

# Applied Mathematics 205

## Unit III: Numerical Calculus

Lecturer: Dr. David Knezevic

# Unit III: Numerical Calculus

## Chapter III.2: Numerical Integration and Differentiation

# Numerical Integration

# Quadrature

Suppose we want to evaluate the integral  $I(f) \equiv \int_a^b f(x)dx$

We can proceed as follows:

1. Approximate  $f$  using a polynomial interpolant  $p_n$
2. Evaluate  $Q_n(f) \equiv \int_a^b p_n(x)dx$ , since we know how to integrate polynomials

$Q_n(f)$  provides a **quadrature** formula, and we should have  $Q_n(f) \approx I(f)$

A quadrature rule based on an interpolant  $p_n$  at  $n + 1$  **equally spaced points** in  $[a, b]$  is known as **Newton-Cotes** formula of order  $n$

## Newton-Cotes Quadrature

Let  $x_k = a + kh, k = 0, 1, \dots, n$ , where  $h = (b - a)/n$

We write the interpolant of  $f$  in Lagrange form as

$$p_n(x) = \sum_{k=0}^n f(x_k)L_k(x), \quad \text{where} \quad L_k(x) \equiv \prod_{i=0, i \neq k}^n \frac{x - x_i}{x_k - x_i}$$

Then

$$Q_n(f) = \int_a^b p_n(x)dx = \sum_{k=0}^n f(x_k) \int_a^b L_k(x)dx = \sum_{k=0}^n w_k f(x_k)$$

where  $w_k \equiv \int_a^b L_k(x)dx \in \mathbb{R}$  is the  $k^{\text{th}}$  quadrature weight

# Newton-Cotes Quadrature

Note that quadrature weights **do not depend** on  $f$ , hence can be precomputed and stored

$n = 1 \implies$  Trapezoid rule ([See lecture](#))

$n = 2 \implies Q_2(f) = \frac{b-a}{6} [f(a) + 4f(\frac{a+b}{2}) + f(b)]$  Simpson rule

We can also develop higher-order Newton-Cotes formulae in the same way

## Error Estimates

Let  $E_n(f) \equiv I(f) - Q_n(f)$

Then

$$\begin{aligned} E_n(f) &= \int_a^b f(x) dx - \sum_{k=0}^n w_k f(x_k) \\ &= \int_a^b f(x) dx - \sum_{k=0}^n \left( \int_a^b L_k(x) dx \right) f(x_k) \\ &= \int_a^b f(x) dx - \int_a^b \left( \sum_{k=0}^n L_k(x) f(x_k) \right) dx \\ &= \int_a^b f(x) dx - \int_a^b p_n(x) dx \\ &= \int_a^b (f(x) - p_n(x)) dx \end{aligned}$$

And we have an expression for  $f(x) - p_n(x)$

## Error Estimates

Recall from I.2

$$f(x) - p_n(x) = \frac{f^{n+1}(\theta)}{(n+1)!} (x - x_0) \cdots (x - x_n)$$

Hence, it follows from  $|E_n(f)| \leq \int_a^b |f(x) - p_n(x)| dx$  that:

$$|E_n(f)| \leq \frac{M_{n+1}}{(n+1)!} \int_a^b |(x - x_0)(x - x_1) \cdots (x - x_n)| dx$$

where  $M_{n+1} = \max_{\theta \in [a,b]} |f^{n+1}(\theta)|$



# Error Estimates

See lecture: Trapezoid rule error bound

$$|E_1(f)| \leq \frac{(b-a)^3}{12} M_2$$

The bound for  $E_n$  depends directly on the integrand  $f$  (via  $M_{n+1}$ )

Just like in Unit I with the Lebesgue constant, it is informative to be able to compare quadrature rules independently of the integrand

## Error Estimates: Another Perspective

**Theorem:** If  $Q_n$  integrates polynomials of degree  $n$  exactly, then  $\exists C_n > 0$  such that  $|E_n(f)| \leq C_n \min_{p \in \mathbb{P}_n} \|f - p\|_\infty$

**Proof:** For  $p \in \mathbb{P}_n$ , we have

$$\begin{aligned} |I(f) - Q_n(f)| &\leq |I(f) - I(p)| + |I(p) - Q_n(f)| \\ &= |I(f - p)| + |Q_n(f - p)| \\ &\leq \int_a^b dx \|f - p\|_\infty + \left( \sum_{k=0}^n |w_k| \right) \|f - p\|_\infty \\ &\equiv C_n \|f - p\|_\infty \end{aligned}$$

where

$$C_n \equiv b - a + \sum_{k=0}^n |w_k|$$

## Error Estimates

Hence a convenient way to compare accuracy of quadrature rules is to **compare the polynomial degree they integrate exactly**

Newton-Cotes of order  $n$  is based on polynomial interpolation, hence in general integrates polynomials of degree  $n$  exactly<sup>1</sup>

---

<sup>1</sup>Also follows from the  $M_{n+1}$  term in the error bound

## Runge's Phenomenon Again...

But Newton-Cotes formulae are based on interpolation at equally spaced points

Hence they're susceptible to **Runge's phenomenon**, and we expect them to be inaccurate for large  $n$

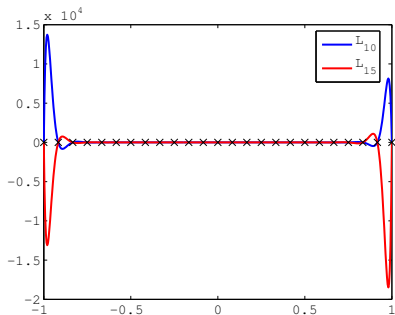
**Question:** How does this show up in our bound

$$|E_n(f)| \leq C_n \min_{p \in \mathbb{P}_n} \|f - p\|_\infty \quad ?$$

## Runge Phenomenon Again...

**Answer:** In the constant  $C_n$

Recall that  $C_n \equiv b - a + \sum_{k=0}^n |w_k|$ , and that  $w_k \equiv \int_a^b L_k(x) dx$



If the  $L_k$  “blow up” due to equally spaced points, then  $C_n$  can also “blow up”

## Runge Phenomenon Again...

In fact, we know that  $\sum_{k=0}^n w_k = b - a$ , why?

This tells us that if all the  $w_k$  are positive, then

$$C_n = b - a + \sum_{k=0}^n |w_k| = b - a + \sum_{k=0}^n w_k = 2(b - a)$$

Hence **positive weights**  $\implies C_n$  is a constant, independent of  $n$   
and hence  $Q_n(f) \rightarrow I(f)$  as  $n \rightarrow \infty$

## Runge Phenomenon Again...

But with Newton-Cotes, quadrature weights become **negative** for  $n > 8$  (e.g. in example above  $L_{15}(x)$  would clearly yield  $w_{15} < 0$ )

**Key point:** Newton-Cotes is not useful for large  $n$

However, there are two natural ways to get quadrature rules that **converge** as  $n \rightarrow \infty$ :

- ▶ Integrate piecewise polynomial interpolant
- ▶ Don't use equally spaced interpolation points

We consider piecewise polynomial-based quadrature rules first

# Composite Quadrature Rules

Integrating piecewise polynomial interpolant  $\implies$  **composite quadrature rule**

Suppose we divide  $[a, b]$  into  $m$  subintervals, each of width  $h = (b - a)/m$ , and  $x_i = a + ih$ ,  $i = 0, 1, \dots, m$

Then we have:

$$I(f) = \int_a^b f(x)dx = \sum_{i=1}^m \int_{x_{i-1}}^{x_i} f(x)dx$$



# Composite Trapezoid Rule

**Composite trapezoid rule:** Apply trapezoid rule to each interval, i.e.  $\int_{x_{i-1}}^{x_i} f(x)dx \approx \frac{1}{2}h[f(x_{i-1}) + f(x_i)]$

Hence,

$$\begin{aligned} Q_{1,h}(f) &\equiv \sum_{i=1}^m \frac{1}{2}h[f(x_{i-1}) + f(x_i)] \\ &= h \left[ \frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{m-1}) + \frac{1}{2}f(x_m) \right] \end{aligned}$$

# Composite Trapezoid Rule

Composite trapezoid rule error analysis:

$$\begin{aligned} E_{1,h}(f) &\equiv I(f) - Q_{1,h}(f) \\ &= \sum_{i=1}^m \left[ \int_{x_{i-1}}^{x_i} f(x) dx - \frac{1}{2} h [f(x_{i-1}) + f(x_i)] \right] \end{aligned}$$

Hence,

$$\begin{aligned} |E_{1,h}(f)| &\leq \sum_{i=1}^m \left| \int_{x_{i-1}}^{x_i} f(x) dx - \frac{1}{2} h [f(x_{i-1}) + f(x_i)] \right| \\ &\leq \frac{h^3}{12} \sum_{i=1}^m \max_{\theta \in [x_{i-1}, x_i]} |f''(\theta)| \\ &\leq \frac{h^3}{12} m \|f''\|_{\infty} \\ &= \frac{h^2}{12} (b-a) \|f''\|_{\infty} \end{aligned}$$

## Composite Simpson Rule

We can obtain the composite Simpson rule in the same way

Suppose that  $[a, b]$  is divided into  $2m$  intervals by the points  $x_i = a + ih$ ,  $i = 0, 1, \dots, 2m$ ,  $h = (b - a)/2m$

Applying Simpson rule on each  $[x_{2i-2}, x_{2i}]$ ,  $i = 1, \dots, m$  yields<sup>2</sup>

$$Q_{2,h}(f) \equiv \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + 2f(x_{2m-2}) + 4f(x_{2m-1}) + f(x_{2m})]$$

---

<sup>2</sup>Here each interval is of width  $2h$

# Adaptive Quadrature

Composite quadrature rules are very flexible, e.g. we need not choose equally sized intervals

Intuitively, we should use smaller intervals where  $f$  varies rapidly, and larger intervals where  $f$  varies slowly

This can be achieved by adaptive quadrature:

1. Initialize to  $m = 1$  (one interval)
2. On each interval, evaluate quadrature rule and estimate quadrature error
3. If error estimate  $>$  TOL on interval  $i$ , subdivide to get two smaller intervals and return to step 2.

**Question:** How can we estimate the quadrature error on an interval?

## Adaptive Quadrature

One straightforward way to estimate quadrature error on interval  $i$  is to compare to a “more refined” result for interval  $i$

Let  $I^i(f)$  and  $Q_h^i(f)$  denote the exact integral and quadrature approx. on interval  $i$ , respectively

Let  $\hat{Q}_h^i(f)$  denote a more refined quadrature approx. on interval  $i$ , e.g. obtained by subdividing interval  $i$

Then for the error on interval  $i$ , we have:

$$|I^i(f) - Q_h^i(f)| \leq |I^i(f) - \hat{Q}_h^i(f)| + |\hat{Q}_h^i(f) - Q_h^i(f)|$$

Then, we suppose we can neglect  $|I^i(f) - \hat{Q}_h^i(f)|$  so that we use  $|\hat{Q}_h^i(f) - Q_h^i(f)|$  as a computable estimator for  $|I^i(f) - Q_h^i(f)|$

## Adaptive Quadrature

Matlab's quad function implements an adaptive Simpson rule

```
>> help quad
```

```
QUAD Numerically evaluate integral, adaptive  
Simpson quadrature. Q = QUAD(FUN,A,B) tries to  
approximate the integral of scalar-valued function  
FUN from A to B to within an error of 1.e-6 using  
recursive adaptive Simpson quadrature.
```

Next we consider the second approach to developing more accurate quadrature rules: **unevenly spaced quadrature points**

# Gauss Quadrature

Recall that we can compare accuracy of quadrature rules based on the polynomial degree that is integrated exactly

So far, we haven't been very creative with our choice of quadrature points: Newton-Cotes  $\iff$  equally spaced

More accurate quadrature rules can be derived by choosing the  $x_i$  to maximize poly. degree that is integrated exactly

Resulting family of quadrature rules is called Gauss quadrature

# Gauss Quadrature

Intuitively, with  $n + 1$  quadrature points and  $n + 1$  quadrature weights we have  $2n + 2$  parameters to choose

Hence we might hope to exactly integrate polynomials with  $2n + 2$  parameters, i.e. of degree  $2n + 1$

It can be shown that this is possible  $\implies$  Gauss quadrature (proof is outside the scope of AM205)

Gauss quadrature is again based on integrating a polynomial interpolant, but we choose a specific set of interpolation points:

Gauss quadrature points are roots of a Legendre polynomial<sup>3</sup>

---

<sup>3</sup>Adrien-Marie Legendre, 1752-1833, French mathematician



# Gauss Quadrature

We will not discuss Legendre polynomials in detail...

Briefly, Legendre polynomials  $\{P_0, P_1, \dots, P_n\}$  form an **orthogonal basis** for  $\mathbb{P}_n$  in the “ $L_2$  inner-product”

$$\int_{-1}^1 P_m(x)P_n(x)dx = \begin{cases} \frac{2}{2n+1}, & m = n \\ 0, & m \neq n \end{cases}$$

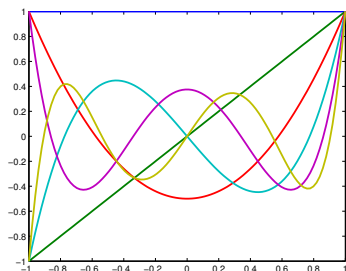
## Gauss Quadrature

As with Chebyshev polys, Legendre polys satisfy a 3-term recurrence relation

$$P_0(x) = 1$$

$$P_1(x) = x$$

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$



The first six Legendre polynomials

## Gauss Quadrature

Hence, can find the roots of  $P_n(x)$  and derive the  $n$ -point Gauss quadrature rule by **integrating the Lagrange interpolant**

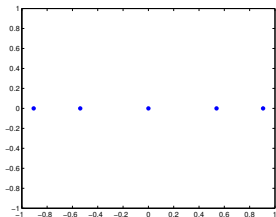
Gauss quadrature rules have been extensively tabulated for  $x \in [-1, 1]$ :

Number of points	Quadrature points	Quadrature weights
1	0	2
2	$-1/\sqrt{3}, 1/\sqrt{3}$	1, 1
3	$-\sqrt{3/5}, 0, \sqrt{3/5}$	5/9, 8/9, 5/9
$\vdots$	$\vdots$	$\vdots$

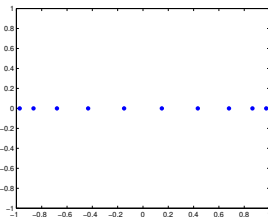
**Key point:** Gauss quadrature weights are always positive, hence Gauss quadrature converges as  $n \rightarrow \infty$

# Gauss Quadrature Points

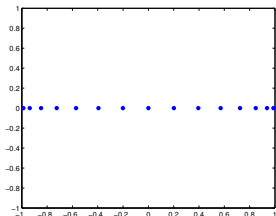
Points cluster toward  $\pm 1$ , prevents Runge's phenomenon!



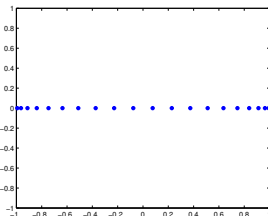
5 points



10 points



15 points



20 points

## Gauss Quadrature

In Matlab, `quadl` performs adaptive, composite Lobatto quadrature

(Lobatto quadrature is closely related to Gauss quadrature, difference is that we ensure that  $-1$  and  $1$  are quadrature points)

From `help quadl`:

```
“    QUAD may be most efficient for low accuracies  
with nonsmooth integrands.
```

```
    QUADL may be more efficient than QUAD at higher  
accuracies with smooth integrands. ”
```

**Take-away message:** Gauss/Lobatto quadrature is usually more efficient for **smooth integrands**

# Numerical Differentiation

# Finite Difference Approximations

Given a function  $f : \mathbb{R} \rightarrow \mathbb{R}$

We want to approximate derivatives of  $f$  via simple expressions involving samples of  $f$

As we saw in Unit 0, convenient starting point is [Taylor's theorem](#):

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \dots$$

# Finite Difference Approximations

Solving for  $f'(x)$  we get the **forward difference formula**

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

Here we neglected an  $O(h)$  term



# Finite Difference Approximations

Similarly, we have the Taylor series

$$f(x - h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{6}h^3 + \dots$$

which yields the **backward difference formula**

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

Again we neglected an  $O(h)$  term

# Finite Difference Approximations

Subtracting Taylor expansion for  $f(x - h)$  from expansion for  $f(x + h)$  gives the **centered difference formula**

$$\begin{aligned} f'(x) &= \frac{f(x + h) - f(x - h)}{2h} - \frac{f'''(x)}{6}h^2 + \dots \\ &\approx \frac{f(x + h) - f(x - h)}{2h} \end{aligned}$$

In this case we neglected an  $O(h^2)$  term

# Finite Difference Approximations

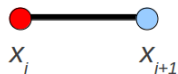
Adding Taylor expansion for  $f(x - h)$  and expansion for  $f(x + h)$  gives the **centered difference formula** for the second derivative

$$\begin{aligned} f''(x) &= \frac{f(x + h) - 2f(x) + f(x - h)}{h^2} - \frac{f^{(4)}(x)}{12}h^2 + \dots \\ &\approx \frac{f(x + h) - 2f(x) + f(x - h)}{h^2} \end{aligned}$$

Again we neglected an  $O(h^2)$  term

# Finite Difference Stencils

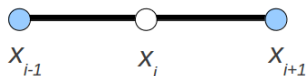
Forward diff.



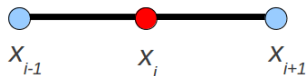
Backward diff.



Centered diff.  
1<sup>st</sup> derivative



Centered diff.  
2<sup>nd</sup> derivative



# Finite Difference Approximations

We can use Taylor expansion to derive approximations with higher order accuracy, or for higher derivatives

This involves developing F.D. formulae with “wider stencils,” i.e. based on samples at  $x \pm 2h, x \pm 3h, \dots$

An alternative approach that yields equivalent results is to **differentiate the interpolant**

# Finite Difference Approximations

Linear interpolant at  $\{(x_0, f(x_0)), (x_0 + h, f(x_0 + h))\}$  is

$$p_1(x) = f(x_0) \frac{x_0 + h - x}{h} + f(x_0 + h) \frac{x - x_0}{h}$$

Differentiating  $p_1$  gives

$$p_1'(x) = \frac{f(x_0 + h) - f(x_0)}{h},$$

which is the **forward difference formula**

## Finite Difference Approximations

Similarly, quadratic interpolant,  $p_2$ , from interpolation points  $\{x_0, x_1, x_2\}$  yields the centered difference formula for  $f'$  at  $x_1$ :

- ▶ Differentiate  $p_2(x)$  to get a linear polynomial,  $p_2'(x)$
- ▶ Evaluate  $p_2'(x_1)$  to get centered difference formula for  $f'$

Also,  $p_2''(x)$  gives the centered difference formula for  $f''$

Note: Differentiating the interpolant can be a convenient way to derive high order F.D. formulae

# Finite Difference Approximations

So far we have talked about finite difference formulae to approximate  $f'(x_i)$  at some specific point  $x_i$

**Question:** What if we want to approximate  $f'(x)$  on an interval  $x \in [a, b]$ ?

**Answer:** We need to simultaneously approximate  $f'(x_i)$  for  $x_i$ ,  $i = 1, \dots, n$



## Differentiation Matrices

We need a map from the vector  $F \equiv [f(x_1), f(x_2), \dots, f(x_n)] \in \mathbb{R}^n$  to the vector of derivatives  $F' \equiv [f'(x_1), f'(x_2), \dots, f'(x_n)] \in \mathbb{R}^n$

Let  $\tilde{F}'$  denote our finite difference approximation to the vector of derivatives, i.e.  $\tilde{F}' \approx F'$

Differentiation is a linear operator<sup>4</sup>, hence we expect the map from  $F$  to  $\tilde{F}'$  to be an  $n \times n$  matrix

This is indeed the case, and this map is a **differentiation matrix**,  $D$

---

<sup>4</sup>Since  $(\alpha f + \beta g)' = \alpha f' + \beta g'$

## Differentiation Matrices

Row  $i$  of  $D$  corresponds to the finite difference formula for  $f'(x_i)$ , since then  $D_{(i,:)}F \approx f'(x_i)$

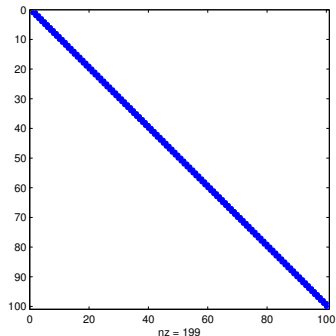
e.g. for forward difference approx. of  $f'$ , non-zero entries of row  $i$  are

$$D_{ii} = -\frac{1}{h}, \quad D_{i,i+1} = \frac{1}{h}$$

This is a **sparse matrix** with two non-zero diagonals

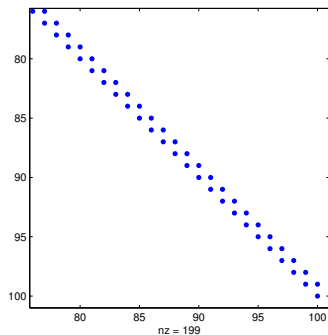
## Differentiation Matrices

```
n=100; h = 1/(n-1);  
e = 1/h*ones(n,1);  
D = spdiags([-e,e],[0,1],n,n);  
spy(D);
```



# Differentiation Matrices

But what about the last row?

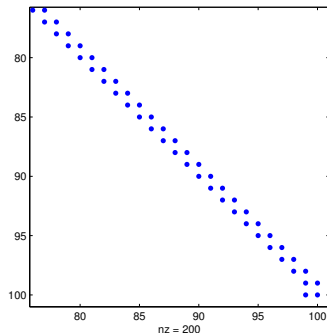


$D_{n,n+1} = \frac{1}{h}$  is ignored!

## Differentiation Matrices

We can use the backward difference formula (which has the same order of accuracy) for row  $n$  instead

$$D_{n,n-1} = -\frac{1}{h}, \quad D_{nn} = \frac{1}{h}$$



[Matlab demo](#): Differentiation matrices