AM 205: lecture 10

- Last time: Singular Value Decomposition, Principal Component Analysis
- Today: Numerical integration and differentiation

Quadrature

Suppose we want to evaluate the integral $I(f) \equiv \int_a^b f(x)\mathrm{d}x$

We can proceed as follows:

1. Approximate f using a polynomial interpolant p_n
2. Evaluate $Q_n(f) \equiv \int_a^b p_n(x)\mathrm{d}x$, 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 $x_k = a + kh$, $k = 0, 1, \ldots, n$, 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

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

where $w_k \equiv \int_a^b L_k(x) \mathrm{d}x \in \mathbb{R}$ is the kth 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)

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

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

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

Error Estimates

Recall from I.2

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

Hence

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

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 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

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

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[1]

[1] 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

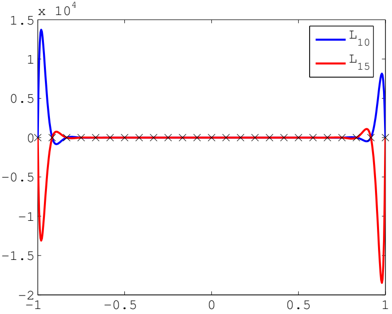
Question: How does this show up in our bound

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

Runge Phenomenon Again...

Answer: In the constant C_n

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



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 $Q_n(f) \to I(f)$ as $n \to \infty$

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 $n \to \infty$:

- 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 $\Longrightarrow$ composite quadrature rule

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

Then we have:

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

Composite Trapezoid Rule

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

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:

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

Hence,

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

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 $x_i = a + ih$, $i = 0, 1, \ldots, 2m$, $h = (b - a)/2m$

Applying Simpson rule on each interval[2] $[x_{2i-2}, x_{2i}]$, $i = 1, \ldots, m$ yields

$$\begin{aligned} Q_{2,h}(f) &\equiv \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots \\ &\qquad +2f(x_{2m-2}) + 4f(x_{2m-1}) + f(x_{2m})] \end{aligned}$$

[2] Interval 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 approximation on interval i, respectively

Let $\hat{Q}_h^i(f)$ denote a more refined quadrature approximation 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

Python and MATLAB both have `quad` functions, although with different implementations. 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 $\Longleftrightarrow$ 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 integrate a poly. 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)

Again the idea is to integrate a polynomial interpolant, but we choose a specific set of interpolation points:

Gauss quad. points are roots of a Legendre polynomial[3]

[3]Adrien-Marie Legendre, 1752-1833, French mathematician

Gauss Quadrature

We will not discuss Legendre polynomials in detail.

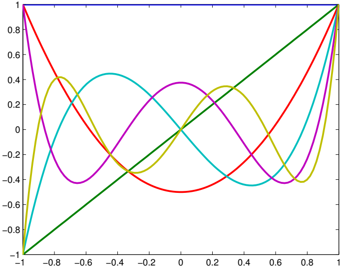
Briefly, Legendre polynomials $\{P_0, P_1, \ldots, P_n\}$ form an orthogonal basis for $\mathbb{P}_n$ in the "L^2 inner-product"

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

Gauss Quadrature

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

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ (n+1)P_{n+1}(x) &= (2n+1)xP_n(x) - nP_{n-1}(x) \end{aligned}$$



The first six Legendre polynomials

Gauss Quadrature

Hence, can find the roots of $P_n(x)$ and derive the n-point Gauss quad. rule in the same way as for Newton–Cotes:

Integrate the Lagrange interpolant!

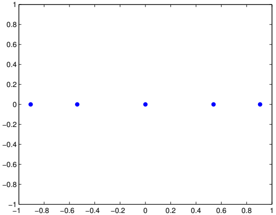
Gauss quadrature rules have been extensively tabulated for $x \in [-1, 1]$:

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

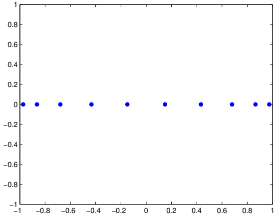
Key point: Gauss quadrature weights are always positive, hence Gauss quadrature converges as $n \to \infty$!

Gauss Quadrature Points

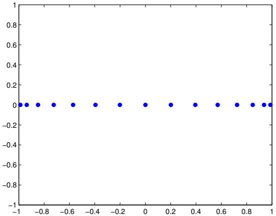
Points cluster toward ± 1, prevents Runge's phenomenon!



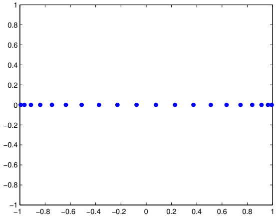
5 points



10 points



15 points



20 points

Generalization

Suppose we wish to evaluate exactly integrals of the form

$$\int_{-1}^{1} w(x) f(x)\, dx.$$

Then we can calculate quadrature based on polynomials u_k that are orthogonal with respect to the inner product

$$\langle u_j, u_k \rangle \int_{-1}^{1} w(x) u_j(x) u_k(x) dx.$$

A typical example case is

$$w(x) = \frac{1}{\sqrt{1 - x^2}}.$$

Orthogonality relation is then

$$\langle u_j, u_k \rangle \int_{-1}^{1} w(x) u_j(x) u_k(x) dx.$$

Try the Chebyshev polynomials $u_j(x) = T_j(x) = \cos(j \cos^{-1} x)$.

Generalization

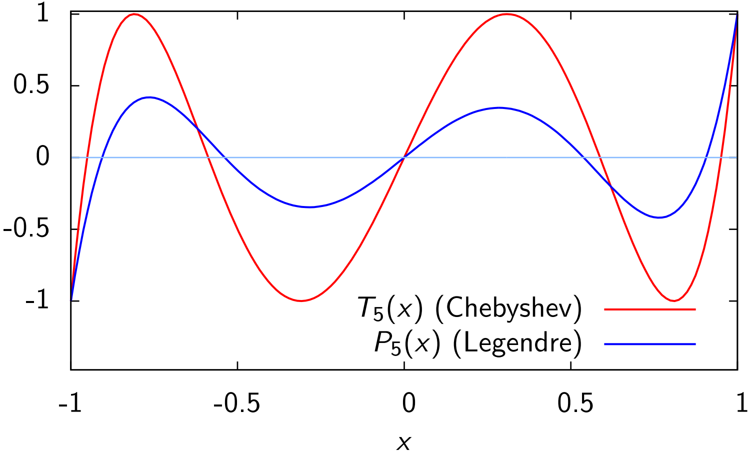
Using the substitution $x = \cos\theta$,

$$\begin{aligned} \langle T_j, T_k \rangle &= \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} \cos(j\cos^{-1}x)\cos(k\cos^{-1}x)dx && (1) \\ &= \int_0^{\pi} \frac{1}{\sqrt{1-\cos^2\theta}} \cos j\theta \cos k\theta (\sin\theta\, d\theta) && (2) \\ &= \int_0^{\pi} \cos j\theta \cos k\theta\, d\theta. && (3) \end{aligned}$$

Using the Fourier orthogonality relations, $\langle T_j, T_k \rangle = 0$ for $j \neq k$, so the Chebyshev polynomials are orthogonal with respect to this weight function.

Hence the roots of the Chebyshev polynomials can be used to construct a quadrature formula for this $w(x)$. This is just one example of many possible generalizations to Gauss quadrature.

Legendre/Chebyshev comparison



Chebyshev roots are closer to the ends—better sampling of the function near ± 1, as expected based on $w(x)$.

Gauss Quadrature

Python's `quad` function makes use of Clenshaw–Curtis quadrature, based on Chebyshev polynomials.

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