Unit IV

Numerical Integration and Differentiation

Numerical integration and differentiation

- quadrature
  - classical formulas for equally spaced nodes
  - improper integrals
  - Gaussian quadrature and orthogonal polynomials
- differentiation of functions
  - finite difference methods
  - smoothing methods for noisy data

Numerical integration

- construct a polynomial interpolation for the function f(x)
- this provides an easy integration to approximate the definite integral

$$\int_a^b f(x)dx$$

- this problem is actually a special case of integration of an ordinary differential equation

$$\frac{dy}{dx} = f(x)$$

  - i.e. find y(b) subject to the boundary condition y(a) = 0
- direct methods for numerical integration of functions are called *quadrature*

Plan of attack

- to integrate f(x) from a to b ...
- add up values of the integrand at selected abscissas within the range of integration
- goal: obtain an accurate value of the integral with the smallest number of evaluations of the integrand
- as with polynomial interpolation we can choose methods of increasingly high order but ....

- ... higher order does NOT necessarily imply higher precision

How?

- n abscissa points (*nodes*), i.e. steps along the x-axis, identified within the integration interval (a,b)
  - $x_1,\ldots,x_n$
  - x_1 and x_n need not coincide with a and/or b
- find low order piecewise polynomial interpolants for f(x) on sub-intervals defined by the nodes
- each method has a *basic rule* (i.e. formula) and ...
  - from integration of the piecewise polynomial segment
- ... a *composite rule*
  - from combining the piecewise integrated polynomial interpolants according to some scheme

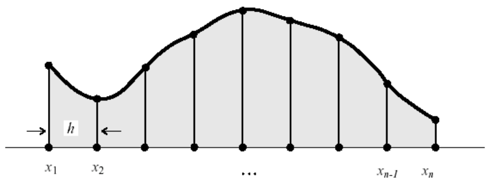
Truncation error

- can be evaluated by comparing numerical vs analytic solutions for test functions
- in general can be expressed as a function of the number of nodes
  - more nodes = less truncation error...
  - ... but not always
- adaptive methods
  - estimates of the truncation error provide feedback on the fly
  - the number of nodes can be adjusted at each step to meet user-specified accuracy

Options for node spacing

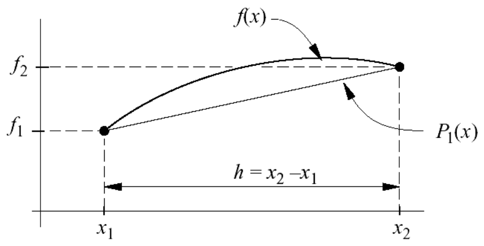
- Newton-Cotes methods
  - family of methods based on equally-spaced nodes
  - historically significant
  - in practice mostly useless nowadays
  - composite versions can be more useful
- Gaussian quadrature
  - based on irregularly-spaced nodes defined as the zeros of special orthogonal sets of polynomials
  - different families of polynomials lead to different basic rules
  - more theoretically complex and difficult
  - complicated programming
  - much reduced truncation error

Newton-Cotes methods



- n *nodes* $x_{i+1}=x_1 + ih$, i=1, ...,n-1 in the integration interval (a,b)
- h is called the *step size*
- f has known values at the nodes $f_i = f(x_i)$
- the method is *closed* if f(a) and f(b) are used as nodes
- when f is not well-behaved at the endpoints
- *open* methods for which f(a) and f(b) are not used are convenient

Trapezoidal step

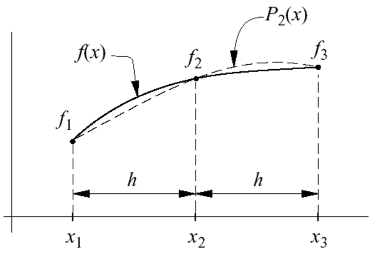


Trapezoidal rule

$$\int_{x_1}^{x_2} f(x)dx = h\left[\frac{1}{2}f_1 + \frac{1}{2}f_2\right] \quad + O(h^3 f'')$$

- a piecewise linear interpolation on each sub-interval (x_i,x_{i+1})
- integrate the first order Lagrange polynomial interpolant to get the basic rule
- error analysis shows error is dependent on h^3 times some (unknown) value of f″ inside the interval
- two point (x_1 and x_2) formula
- exact for any polynomial function up to degree 1, i.e. linear polynomial

Simpson step



Simpson’s rule

$$\int_{x_1}^{x_3} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right] \quad + O(h^5 f^{(4)})$$

- a piecewise quadratic interpolation on each sub-interval (x_i,x_{i+2})
- integrate the 2nd order Lagrange polynomial interpolant to find the basic rule
- sub-intervals used in pairs so requires an odd number of nodes
- a three point formula exact for polynomials up to degree 2
- error is $O(h^5)$ instead of anticipated $O(h^4)$, due to a lucky cancellation from symmetry
- note the step size 2h = sum of the *weights* (coefficients)

Simpson's 3/8 rule

$$\int_{x_1}^{x_4} f(x)dx = h\left[\frac{3}{8}f_1 + \frac{9}{8}f_2 + \frac{9}{8}f_3 + \frac{3}{8}f_4\right] \quad + O(h^5 f^{(4)})$$

- a piecewise cubic interpolation on each sub-interval (x_i,x_{i+3})
- contrast this with what we did in constructing the cubic spline interpolation, viz. fit cubic through pairs of points
- the basic rule is calculated by integrating the 3nd order Lagrange polynomial interpolants
- extending requires n = 3k+1 nodes, since the sub-intervals are used in threes
- a four point formula exact for polynomials up to degree 3 (no luck this time)
- note the step size 3h = sum of the weights

Bode's rule

$$\int_{x_1}^{x_5} f(x)dx = h\left[\frac{14}{45}f_1 + \frac{64}{45}f_2 + \frac{24}{45}f_3 + \frac{64}{45}f_4 + \frac{14}{45}f_5\right] \quad + O(h^7 f^{(6)})$$

- a piecewise 4th order polynomial interpolation on each sub-interval (x_i,x_{i+4})
- the basic rule is calculated by integrating the 4th order Lagrange polynomial interpolants
- requires n = 4k+1 nodes, since the sub-intervals are used in fours
- a five point formula exact for polynomials up to degree 5 (lucky cancellation again this time)
- note the step size 4h = sum of the weights

Composite (closed) trapezoidal rule

$$\int_{x_1}^{x_n} f(x)dx = h\left[\frac{1}{2}f_1 + f_2 + f_3 + \cdots + f_{n-1} + \frac{1}{2}f_n\right]$$

- apply the trapezoidal rule n-1 times to the sub-intervals
- error is $O[f''.(b-a)^3/n^2]$
- usually we want to adjust n and keep (b-a) fixed, e.g. take twice as many steps and see how the error is reduced
- so we write error is $O(1/n^2)$ and ignore the other parts
- this equation is the most important in the series, used as the basis for subsequent more sophisticated methods

Example: Composite trapezoidal rule

$$\int_1^2 \frac{dx}{x} = \frac{h}{2}\left[f_1 + 2f_2 + 2f_3 + \cdots + 2f_{n-1} + f_n\right] \qquad O(n^2)$$

- for n = 2 sub-intervals, h = (2-1)/2 = 1/2 and you get
  I_1 = (1/4) [1/1 + 2/1.5 + 1/2] = 17/24 ≈ 0.7083
- for n = 2^2 = 4 sub-intervals h = 1/4 and you get
  I_2 = (1/8) [f(1) + 2f(5/4) + 2f(3/2) + 2f(7/4) + f(2)]
  = (1/8) [1 + 8/5 + 4/3 + 8/7 + 1/2] ≈ 0.6970
- for n = 2^3 = 8 sub-intervals h = 1/16 and you get
  I_3 = (1/16) [f(1) + 2f(9/8) + 2f(5/4) + 2f(11/8)+ 2f(3/2)
  + 2f(13/8) + 2f(7/4) + 2f(15/8) + f(2)] ≈ 0.6941...
- the exact value of the integral is ln(2) ≈ 0.693147...

Composite (closed) Simpson's rule

$$\int_{x_1}^{x_n} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{2}{3}f_3 + \frac{4}{3}f_4 + \cdots + \frac{2}{3}f_{n-2} + \frac{4}{3}f_{n-1} + f_n\right]$$

- 4th order method, i.e. error = $O(1/n^4)$ as for Simpson's rule
- derived by applying Simpson's rule to sub-intervals sequentially
- requires an odd number of nodes, so even number of sub-intervals
- it's also possible to use over-lapped Simpson steps, but requires special care at the ends....

Composite (closed) third order method

$$\int_{x_1}^{x_n} f(x)dx = h\left[\frac{5}{12}f_1 + \frac{13}{12}f_2 + f_3 + f_4 + \cdots + f_{n-2} + \frac{13}{12}f_{n-1} + \frac{5}{12}f_n\right]$$

- error = $O(1/n^3)$
- this is derived by averaging two shifted applications of the composite Simpson's rule over (a,b)
- a single trapezoidal step is included to fill in opposite ends
- the two trapezoidal steps reduce the order to n^3 instead of the expected n^4

Example: Composite O(n^3) rule

$$\int_{x_1}^{x_n} f(x)dx = h\left[\frac{5}{12}f_1 + \frac{13}{12}f_2 + f_3 + f_4 + \cdots + f_{n-2} + \frac{13}{12}f_{n-1} + \frac{5}{12}f_n\right]$$

- for n = 2 sub-intervals, h = (2-1)/2 = 1/2 and you get
  I_1 = (1/2) [(5/12)(1/1) + (13/12)(1/1.5) + (5/12)(1/2)]
  ≈ 0.6943
- for n = 4 sub-intervals h = 1/4 and you get
  I_2 = (1/4) [(5/12)f(1) + (13/12)f(5/4) + f(3/2) + (13/12)f(7/4) + 5/12)f(2)]
  = (1/4) [(5/12)(1/1) + (13/12)(1/1.25) + 1/1.5 + (13/12)(1/1.75)
  + (5/12)(1/2)] ≈ 0.6943
- the exact value of the integral is ln(2) ≈ 0.693147...
- compare convergence of this third order method to that of the second order trapezoidal rule

Improper integrals

- one of the following may apply....
  - the integrand has a finite limit at either of the finite endpoints, but cannot be evaluated (e.g. sin x/x at x=0)
  - one of the limits is ∞ or -∞
  - there is an integrable singularity at either limit (e.g. $x^{-1/2}$ at x=0)
  - there is an integrable singularity at some known place inside the interval of integration
  - there is an integrable singularity at some unknown place inside the interval of integration
- we need to be able to handle any of these cases in numerical integration of functions

Improper integrals

- first four cases are resolvable with basic methods
- something like the composite trapezoid rule is useful, because....
- to handle the improper aspects we need to use an open formula
  - neither f(a) nor f(b) need to be evaluated
- composite midpoint method is an open alternative:

$$\int_{x_1}^{x_n} f(x)dx = h[f_{3/2} + f_{5/2} + f_{7/2} + \cdots + f_{n-3/2} + f_{n-1/2}] + O(1/n^2)$$

  - cannot double the number of steps and retain past work, but you can *triple* the steps

Infinite limits

- change variables to transform an infinite integration interval to a finite one
- example:

$$\int_a^b f(x)dx = \int_{1/b}^{1/a} \frac{1}{t^2} f\left(\frac{1}{t}\right) dt \qquad ab > 0$$

  - works if b = ∞ and a>0, or a = -∞ and b<0
  - if say b = ∞ and a<0 split into two separate integrals
- an clever integration routine can make the transformation for you when one of the required limits is 'very large'

Integrable singularities

- a transformation can remove an integrable singularity when f(a) or f(b) is infinite
- example:
  - an inverse square root singularity at a can be fixed by using t = √x-a ..... so x = t^2+a and dx = 2tdt

$$\int_a^b f(x)dx = \int_0^{\sqrt{b-a}} 2tf(a+t^2)dt \qquad (b > a)$$

- at the lower limit we would have

$$\int_a^b f(x)dx = \int_0^{\sqrt{b-a}} 2tf(b-t^2)dt \qquad (b > a)$$

- numerical methods cannot fix ill-posed problems with integrals that are impossible

Newton-Cotes formulas re-visited

$$\int_a^b f(x)dx \approx \int_a^b p_{n-1}(x)dx$$

- p_{n-1}(x) is the Lagrange polynomial interpolation for f(x) on the interval (a,b)
- all Newton-Cotes formulas can be considered in the following view:

$$\begin{aligned}\int_a^b p_{n-1}(x)dx &= \int_a^b \left[\sum_{j=1}^{n} L_j(x)f_j\right] dx \\ &= \sum_{j=1}^{n}\left[\int_a^b L_j(x)dx\right] f_j \\ &= \sum_{j=1}^{n} w_j f_j\end{aligned}$$

Weights and nodes

$$\int_a^b f(x)dx \approx \sum_{j=1}^n w_j f(x_j)$$

- the x_j are *nodes* $a \le x_j \le b$
- the w_j are called *weights*
  - obtained by integrating the jth Lagrange interpolating polynomial for f(x) on (a,b):

$$w_j = \int_a^b L_j(x)dx$$

Weights and nodes

- Newton-Cotes rule with n nodes cannot give precision greater than O(n-1)
  - exact for polynomials of degree n-1 or less
- we can improve this situation without increasing the number of nodes ....
- ... by providing more degrees of freedom in the fit

- Gaussian quadrature chooses the nodes carefully to have special properties with respect to f(x)

Gaussian quadrature

$$\int_a^b f(x)dx \approx \sum_{j=1}^n w_j f(x_j)$$

- both weights and nodes are freely chosen for best precision
  - twice as many degrees of freedom
  - nodes will not be equally spaced in general
- how do we ...
  - optimize the node positions?

    and ...
  - find the corresponding weights?

Two versions of Gaussian quadrature

- given function w(x) and integer n we want to find weights w_j and nodes x_j so that the approximation

$$\int_a^b w(x)f(x)dx \approx \sum_{j=1}^n w_j f(x_j)$$

is exact if f(x) is a polynomial

- if we write g(x) = w(x)f(x) and $v_j = w_j/w(x_j)$ we get an alternative form of the Gaussian quadrature formula

$$\int_a^b g(x)dx \approx \sum_{j=1}^n v_j g(x_j)$$

The easy way to nodes and weights

- lookup tables
  - nodes and weights are available for given weight functions w(x) and values of n
- *can* be used without understanding the theory *but* ....
  - the theory is important for its intrinsic value so you should be aware of the basics
  - custom-designed weight functions are sometimes needed to solve a specific problem
  - you have to know enough to know which version of the Gaussian quadrature formula (slide 28) to use
- compromise: we give some results and outline the theoretical background (so dive in ....)

Gaussian quadrature

- how exact can the approximate integral be with n nodes?
  - to a polynomial of degree n-1 or less with Newton-Cotes formulas
  - to a polynomial of degree 2n-1 or less for movable nodes
- compared to using fixed nodes Gaussian quadrature gives
  - an order twice as large for the same number of function evaluations
- that's a LOT of room to improve precision but .....

Gaussian quadrature

- .... higher order does not *necessarily* mean better accuracy
  - need a smooth function to benefit from Gaussian quadrature
  - it should be well approximated by a polynomial
- for craggy functions .... you do just a well or better with the composite trapezoid rule
- an additional benefit of Gaussian quadrature:
  - the approximation is exact for functions which are products of a given w(x) and a polynomial, not just plain polynomials
- to understand the basis for all this we need to visit the world of orthogonal polynomials .....

Digression: Orthogonal functions

- recall .... the concept of *inner product space* from linear algebra
- an inner product with *weight function* w(x) can be defined in a function space:

$$\langle f|g\rangle \equiv \int_a^b W(x)f(x)g(x)dx$$

- the result is a number, not a function, of course
- on the interval [a,b] two functions f and g are ....
  - *orthogonal* if $\langle f|g\rangle = 0$ (and $f \neq g$)
  - *normalized* if $\langle f|f\rangle = ||f||^2 = 1$

Digression: Orthogonal functions

- an *orthonormal set* of functions is mutually orthogonal and normalized (just like an orthonormal basis)
- example: {1, x} are orthogonal on the interval [-1,1] with either of the weight functions:
  - w(x) = 1
  - $w(x) = \sqrt{1-x^2}$
- example: {cos kx}, {sin kx} k=1,...,n are orthogonal sets of functions on $[0,\pi]$
- the mononomials $\{1, x, x^2, x^3, ...\}$ are NOT an orthogonal set, but ....

Digression: Orthogonal functions

- .... important sets of polynomials $p_j(x)$, j = 0,1,2,.... can be defined on an interval (a,b) so that
  - there is exactly one polynomial of each degree
  - they are mutually orthogonal with respect to a given weight function w(x)
- after a heavy dose of linear algebra we can obtain recursive definitions for these orthogonal polynomials ....

Digression: Orthogonal functions

$$p_{-1}(x) \equiv 0$$

$$p_0(x) \equiv 1$$

$$p_{j+1}(x) = (x - a_j)p_j(x) - b_j p_{j-1}(x) \qquad j = 0, 1, 2, \ldots$$

$$a_j = \frac{\langle xp_j|p_j\rangle}{\langle p_j|p_j\rangle} \qquad j = 0, 1, \ldots$$

$$b_j = \frac{\langle p_j|p_j\rangle}{\langle p_{j-1}|p_{j-1}\rangle} \qquad j = 1, 2, \ldots$$

Digression: Orthogonal functions

- the polynomials are monic
  - coefficient is one for the highest degree term x^j
- they can be normalized in the usual way
  - divide each p_j by its 2-norm $\langle p_j|p_j\rangle^{1/2}$
- each polynomial $p_j(x)$ has j distinct (real) roots in the interval (a,b)
  - these j roots of $p_j(x)$ interlace with the j-1 roots of p_{j-1}
  - exactly one root of p_j lies between any two adjacent roots of p_{j-1}
- root-interlacing is very handy for finding all roots of $p_j(x)$
  - find the one root of p_1
  - continually bracket the roots for each higher j
  - apply something like Newton's method to locate the roots

Digression: Orthogonal functions

- but .... why would anyone want to find all the roots of an orthogonal polynomial anyway?
- because the roots of orthogonal polynomials are the key to Gaussian quadrature

read on ☞

How to get the nodes?

- in an n-point Gaussian quadrature on the interval (a,b) with weight function w(x):

  the nodes are chosen to be the n roots of the orthogonal polynomial $p_j(x)$ with the same weight function and interval
- this is a fundamental theorem and the starting point for myriad different pathways in the subject
  - calculations tend to be quite complicated
- once you have the nodes you need to calculate the corresponding weights w_j .....

How to get the weights?

- simplest approach: substitute each $p_j(x)$ for f(x) in the integral approximation (slide 28)
  - the result should be exact
  - (witchcraft) slip a $p_0(x)$ inside the integral without changing it since $p_0(x) = 1$ (a constant)
  - the thing evaluates to zero for j=1,2,... because all the other polynomials are orthogonal to p_0
- you get the system of equations

$$\begin{bmatrix} p_0(x_1) & \cdots & p_0(x_n) \\ p_1(x_1) & \cdots & p_1(x_n) \\ \vdots & & \vdots \\ p_{n-1}(x_1) & \cdots & p_{n-1}(x_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \int_a^b w(x)p_0(x)dx \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

How to get the weights?

- solve to get the weights $\{w_1,..., w_n\}$
- it turns out that they give an integral approximation exact for polynomials up to the next n-1 degree as well
- so the quadrature formula is exact for polynomials up to degree 2n-1 (as anticipated)
- other ways to get the weights include .....
  - use lookup tables for known weight functions
  - study the theory in depth and develop the most well-known and useful cases [an important study for engineers]
  - use the integral formula (slide 25) with the Lagrange interpolating polynomials for f(x)

Taking stock

- three steps are used to develop a Gaussian quadrature formula for a given w(x) ....

  1. **Generate the orthogonal polynomials $\{p_0, ..., p_n\}$**
     - i.e. evaluate coefficients a_j and b_j in the formulas on slide 39
  2. **Find the zeros of $p_n(x)$ .... the NODES**
  3. **Calculate the associated w_j .... the WEIGHTS**

- for classical weight functions ....
  - the orthogonal polynomials are known so the work is simplified
  - but still somewhat involved
- for non-classical weight functions ....
  - the non-trivial process above must be followed

Legendre polynomials

- the simplest case: w(x) = 1 and interval (-1,1)
- the *Legendre polynomials* are generated by

$$(j+1)P_{j+1} = (2j+1)xP_j - jP_{j-1}$$

- the first few Legendre polynomials are:

  $P_0(x) = 1$
  $P_1(x) = x$
  $P_2(x) = (3x^2 - 1)/2$
  $P_3(x) = (5x^3 - 3x)/2$
  $P_4(x) = (35x^4 - 30x^2 + 3)/8$
- these lead to *Gauss-Legendre quadrature*.....

2-point Gauss-Legendre quadrature

- nodes are the zeros of $P_2(x)$
  - the solutions of $3x^2-1 = 0$
  - $x_1 = -1/\sqrt{3}$ and $x_2 = 1/\sqrt{3}$
- weights are obtained from the Lagrange interpolating polynomial formula on slide 25

$$\begin{aligned} w_1 &= \int_{-1}^{1} L_1(x)dx \\ &= \int_{-1}^{1} \frac{x-x_2}{x_1-x_2}dx \\ &= \int_{-1}^{1} \frac{x-1/\sqrt{3}}{-2/\sqrt{3}}dx \\ &= \int_{-1}^{1} -\frac{\sqrt{3}}{2}\left(x-\frac{1}{\sqrt{3}}\right)dx \\ &= 1 \end{aligned} \qquad \begin{aligned} w_2 &= \int_{-1}^{1} L_2(x)dx \\ &= \int_{-1}^{1} \frac{x-x_1}{x_2-x_1}dx \\ &= \int_{-1}^{1} \frac{x+1/\sqrt{3}}{2/\sqrt{3}}dx \\ &= \int_{-1}^{1} -\frac{\sqrt{3}}{2}\left(x+\frac{1}{\sqrt{3}}\right)dx \\ &= 1 \end{aligned}$$

Example: 2-point Gauss-Legendre quadrature

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{1} (x^3+x^2+x+1)dx = 8/3$$

- 2-point Gauss Legendre quadrature should give the exact answer for this integral (cubic, 2n-1 = 3)

$$\begin{aligned} \int_{-1}^{1} f(x)dx &= w_1 f(x_1) + w_2 f(x_2) \\ &= 1f(-1/\sqrt{3}) + 1f(1/\sqrt{3}) \\ &= 0.56353297 + 2.10313369 \\ &= 2.66666666 \end{aligned}$$

- as expected the answer is exact to sig. figures given
- consider a more involved example .....

5-point Gauss-Legendre quadrature

- nodes are the zeros of $P_5(x)$
- these can be found numerically, or by lookup tables:
  0, ±0.33998 10435 84856, ±0.90617 663115 94053
- ditto for the corresponding weights
  0.56888 88888 88889, 0.47862 86704 99366,
  0.23692 68850 56189
- use the 5-point formula to estimate ln 2 by evaluating

$$\int_{1}^{2} \frac{dx}{x}dx = \ln 2 - \ln 1 = \ln 2 \doteq 0.69314718$$

- a problem here .... integration is over [1,2] so Gauss-Legendre is not applicable directly

Adjusting integration limits

- to apply Gauss-Legendre quadrature over [a,b] not [-1,1]
- define a new function using the transformation
  z = (2x - (a+b)) / (b - a)
- x = a → z = -1 and x = b → z = 1 as required
- new function is g(z) = f(x) = f((b - a)z + (a + b)/2)
- integral is converted to a standard [-1,1] integral of form F(z) dz
- nodes z_i and weights w_i obtained from tabulated values are used in the expansion with F(z)
- in the 5-point example we have:
  z = (2x - (2 + 1)) / (2 - 1) = 2x - 3
  g(z) = 2 / (z + 3)

Example: 5-point Gauss-Legendre quadrature

- applying the transformation gives

$$\int_{1}^{2} \frac{dx}{x}dx = \int_{-1}^{1} \frac{2}{z+3}\frac{dz}{2} = \int_{-1}^{1} \frac{1}{z+3}dz$$

- so F(z) = 1 / (z+3)
- now apply the 5-point Gauss Legendre quadrature formula, using the tabulated nodes and weights with F(z)

$$\begin{aligned} \int_{-1}^{1} F(z)dz &= w_1F(z_1) + w_2F(z_2) + w_3F(z_3) + w_4F(z_4) + w_5F(z_5) \\ &= w_1\frac{1}{z_1+3} + w_2\frac{1}{z_2+3} + w_3\frac{1}{z_3+3} + w_4\frac{1}{z_4+3} + w_5\frac{1}{z_5+3} \\ &\doteq 0.69314712 \end{aligned}$$

- the answer is accurate to 6 significant figures

Another example

$$\int_{0}^{2} e^{-x^2}dx$$

- f(x) = exp(-x^2), a = 0, b = 2
- transformation z = (2x - (2+0)) / (2-0) = x-1 or x = z+1
- g(z) = f(x) = exp(-$(z+1)^2$)
- f(x)dx = g(z)dz, so F(z) = g(z) since dz = dx
- apply 2-point Gauss-Legendre quadrature

$$\begin{aligned} \int_{-1}^{1} e^{-(z+1)^2}dz &= \int_{-1}^{1} F(z)dz \\ &= 1F(z_1) + 1F(z_2) \\ &= e^{-(-0.5773503+1)^2} + e^{-(0.5773503+1)^2} \\ &\doteq 0.9195 \end{aligned}$$

Laguerre polynomials

- take w(x) = e^{-x} and interval [0,∞]
- the *Laguerre polynomials* are generated by

$$\mathcal{L}_j(x) = (2j - x - 1)\mathcal{L}_{j-1}(x) - (j-1)^2\mathcal{L}_{j-2}(x)$$

- the first few Laguerre polynomials are:

$$\begin{aligned}\mathcal{L}_0(x) &= 1\\ \mathcal{L}_1(x) &= -x+1\\ \mathcal{L}_2(x) &= x^2-4x+2\\ \mathcal{L}_3(x) &= -x^3+9x^2-18x+6\end{aligned}$$

- this leads to *Gauss-Laguerre quadrature*.....

Gauss-Laguerre quadrature

$$\int_0^\infty e^{-x} f(x)dx \doteq \sum_{i=1}^{n+1} w_i f(x_i)$$

- use to evaluate integrals of the above type
- nodes x_i are the roots of the nth Laguerre polynomial
- apply the Lagrange interpolation formula

$$f(x) = \sum_{i=0}^{n} L_i(x) f(x_i) + \left[\prod_{i=0}^{n}(x-x_i)\right]\frac{f^{(n+1)}(\xi)}{(n+1)!}$$

  - this is valid for some 0<ξ<∞
  - the error term R_n is as given [not derived here]
- at this point the nodes x_i have yet to be defined....

Why does Gaussian quadrature work?

$$\int_a^b f(x)dx \approx \int_a^b p_{n-1}(x)dx$$

- the Gauss-Legendre approximation is
  - exact for polynomials of deg n-1 or less [for sure] but ....
  - the error term above can also be made zero for any polynomial f(x) of deg 2n-1 or less provided ....
- **the nodes are chosen to be the zeros of the nth Legendre polynomial P_n(x)**
  - we'll explain how the orthogonality properties of Legendre polynomials ensure this
- *notation confusion:*
  - p_{n-1}(x) is the interpolating Lagrange polynomial BUT ....
  - P_n(x) is the nth Legendre polynomial used to define the nodes

Gauss-Laguerre quadrature

- we'll explain how it works with Gauss-Laguerre
- first pick the nodes ...
  - suppose f(x) is a polynomial of degree 2n-1
  - then $f^{(n+1)}(\xi)/(n+1)!$ is a polynomial of degree n-1
  - so let's call this polynomial ψ(x)
- now we have

$$f(x) = \sum_{i=0}^{n} L_i(x) f(x_i) + \left[\prod_{i=0}^{n}(x-x_i)\right]\psi(x)$$

- to compute the integral at the top of slide 51, we can
  - multiply each term in this expression by e^{-x} and ....
  - integrate both sides

Gauss-Laguerre quadrature

$$\int_0^\infty e^{-x} f(x) = \sum_{i=0}^{n} f(x_i)\int_0^\infty e^{-x}L_i(x)dx + \int_0^\infty e^{-x}\left[\prod_{i=0}^{n}(x-x_i)\right]\psi(x)dx$$

- how to pick the x_i nodes so the error term is *Z E R O ?*
- *COOL OBSERVATION 1:*
  - the product [..] can be made into the Laguerre polynomial L_n of degree n by ....
  - **picking the nodes x_i to be the n zeros of the nth Laguerre polynomial**
- *COOL OBSERVATION 2:*
  - can expand the (degree n-1) polynomial ψ(x) in terms of Laguerre polynomials L_1 to L_{n-1} because ....
  - the Laguerre polynomials are a basis for the space of polynomials deg n-1 or less

Gauss-Laguerre quadrature

- **the integral in the righthand side is identically zero**
  - **by the orthogonality properties of the Laguerre polynomials**
- so.... the lefthand side integral is identically equal to the first term above
- D O N E
  - the approximation is EXACT for polynomials of deg 2n-1 or less
- the same reasoning justifies the principles of Gaussian quadrature with other polynomial types
  - Gauss-Legendre
  - Gauss-Chebyshev
  - Gauss-Hermite
  - . . . .

Gauss-Laguerre quadrature

- weights w_i are given by

$$w_i = \int_0^\infty e^{-x} L_i(x) dx = \int_0^\infty e^{-x} \prod_{j=0, j\neq i}^{n} \left[\frac{x - x_j}{x_i - x_j} \right] dx$$

- 2-point values for nodes are
  0.58578 64376 27, 3.41421 35623 73
- with corresponding 2-point weights
  0.85355 33905 93, 0.14644 66094 07
- example: calculate Γ(1.8) where the *gamma function* interpolates the integral factorial function

$$\Gamma(\alpha) = \int_0^\infty e^{-x} x^{\alpha-1} dx$$

- for integral values $\Gamma(\alpha) = (\alpha - 1)!$

Example: Gauss-Laguerre quadrature

- use Gauss-Laguerre quadrature with $\Gamma(\alpha) \doteq \sum_{i=0}^{n} w_i x_i^{\alpha-1}$
- say $\Gamma(2) \doteq \sum_{i=0}^{n} w_i x_i$
- examine the n-point weights and nodes for Laguerre
  - verify this sum is 1 for all n
  - you can see this in the tabulated values
- for non-integral α = 1.8 use f(x) = $x^{1.8-1} = x^{0.8}$
  - not a polynomial so the integral will not be exact
- with the 2-point formula $\Gamma(1.8) \doteq w_1 x_1^{0.8} + w_2 x_2^{0.8} = 0.947566$
  - 6 significant figure value is 0.931384
- 14-point Gauss-Laguerre gives 0.931771

Composite Gaussian quadrature

- apply Gaussian quadrature to sub-panels across a larger integration interval [a,b]
- need to ....
  - locate the nodes for each sub-integration
  - adjust the limits of integration as necessary for each sub-interval
  - e.g. convert all limits to [-1,1] if you use Gauss-Legendre quadrature
  - decide on closed vs open vs semi-open question

Adaptive quadrature

- balances efficiency and accuracy
- allows a variable step size
  - take into account differences in the shape of the function across the integration panel
- algorithms are available to adjust automatically
  - the step size and ...
  - the number of sub-panels to achieve the desired tolerance [if possible!]
- efficient because small sub-panels (=more function evaluations) are used only where necessary
- the standard modern method of numerical integration

Matlab implementation

- *q = quad(**fun,a,b**,tol)*
  - uses an adaptive Simpson's rule
- *q = quadl(**fun,a,b**,tol)*
  - uses a modified adaptive closed Gauss-Lobatto quadrature
  - nodes selected to optimize refining the panel size later
  - based on Legendre polynomials
- function arguments
  - *fun* can be defined as an inline function
  - use vector expressions in *fun*:
    e.g. *f = inline('1./(x.^3 - 2*x -5)'); q = quadl(f,0,2)*
  - default *tol* = 1e-6
  - also count function evaluations by *[q,fcnt] = quadl(**fun,a,b**,tol)*
- these are very powerful algorithms.....!

Introduction to numerical differentiation

- nodes and weights for derivative approximation
- finite difference methods
  - forward, backward, and central differences
- smoothing methods
  - Lagrange interpolation
  - Newton interpolation and divided differences
  - cubic splines

Numerical differentiation

Three situations may occur in numerical differentiation ....

1. The function f(x) is known in symbolic form but
   - may be difficult or inconvenient to differentiate symbolically
2. Some exact function values f(x_i) are tabulated
   - for instance calculated by function evaluations
3. Some approximate function values f(x_i) are tabulated
   - for instance obtained from experimental data

Numerical differentiation

- numerical differentiation approximates the derivative f′(c) as a weighted sum

$$f'(c) \approx \sum_{i=1}^{n} w_i f_i$$

  - the *sampled values* f_i = f(x_i) are obtained at....
  - ... *nodes* x_i near c
  - the *weights* w_i depend on c and the x_i
- the *exactness degree* m is the largest integer that gives an exact

$$f'(c) = \sum_{i=1}^{n} w_i f_i$$

for polynomials of degree ≤ m

Numerical differentiation

- to be useful the weights must sum to zero
  - this is true regardless of c and x_i
- here's why ....
  - for an exactness degree ≥ 0 we have f′(c) = 0 to be exact when f(x) = 1
  - substitute in the weighted sum:

$$\begin{aligned} f'(c) &= w_1f_1 + w_2f_2 + \dots + w_nf_n \\ &= w_1f(x_1)+ w_2f(x_2) + \dots + w_nf(x_n) \\ &= w_1\text{•}1+ w_2\text{•}1 + \dots + w_n\text{•}1 \\ &= w_1+ w_2 + \dots + w_n \\ &= 0 \end{aligned}$$

Difference approximations

- the *2-point forward-difference* approximation is

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

- the *2-point backward-difference* is

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

- accuracy is obtained by balancing
  - roundoff error as the *stepsize* h gets small and
  - truncation error caused by the first order Taylor series approximation used

First-order approximation

- the first-order Taylor series approximation for f about c is

$$f(c+h) = f(c) + f'(c)h + f''(\xi)h^2/2$$

  - the last term is the second-order error
  - the unknown value ξ is between c and c+h
- we can solve for

$$f'(c) = \frac{f(c+h) - f(c)}{h} - f''(\xi)h/2$$

- the Taylor series for f(c-h) gives

$$f'(c) = \frac{f(c) - f(c-h)}{h} + f''(\zeta)h/2$$

  - the unknown value ζ is between c-h and c now

First-order approximation

- the forward-difference approximation will be exact at some point between c and c+h
  - this is around c+h/2 if the funciton isn't too wild
- same for the backward difference but
  - the point is between c-h and c
  - so exact around c-h/2 if the function is reasonable
- the mean value theorem explains these observations or ...
- use geometric reasoning

Central differences

- averaging the forward and backward differences gives

$$f'(c) = \frac{f(c+h) - f(c-h)}{2h} + f'''(\xi)h^2/6$$

  - error is *O*(h^2) now
  - the Taylor series are second order since the *O*(h) error terms cancel
- the *central difference* approximation is

$$f'(c) = \frac{f(c+h) - f(c-h)}{2h} .$$

  - the error term is now O(h^2) .... a better choice
  - the approximation is exact at some value between c-h and c+h
  - ... so around x=c if the function is well-behaved

Using Lagrange interpolation

- given: n (not necessarily equi-spaced) nodes x_i
- to find: weights for the kth derivative approximation

$$f^{(k)}(c) \approx \sum_{i=1}^{n} w_i f_i$$

- use the Lagrange interpolation for f(x) with the given nodes (Unit III) :

  $p_{n-1}(x) = y_1 L_1(x) + y_2 L_2(x) + \dots + y_n L_n(x)$

  with $L_j(x) = \prod_{k=1, k\neq j}^{n} \frac{x - x_k}{x_j - x_k}$

- $f(x) \approx p_{n-1}(x)$ near c so we can expect $f^{(k)}(x) \approx p_{n-1}^{(k)}(x)$

Using Lagrange interpolation

- evaluating the Lagrange polynomial derivative is easy:

$$p_{n-1}^{(k)}(x) = L_1^{(k)}(x)f(x_1) + L_2^{(k)}(x)f(x_2) + \cdots + L_n^{(k)}(x)f(x_n)$$

- so we choose the weights in the derivative approximation f′(x) (slide 62) as

$$w_i = L_i^{(k)}(x)$$

- in fact the converse is also true by uniqueness:
  - exactness of degree n is ensured by choosing the weights as above

Using Newton interpolation

- with a Newton interpolation we have

$$f(x) \approx P_n(x) = f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2) +$$
$$\cdots + f[x_1, x_2, \cdots, x_{n+1}](x - x_1)(x - x_2)\cdots(x - x_n)$$

- brute force .... differentiate P_n(x) to approximate f′(x) across the interval (x_1,x_n)
- locally .... use groups of points to get lower order approximations for local regions
  - can evaluate f′(x) for a given x value
  - point of interest cannot lie at the upper end of the data range
  - use divided difference tables for calculations
  - can reverse the points to gain choice-or-order flexibility at the upper end

Using plain function differences

- a general formula for the derivative of the Newton poly at an arbitrary point is messy
  - easiest to handle specific examples as they arise
- it's possible to write a simpler general formula if ....
  - the x-values are equally spaced
  - the derivative f(x) is to be evaluated at one of the data values x_i
- using differences we have

$$f'(x_i) = (1/h)[\Delta f_i - \Delta^2 f_i/2 + \Delta^3 f_i/3 - \cdots + (-1)^{n-1}\Delta^n f_i/n]$$

  - h = x - x_{i-1} and $\Delta^n f_i$ is the nth order function difference at x_i
  - error is O(h^n)
  - n=1 is forward difference
  - n=2 is central difference

Smoothing sampled data

- sampled data should be smoothed before differentiating
  - for instance to obtain velocity and acceleration from sampled position data
  - applying difference techniques directly to noisy data is likely to produce nonsense

Three Standard Methods

1. Cubic splines
2. B-splines
3. Least-squares curve fitting

REMEMBER......

- numerical differentiation is an inherently unstable local process
- quadrature is a global process that includes an inherent smoothing
  - positive and negative errors will tend to cancel in integration