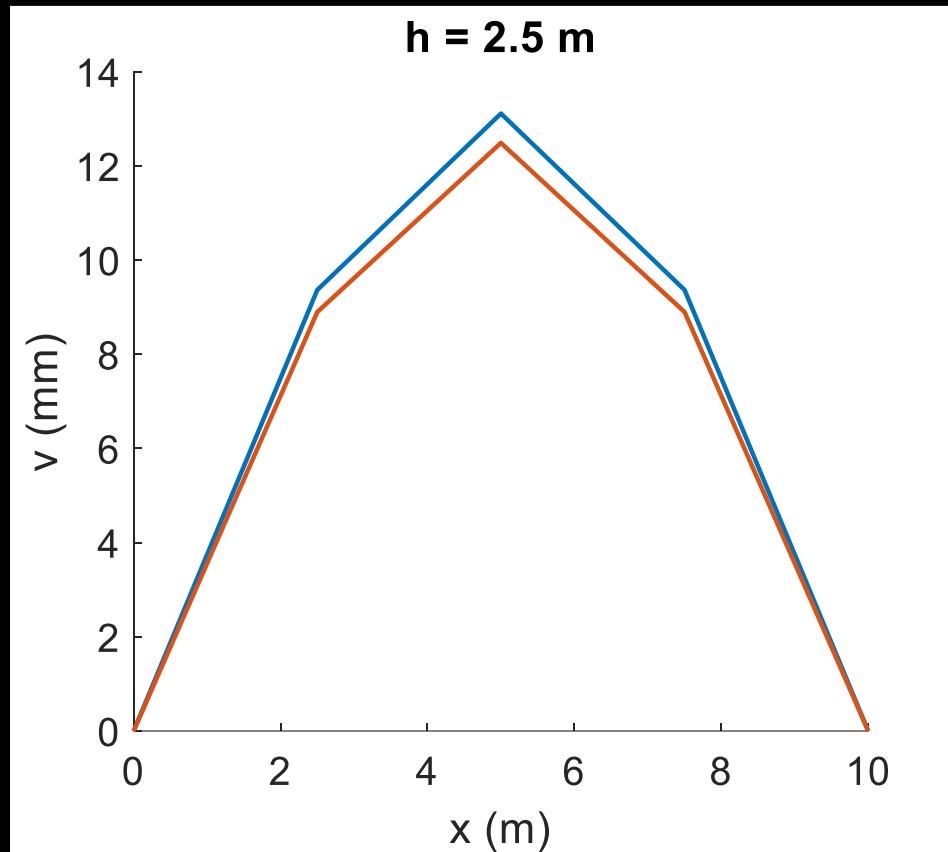
3) Solve: $\boldsymbol{v} = \boldsymbol{A}^{-1}\boldsymbol{d} = \begin{pmatrix} 0 \\ 4.7 \\ 7.5 \\ 6.6 \\ 0 \end{pmatrix} \text{ mm}$

What do you think our order of accuracy is?



Worked Example 35: Finite Difference - Direct Method

If $I(x) = I_0 = \text{constant}$ then we know the analytical result: $v(x) = \frac{qx(L^3 - 2Lx^2 + x^3)}{24EI_0}$



Worked Example 35: Finite Difference - Iterative Method

Steps 1 and 2 stay the same. Thus, from the beam BVP:

$$v_{i+1} - 2v_i + v_{i-1} = \frac{qh^2}{2EI_0} \frac{x_i(x_i - L)}{\left(2 + \sin\left(\frac{2\pi x_i}{L}\right)\right)}$$

k is my iteration step

3) Set up iterative scheme, either Jacobi:

$$(v_i)_{k+1} = \frac{(v_{i+1})_k + (v_{i-1})_k}{2} - \frac{qh^2}{4EI_0} \frac{x_i(x_i - L)}{\left(2 + \sin\left(\frac{2\pi x_i}{L}\right)\right)}$$

Or Gauss-Seidel:

$$(v_i)_{k+1} = \frac{(v_{i+1})_k + (v_{i-1})_{k+1}}{2} - \frac{qh^2}{4EI_0} \frac{x_i(x_i - L)}{\left(2 + \sin\left(\frac{2\pi x_i}{L}\right)\right)}$$

Make initial guess for each internal node v_i .

Iterate to solve (demonstrated in Excel).

Summary

- Initial Value Problems
 - All initial conditions imposed at the same location/time
 - Solved using Euler, RK4, BDF, etc.
- Boundary Value Problems
 - Initial/boundary conditions at more than one location/time
 - Solve using either the Shooting Method or Finite Difference Method
 - Shooting method easy to implement, but not guaranteed to converge
 - Finite Difference Method can be solved directly (with matrices) for linear problems
 - Finite Difference Method can also be solved iteratively (Jacobi/Gauss-Seidel). Iterative schemes can also work for non-linear systems (doesn't always converge).

Chapter 4: Transforms

Dr Richard J van Arkel

Chapter 4: Transforms | Fourier Transform and Its Properties

Transforms: Learning Outcomes

Students should be able to:

- Calculate Fourier Transforms and Inverse Fourier Transforms from the integral formulae
- Prove and apply Fourier and Laplace Transform properties
- Solve ODEs with Laplace Transforms

Revision: Fourier Series

Any periodic function $f(t)$ can be expressed as a Fourier Series:

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{2n\pi t}{T} + b_n \sin \frac{2n\pi t}{T} \right)$$

Where:

$$a_n = \frac{2}{T} \int_d^{d+T} f(t) \cos \frac{2n\pi t}{T} dt \quad \text{and} \quad b_n = \frac{2}{T} \int_d^{d+T} f(t) \sin \frac{2n\pi t}{T} dt$$

The function is expressed as a linear function of sinusoidal basis functions.

As more and more higher frequency sinusoids are added, the Fourier Series tends to the true function.

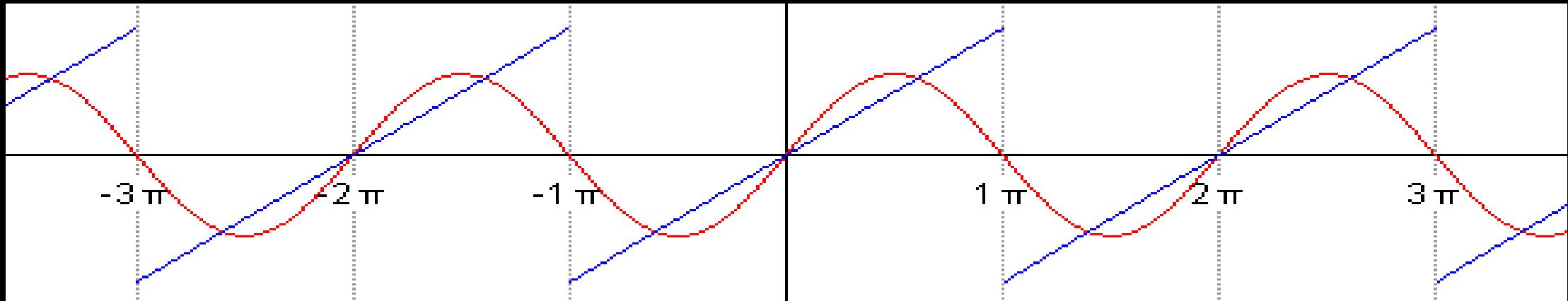
Revision: Fourier Series

For example, for the function:

$$f(t) = t \text{ if } -\pi < t < \pi \text{ and } f(t + 2\pi) = f(t)$$

Its Fourier series is:

$$f(t) = 2 \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n} \sin nt$$



Fourier Transforms: link to solutions of PDEs

- Fourier Series provide an expansion of any periodic function
- A further generalisation of these ideas is the Fourier Transform
- Recall that when studying separable solutions to PDEs, we constructed a general solution which was a ‘sum’ over all values of some real number k
- In the diffusion equation, we also considered a finite length interval, at the ends of which we applied boundary conditions. This finite length interval can be identified with the d and T of the periodic function in a Fourier series expansion
- When we applied the boundary conditions, this resulted in a restriction of the parameter k to be some integer multiple m of π (or something like it)
- We could then write the ‘sum’ over k as a proper sum over m
- Suppose instead, that we did not have a finite length interval, and that it extends from $t = -\infty$ to $t = +\infty$. (It is then NOT periodic). We cannot apply boundary conditions in this case. Thus, we cannot reduce the ‘sum’ of real numbers to a proper sum over integers
- We can still do it - but now the ‘coefficients’ will be a function of the continuous variable k and the ‘sum’ will be an integral

Fourier Transforms: Definition

- It is also common terminology to use the symbol ω instead of k
- Given this, and considering the complex form of Fourier's expansion we may define the Fourier transform $F(\omega)$ of a function $f(t)$ as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t} dt$$

- What this represents is an entirely equivalent way of representing the function $f(t)$. We have transformed the function $f(t)$ which is a function of the variable t into a function $F(\omega)$ which is a function of the variable ω .
- What is ω ? We have seen earlier how ω represents the frequency of sinusoids (in the variable t) which, when combined appropriately, represent the function $f(t)$. The function $F(\omega)$, then represents the amount of each frequency ω 'present' in the function $f(t)$.
- Remember, in the Fourier series, we had a series of constants which represented the 'amplitudes' of the sinusoids.

$$\text{mag} = |a + ib| = \sqrt{a^2 + b^2}$$

$$\text{phase} = \tan^{-1}\left(\frac{b}{a}\right)$$

Inverse Fourier Transforms

- This representation $F(\omega)$ of the function $f(t)$ is entirely equivalent.
- Therefore we can transform from the function $F(\omega)$ back to the function $f(t)$.
This is the inverse Fourier transform calculated by:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega$$

- The two functions $f(t)$ and $F(\omega)$ are called a Fourier transform pair.
- We will usually denote the function of t in lower case and the function of ω in upper case.

Comparison to Fourier Series

- Continuous periodic functions can be expressed as Fourier Series: a "discrete" superposition of exponentials
- Continuous non-periodic functions can be Fourier Transformed: a “continuous” superposition of exponentials

Fourier Series	Periodic Functions Only	$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i\omega_n t}$	$c_n = \frac{1}{T} \int_d^{d+T} f(t) e^{-i\omega_n t} dt$
Fourier Transform	Also works for non-periodic functions	$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega$	$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$

$$f(t) \leftrightarrow F(\omega)$$

$$f \neq F$$

$$f(t) = e^{-at|t|}$$

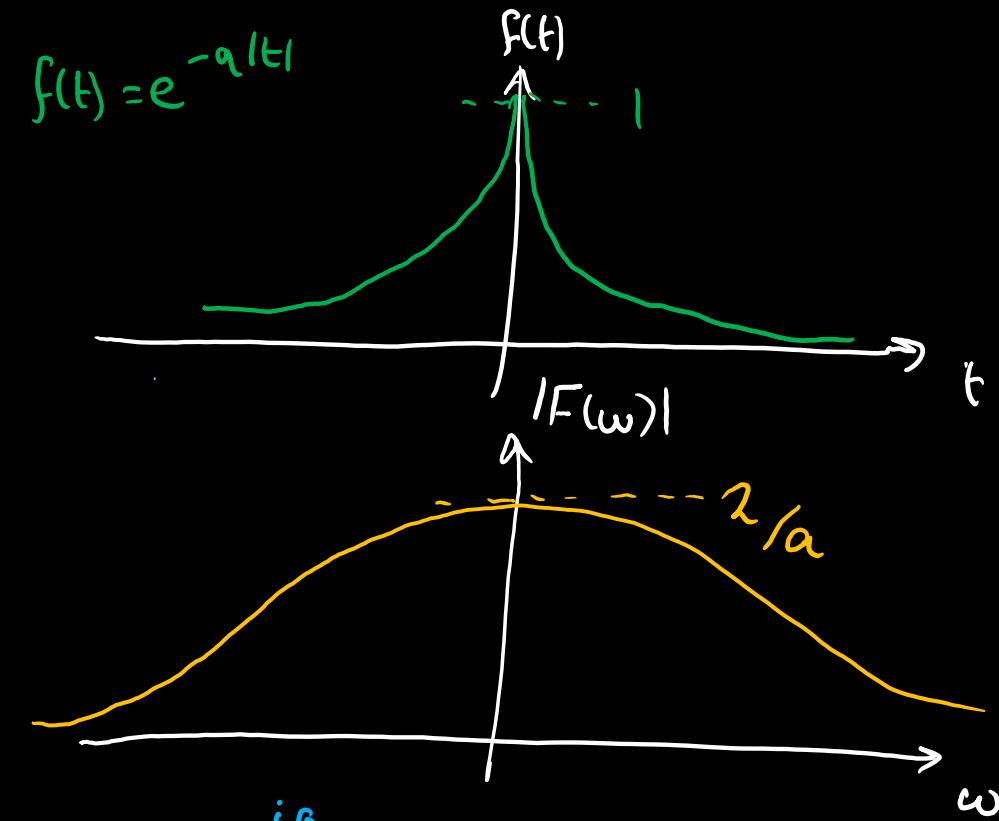
$$f(t)$$

$$|F(\omega)|$$

Worked Example 36

$$\begin{aligned}
 F(\omega) &= \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt = \int_{-\infty}^{\infty} e^{-at|t|} e^{-i\omega t} dt \\
 &= \int_{-\infty}^{0} e^{at} e^{-i\omega t} dt + \int_{0}^{\infty} e^{-at} e^{-i\omega t} dt \\
 &= \int_{-\infty}^{0} e^{(a-i\omega)t} dt + \int_{0}^{\infty} e^{-(a+i\omega)t} dt \\
 &= \left[\frac{1}{a-i\omega} e^{(a-i\omega)t} \right]_0^{-\infty} + \left[-\frac{1}{a+i\omega} e^{-(a+i\omega)t} \right]_0^{\infty} \\
 &= \left[\frac{1}{a-i\omega} e^{at} e^{-i\omega t} \right]_0^{-\infty} + \left[-\frac{1}{a+i\omega} e^{-at} e^{-i\omega t} \right]_0^{\infty} \quad \textcircled{a} \\
 &= \frac{1}{a-i\omega} e^0 e^0 - 0 + 0 - \left(-\frac{1}{a+i\omega} e^0 e^0 \right) = \frac{1}{a-i\omega} + \frac{1}{a+i\omega}
 \end{aligned}$$

$$\frac{a+i\omega + a-i\omega}{(a-i\omega)(a+i\omega)} = \frac{2a}{a^2 + \omega^2}$$



$$\begin{aligned}
 e^{i\theta} &= \cos \theta + i \sin \theta \\
 e^{i\infty} &= \underbrace{\cos \infty + i \sin \infty}_{\text{less than } 111} \\
 &\text{but undefined}
 \end{aligned}$$

Properties of Fourier Transforms

- The transforms are not generally easy to do. To help, there are many useful properties that can be used:

	Function/Signal	Transform
Linearity:	$af(t) + bg(t)$	$aF(\omega) + bG(\omega)$
Shift in time:	$f(t - a)$	$e^{-ia\omega} F(\omega)$
Shift in frequency:	$e^{iat} f(t)$	$F(\omega - a)$
Differentiation:	$\frac{df(t)}{dt}$	$i\omega F(\omega)$
Scaling:	$f(at)$	$\frac{1}{ a } F\left(\frac{\omega}{a}\right)$

Worked Example 37 Prove shift in time

$$G(\omega) = \int_{-\infty}^{\infty} g(t) e^{-i\omega t} dt$$

$$= \int_{-\infty}^{\infty} f(t-a) e^{-i\omega t} dt$$

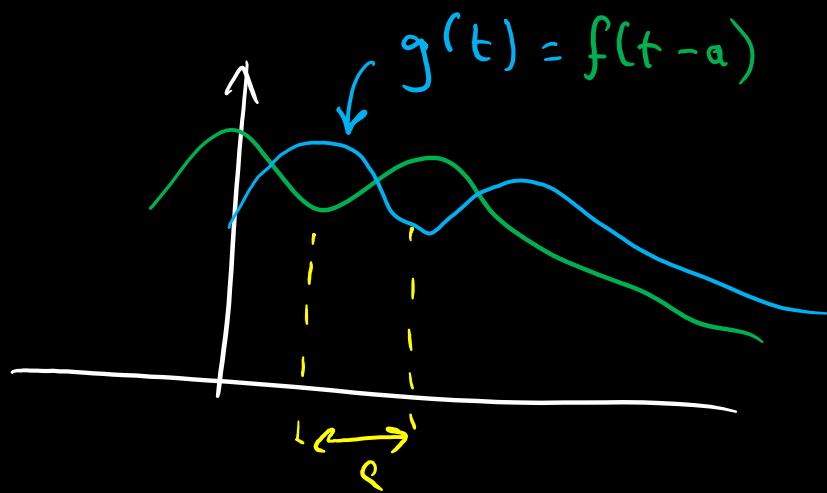
substitution $\tau = t-a \Rightarrow \frac{d\tau}{dt} = 1 \Rightarrow d\tau = dt$

$$= \int_{-\infty}^{\infty} f(\tau) e^{-i\omega(\tau+a)} d\tau$$

$$= \int_{-\infty}^{\infty} f(\tau) e^{-i\omega\tau} e^{-i\omega a} d\tau$$

$$= e^{-i\omega a} \underbrace{\int_{-\infty}^{\infty} f(\tau) e^{-i\omega\tau} d\tau}_{= \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt = F(\omega)}$$

$$= e^{-i\omega a} F(\omega)$$



$$\int_0^1 x^2 dx$$

$$= \left[\frac{x^3}{3} \right]_0^1 = \frac{1}{3}$$

Prove derivative property

Inverse transform:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega$$

$$\frac{df}{dt} = \frac{d}{dt} \left[\text{---} \right]$$

$$\frac{df}{dt} = \frac{1}{2\pi} \int_{-\infty}^{\infty} [F(\omega) i\omega] e^{i\omega t} d\omega$$

$$\tilde{g}(t) \quad G(\omega)$$

$$g(t) = \underbrace{\frac{1}{2\pi} \int_{-\infty}^{\infty} G(\omega) e^{i\omega t} d\omega}_{\text{inverse transform formula}}$$

$$g(t) \leftrightarrow G(\omega)$$

$$\frac{df}{dt} \leftrightarrow i\omega F(\omega)$$

$$f(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

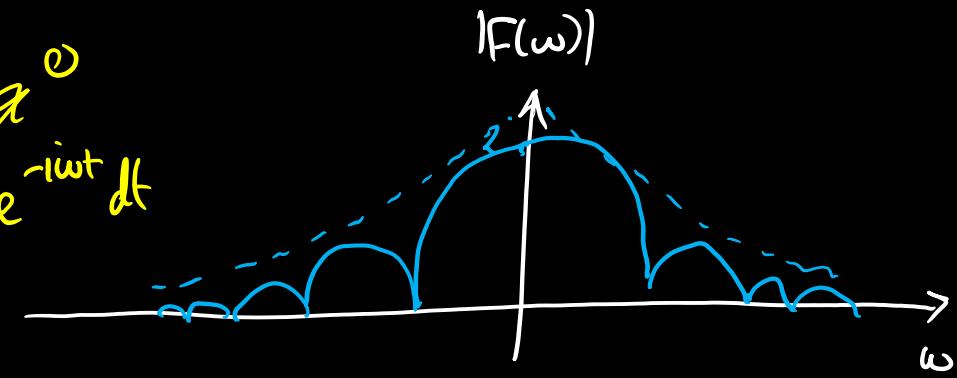
$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(\omega) e^{i\omega t} d\omega$$

Chapter 4: Transforms | Fourier Transform Pairs & Examples

$$f_1(t) =$$

$$\begin{aligned} F_1(\omega) &= \int_{-\infty}^{\infty} f_1(t) e^{-i\omega t} dt = \int_{-\infty}^{-1} 0 \cdot e^{-i\omega t} dt + \int_{-1}^{1} 1 \cdot e^{-i\omega t} dt + \int_{1}^{\infty} 0 \cdot e^{-i\omega t} dt \\ &= \left[-\frac{1}{i\omega} e^{-i\omega t} \right]_{-1}^{1} = -\frac{1}{i\omega} e^{-i\omega b} + \frac{1}{i\omega} e^{i\omega b} \end{aligned}$$

$$\begin{aligned} e^{i\theta} &= \cos\theta + i\sin\theta \quad \#1 \\ e^{-i\theta} &= \cos\theta - i\sin\theta \quad \#2 \end{aligned} \quad \begin{aligned} &= \frac{2}{\omega} \left(e^{i\omega b} - e^{-i\omega b} \right) \quad \#1 - \#2 \\ &= \frac{2}{\omega} \sin(\omega b) \end{aligned}$$

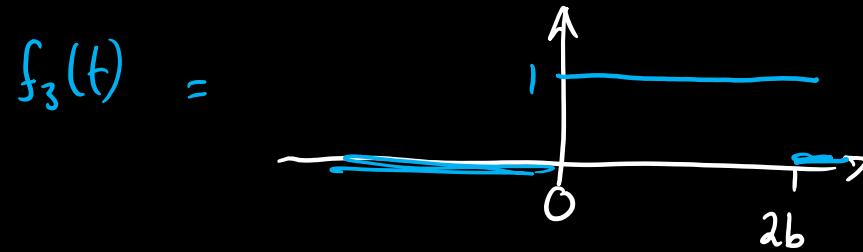


$$f_2(t) =$$

where b is a free constant

$$\begin{aligned} f_2(t) &= f_1\left(\frac{t}{b}\right) \\ \text{Scaling where } a &= \frac{1}{b} \end{aligned}$$

$$\begin{aligned} F_2(\omega) &= |b| F_1(\omega b) \\ &= \frac{|b| 2}{\omega b} \sin(\omega b) = \frac{2}{\omega} \sin(\omega b) \end{aligned}$$

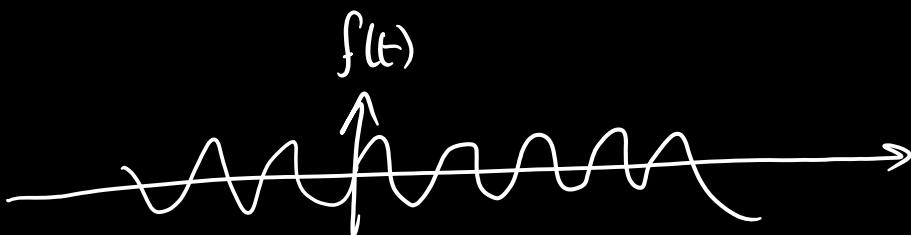


$$f_3(t) = f_2(t - b)$$

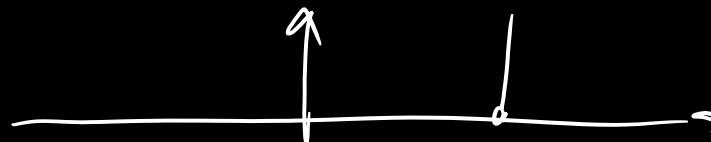
shift in time where ' $a = b$ '

$$F_3(\omega) = e^{-ib\omega} F_2(\omega) = e^{-ib\omega} \frac{2}{\omega} \sin(\omega b)$$

We need a way to transform \sin & \cos



$$|F(\omega)|$$



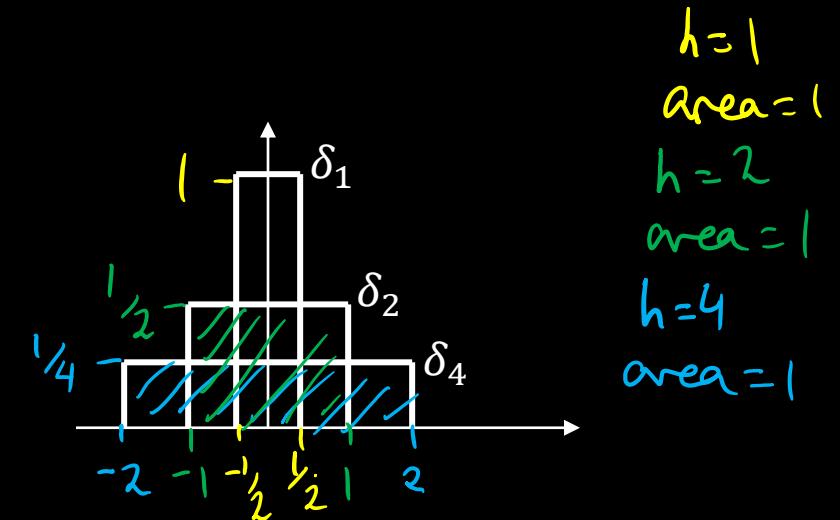
Fourier Transforms: The Dirac Delta δ

- When doing Fourier transforms, an important function (actually its not really a function - its a functional) often arises, called the **Dirac delta**
- It can be viewed as a limit of the following function:

$$\delta_h(x) = \begin{cases} 1/h & \text{if } |x| < h/2 \\ 0 & \text{if } |x| > h/2 \end{cases}$$

- Importantly, for any h , $\int_{-\infty}^{\infty} \delta_h(x) dx = 1$
- The Dirac delta is defined as the limit of these functions as $h \rightarrow 0$
- It is zero everywhere, except for when $x = 0$, where it is infinite
- It has the following important property, known as the ‘sifting’ property (‘sifting’ not ‘shifting’):

$$\int_{-\infty}^{\infty} \delta(x - x_0) f(x) dx = f(x_0)$$



$\delta(x - x_0) = \text{zero everywhere}$
except x_0 where
it is infinite

Fourier Transform: Common Transform Pairs

Function/Signal	\longleftrightarrow	Transform
1		$2\pi\delta(\omega)$
$\delta(t)$		1
e^{iat}	$(F(\omega))$	$2\pi\delta(\omega - a)$
cos at		$\pi[\delta(\omega - a) + \delta(\omega + a)]$
$\sin at$		$\frac{\pi}{i}[\delta(\omega - a) - \delta(\omega + a)]$
	$u(t)$ <small>Heaviside</small> κ <small>unit step</small>	$\pi\delta(\omega) + \frac{1}{i\omega}$
	$H(t)$ $u(t)$ $G(t)$	

Discrete Fourier Transforms

- So far, we have considered Fourier Series and Transforms of continuous period and non-periodic functions.
- The real world application of Fourier Transforms is most commonly applied to **discrete** data. For example:
 - Sounds/music sampled at points in time
 - Images from a digital camera
 - Experimental measurements
- There is a form of Fourier transform, called the **discrete Fourier transform (DFT)** which applies to such data
- Suppose we have a signal x samples at N evenly spaced points in time (or space):
 $\{x_0, x_1, x_2, \dots, x_{N-1}\}$ where $x_n = x(n\Delta t)$
- The DFT of this discrete signal (and its inverse) is defined as:

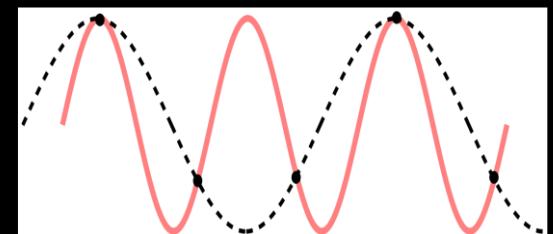
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i k n}{N}} \text{ for } k = 0, \dots, N-1 \quad x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i k n}{N}} \text{ for } n = 0, \dots, N-1$$

Notes on Discrete Fourier Transforms

- Note that the X_k are complex numbers, even if the x_n are real
- Each X_k , when expressed in polar form, represent the amplitude and phase of the sinusoidal component of frequency k/N cycles per Δt of the signal x
- Commonly, we want the frequency to be represented in Hertz (Hz), i.e., the number of cycles/second
- In such a case the X_k , when expressed in polar form, represent the amplitude and phase of the sinusoidal component of frequency $k/(N\Delta t)$ Hz of the signal x
- We can deduce a number of important results from this:
- We have a frequency resolution $\Delta f = 1/(N\Delta t)$, that is, the DFT represents the frequency components of the signal at frequencies separated by Δf
- The highest frequency we can obtain is $f_{max} = 1/\Delta t$
- Thus if we want greater frequency resolution - we should increase N , if we want to investigate higher frequencies, we should reduce Δt

Notes on Discrete Fourier Transforms

- Note also, that the transform converts N real numbers into N complex numbers. This does not make sense - since each complex number is made up of a real and an imaginary part. We seem to be changing N numbers for $2N$ numbers.
- In fact this is not the case, there are N complex numbers, but they occur in conjugate pairs (i.e., of the form $a + ib$ and $a - ib$)
- So-called ‘aliasing’ means that only frequencies $< f_{max}/2$ are meaningful - in effect we ‘ignore’ the rest
- Our DFT applies to a signal of fixed length (i.e. N samples) - but the Fourier transform generally applies to signals of infinite length. The reason for this is that the DFT is actually a Fourier transform of a periodic signal, with period equal to $N\Delta t$.
- In this way, the DFT is actually more like the Fourier series expansion
- The DFT is therefore periodic in both the time and frequency domains

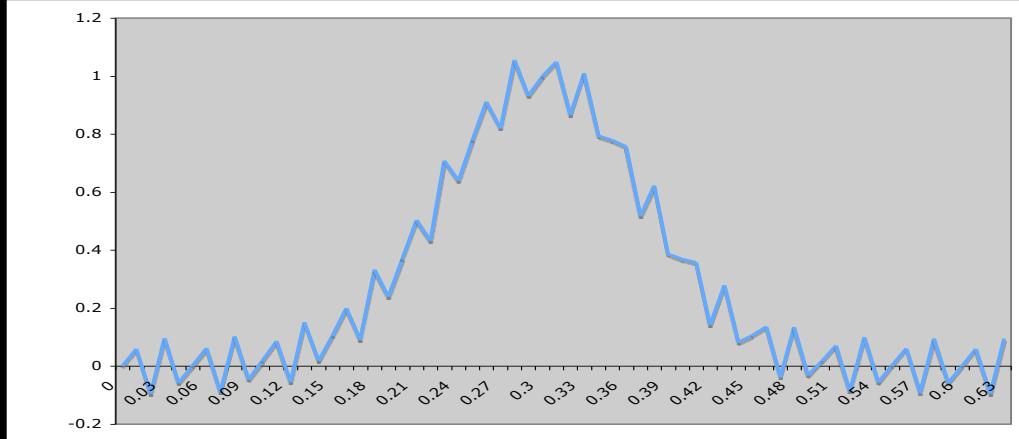


Discrete Fourier Transforms: Python & MatLab

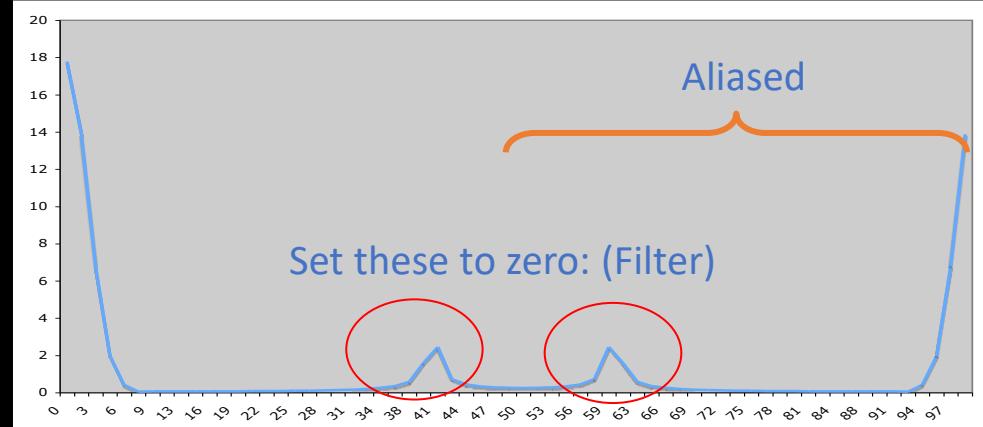
- Program such as Python and MatLab have inbuilt functions for computing the DFT of a sequence
- This is a special form of DFT, which can be computed extremely rapidly, and is known as the Fast Fourier Transform (FFT)
- The coding input is typically $dftx=fft(x)$.
- We typically then modify the $dftx$ data in the frequency domain (e.g. to filter noise).
- The inverse Fourier Transform is then applied: $xnew=ifft(dftx)$
- Many FFT algorithms require the data input to be a power of 2.
 - Functions such as `nextpow2` are often used in combination with `fft`.

Filtering with the DFT

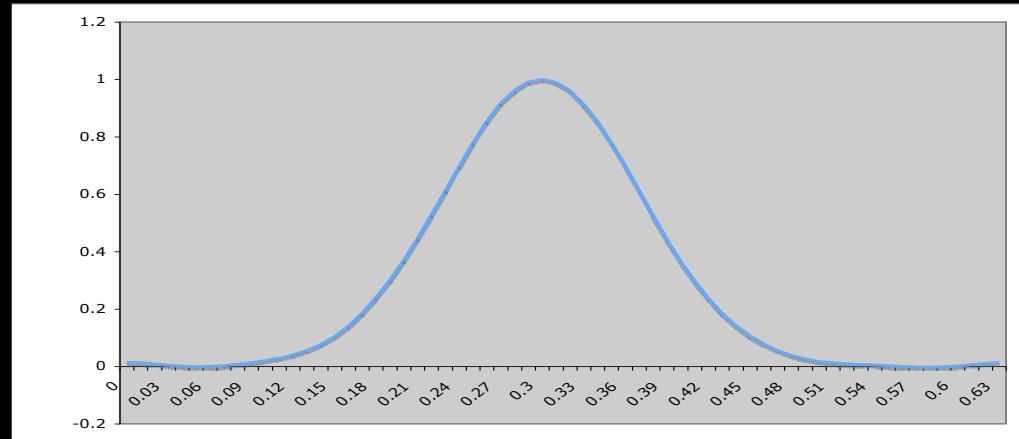
$x =$



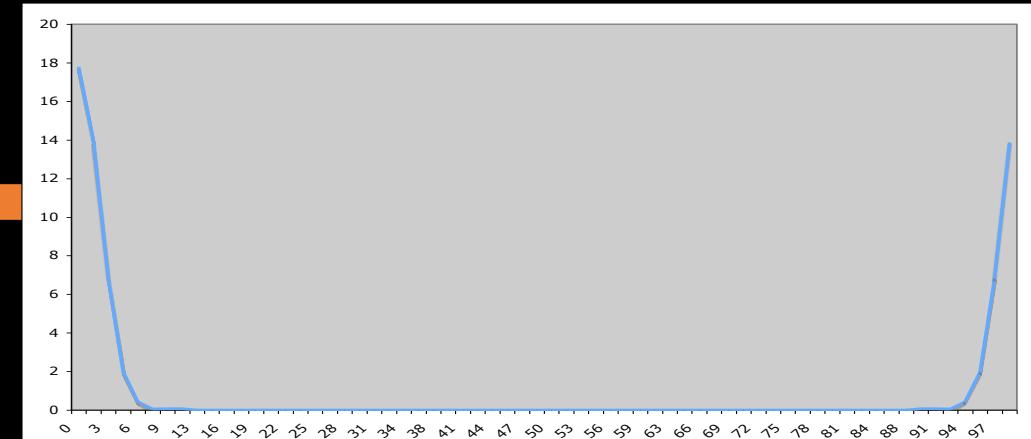
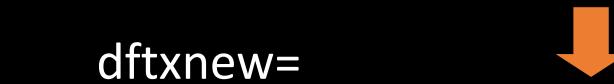
$dftx = fft(x)$



$x_{new} = ifft(dftx_{new})$

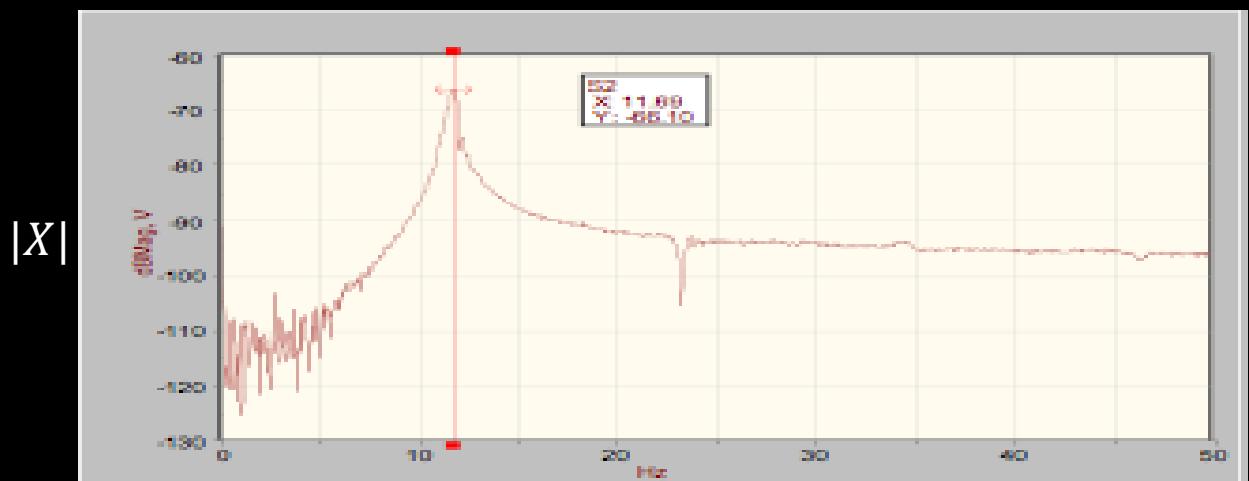
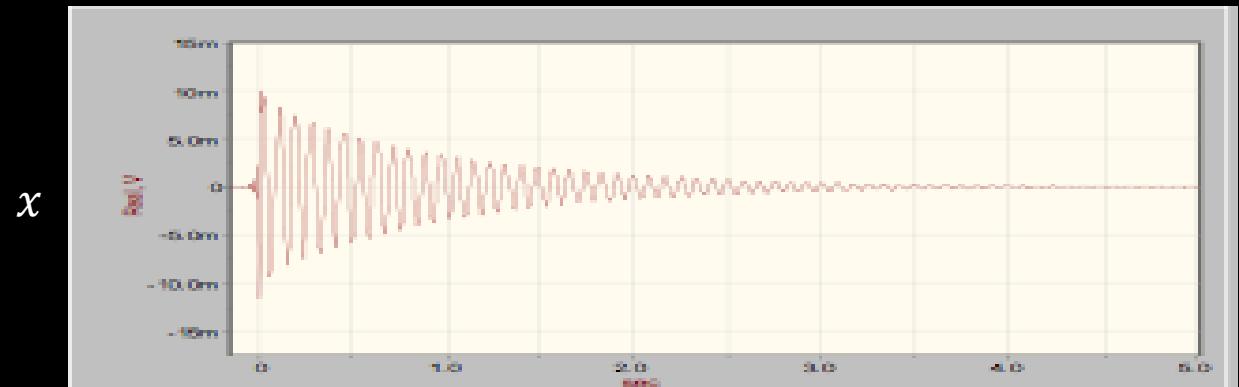


$dftx_{new} =$



Determining Vibration Frequency

- Hammer test on 2M vibrations lab beam. Measured free decay with accelerometer

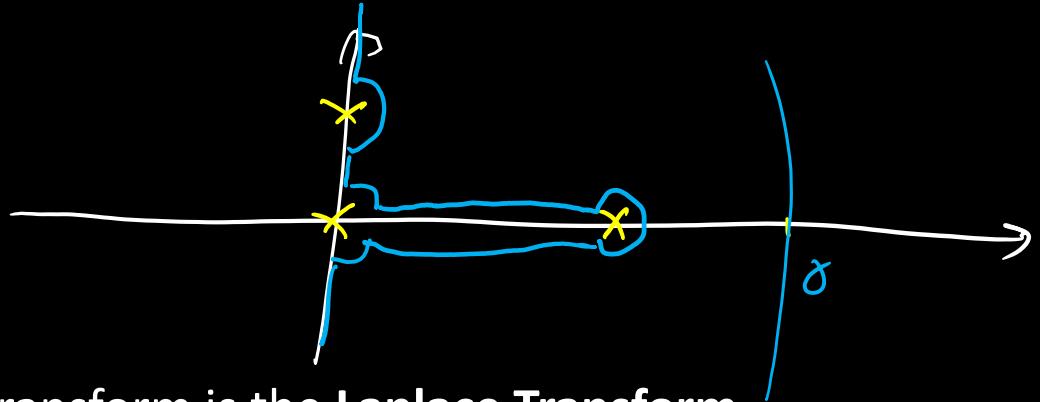


Worked Example 38: Image Compression

- Demonstrated in Lecture

Chapter 4: Transforms | Laplace Transform

Laplace Transforms



- Another useful ‘transform’, related to the Fourier transform is the **Laplace Transform**
- The Laplace Transform $F(s)$ of a function $f(t)$ is defined as:
$$F(s) = \int_0^{\infty} f(t)e^{-st} dt$$
- Again, this represents is an entirely equivalent way of representing the function $f(t)$. We have transformed the function $f(t)$ which is a function of the variable t into a function $F(s)$ which is a function of the variable s .
- We can, just like the Fourier transform, convert back into ‘t-space’, using the following definition
$$f(t) = \frac{1}{2\pi i} \int_{\gamma-i\infty}^{\gamma+i\infty} F(s)e^{st} ds$$
- Where γ is a real number such that the contour of the path integral (a ‘Bromwich’ contour) is in the region of convergence of $F(s)$ actually don’t even worry about this. It is a (complicated) mathematical detail and in practice we never use this expression for the inverse.

Properties of Laplace Transforms

- As with Fourier Transforms, there are many useful properties that can be proved:

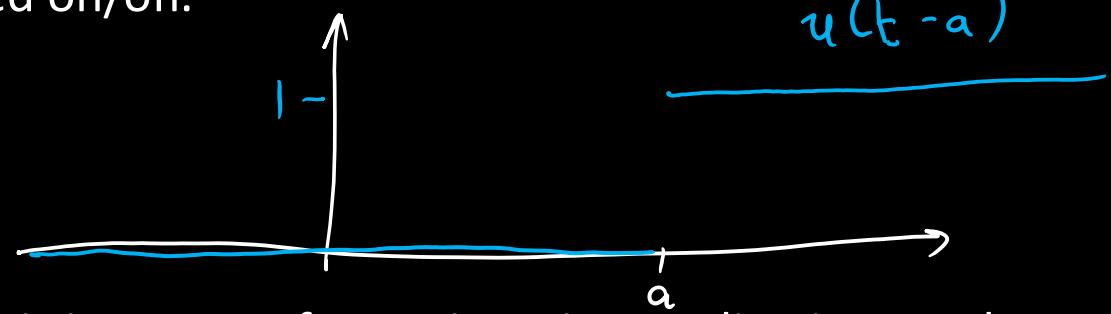
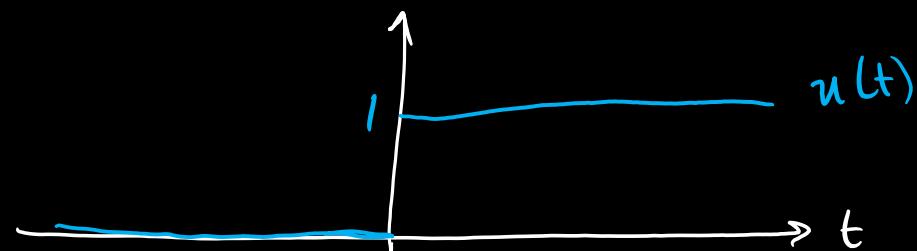
	Function/Signal	\longleftrightarrow	Transform
Linearity:	$af(t) + bg(t)$		$aF(s) + bG(s)$
Shift in time:	$f(t - a)u(t - a)$		$e^{-as}F(s)$
Shift in ‘frequency’:	$e^{at}f(t)$		$F(s - a)$
Differentiation:	$\frac{df(t)}{dt}$		$sF(s) - f(0)$
Scaling:	$f(at)$		$\frac{1}{ a }F\left(\frac{s}{a}\right)$

The Heaviside Function and Dirac Delta: Engineering Relevance

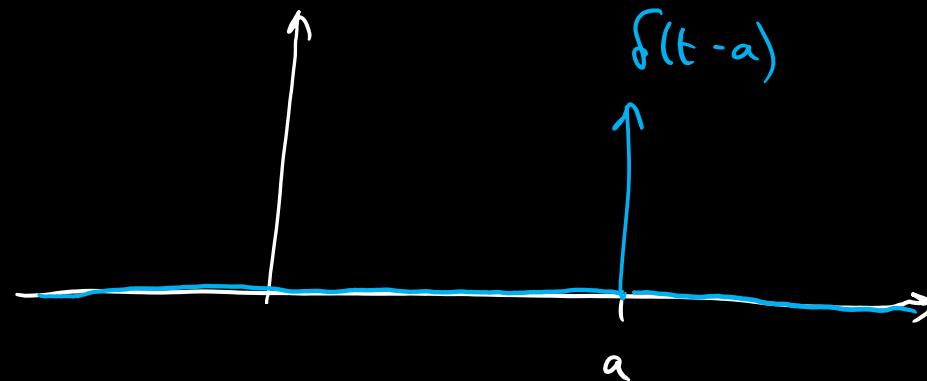
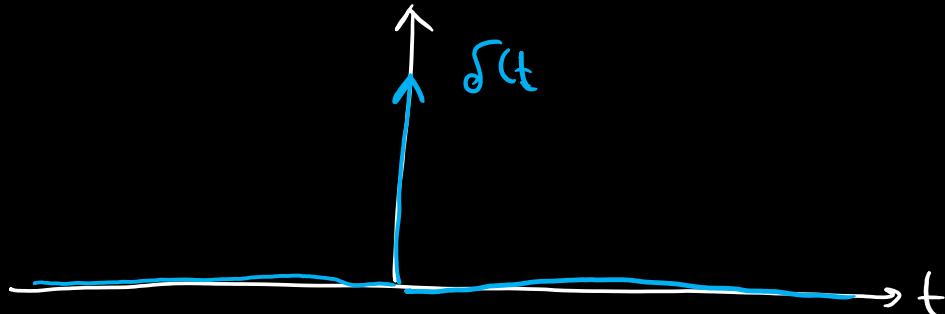
$$\Theta(t)$$

$$H(t)$$

- The Heaviside Function $u(t)$ is a unit step function. It is important for engineering application as it can be used to model something being switched on/off.

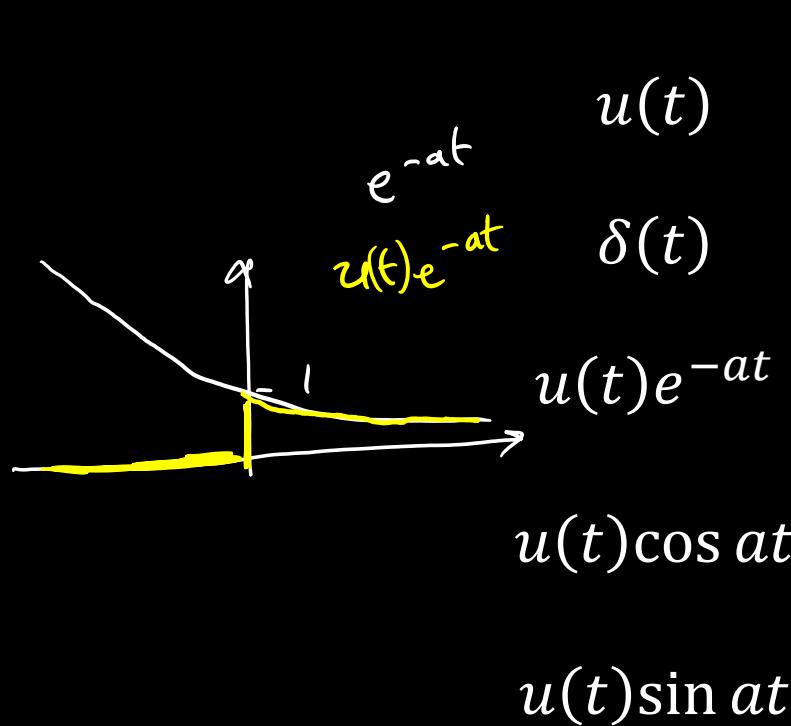


- The Dirac Delta represents a ‘perfect’ impulse. It is important for engineering applications such as aeroplanes landing, a cricket/tennis shot, hammer strikes, etc.



Common Laplace Transform Pairs

Function/Signal	Transform
$u(t)$	$\frac{1}{s}$
e^{-at}	1
$u(t)e^{-at}$	$\frac{1}{s+a}$
$u(t)\cos at$	$\frac{s}{s^2 + a^2}$
$u(t)\sin at$	$\frac{a}{s^2 + a^2}$



Many more available in the literature and your formula book (i.e. they are given in an exam!)

Proof of differentiation property

Worked Example 39

$$G(s) = \int_0^\infty \frac{df}{dt} e^{-st} dt$$

v' u

$$= \left[f(t) e^{-st} \right]_0^\infty - \int_0^\infty f(t)(-s)e^{-st} dt$$

u v *v u'*

s=a+ib ; assume $f(t) \rightarrow 0$ at $t \rightarrow \infty$

$$= 0 - f(0)e^0 + s \underbrace{\int_0^\infty f(t) e^{-st} dt}_{F(s)}$$

$$= sF(s) - f(0)$$

$$g(t) = \frac{df}{dt}$$

$$f(t) \rightarrow F(s)$$

$$F(s) = \mathcal{L}\{f(t)\}$$

$$f(t) = \mathcal{L}^{-1}\{F(s)\}$$

$$P(s) = \mathcal{L}\left\{ \frac{d^2f}{dt^2} \right\}$$

$$p(t) = \frac{d^2f}{dt^2} = \frac{dg}{dt} \quad \leftarrow \text{substitution}$$

$$g(0) = \frac{df}{dt}(0) = f'(0)$$

$$G(s) = sF(s) - f(0)$$

$$P(s) = \mathcal{L}\left\{ \frac{dg}{dt} \right\}$$

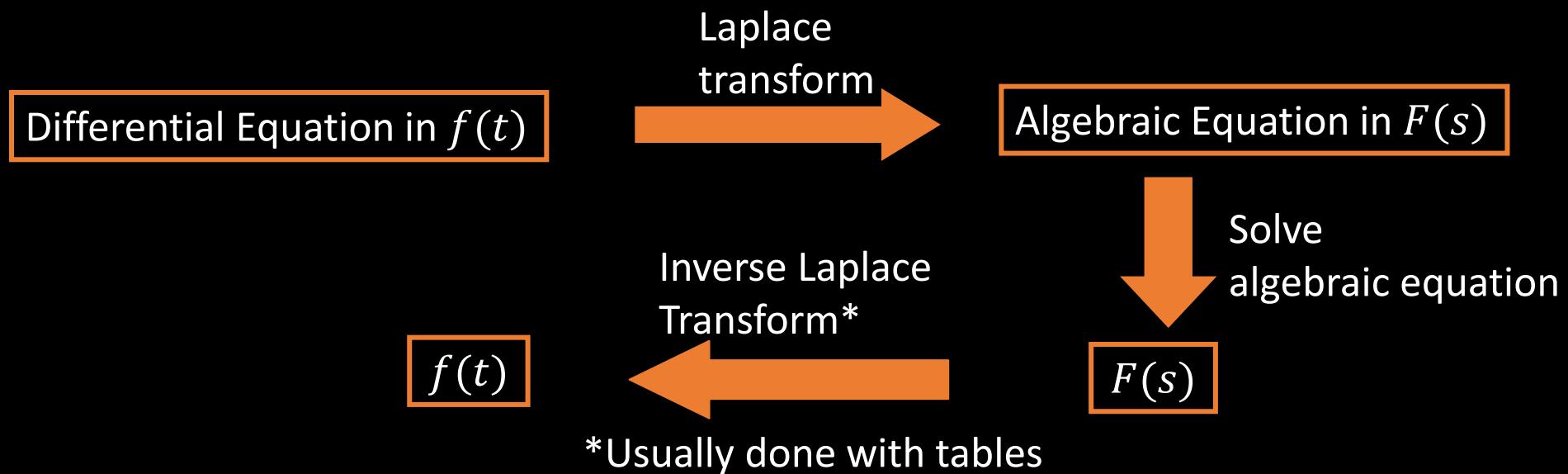
$$= sG(s) - g(0)$$

$$= s(sF(s) - f(0) - f'(0))$$

$$= s^2 F(s) - sf(0) - f'(0)$$

Laplace Transforms: Application to ODEs

- Laplace transforms are useful for solving ordinary differential equations. This is best shown by example, but the process is quite simple:



- Note: In Laplace transform methods, the initial conditions enter at the first stage, that is, when the transform is calculated.

$$\frac{df(t)}{dt} \rightarrow sF(s) - f(0)$$

Worked Example 40

$$\text{try } f = e^{-1t}$$

$$\frac{df(t)}{dt} + 5f = 0, \quad f(0) = 2$$

Laplace transform

$$sf(s) - f(0)$$

$$sF(s) - 2 + 5F(s) = 0$$

Solve algebraic equation

Inverse Laplace Transform

$$f(t) = 2u(t)e^{-5t}$$

$$F(s) = \frac{2}{s + 5}$$

Q: solve $\frac{d^2f}{dt^2} + 2\frac{df}{dt} + 6f = 0$ where $f(0) = 1$ & $f'(0) = 0$ with the Laplace Transform method

Worked Example 41

Step 1) convert ODE $\left\{ \frac{d^2f}{dt^2} + 2\frac{df}{dt} + 6f \right\} = \left\{ 0 \right\}$
to Laplace domain

$$s^2F(s) - sf(0) - f'(0) + 2(sF(s) - f(0)) + 6F(s) = 0$$

$$s^2F(s) - s + 2sF(s) - 2 + 6F(s) = 0$$

Step 2) solve in the Laplace Domain

$$F(s)(s^2 + 2s + 6) = s + 2$$

$$F(s) = \frac{s+2}{s^2 + 2s + 6}$$

step 3) go back to time domain

$$f(t) = \mathcal{L}^{-1} \left\{ F(s) \right\} = \mathcal{L}^{-1} \left\{ \frac{(s+2)}{s^2 + 2s + 6} \right\}$$

either denominator factors \Rightarrow partial fractions, or complete the square

$$\begin{aligned} f(t) &= \mathcal{L}^{-1} \left\{ \frac{(s+2)}{(s+1)^2 + 5} \right\} \\ &= \mathcal{L}^{-1} \left\{ \frac{(s+1)}{(s+1)^2 + (\sqrt{5})^2} + \frac{(2-1)}{(s+1)^2 + (\sqrt{5})^2} \right\} \end{aligned}$$

Shift in freq where ' $a = -1$ ' \Rightarrow have e^{-t} in front

$$\begin{aligned} f(t) &= e^{-t} \left\{ \frac{s}{s^2 + (\sqrt{5})^2} + \frac{1}{\sqrt{5}} \frac{\sqrt{5}}{(s^2 + (\sqrt{5})^2)} \right\} \\ &= e^{-t} \left(\cos \sqrt{5}t + \frac{1}{\sqrt{5}} \sin \sqrt{5}t \right) u(t) \\ &= e^{-t} \left(\cos \sqrt{5}t + \frac{1}{\sqrt{5}} \sin \sqrt{5}t \right) \quad \text{where } t > 0 \end{aligned}$$

Spare slides

Progress Test to Exam

