Queens College, CUNY, Department of Computer Science

Numerical Methods

CSCI 361 / 761

Fall 2017

Instructor: Dr. Sateesh Mane

January 25, 2019

6 Lecture 6

6.1 Numerical evaluation of integrals

- This lecture is concerned about the numerical evaluation of integrals

$$I = \int_a^b f(x)\,dx\,. \qquad (6.1.1)$$

- We shall mainly consider only one-dimensional integrals. The numerical evaluation of integrals in multiple dimensions is much more difficult. The computational complexity increases rapidly as the number of dimensions increases.
- We shall also be concerned with integrals over a finite domain, i.e. both the limits of integration a and b are finite.
- We shall also assume that the integrand $f(x)$ is finite throughout the domain of integration.
- An integral with these properties is called a **proper integral** (but see exceptions below).
- An **improper integral** is an integral where one or both of the limits of integration are infinite, and/or the functon $f(x)$ diverges at one or more points in the range of integration. Such integrals require special treatment and are much more difficult to compute numerically.
- Numerical integration is a much more stable procedure than numerical differentiation. There are many good quality algorithms for numerical integration.

6.2 Integral with $0/0$ integrand

- There is another case not listed above. Consider the following integral

$$I = \int_{-\pi}^{\pi} \frac{\sin(x)}{x}\, dx\,. \tag{6.2.1}$$

- The integrand in eq. (6.2.1) is finite for all values of x in the domain of integration.
- The limits of integration in eq. (6.2.1) are both finite.
- The value of the integral in eq. (6.2.1) is finite.
- *Nevertheless, there is a difficulty with the numerical evaluation of the integral in eq.* (6.2.1)*.*
- The difficulty is of course at $x = 0$. Both the numerator and the denominator of the integrand equal zero at $x = 0$. Hence the integrand equals $0/0$ at $x = 0$:

$$\frac{\sin(x)}{x} = \frac{0}{0} \qquad\qquad (x = 0). \tag{6.2.2}$$

- *What to do?*
- More generally, suppose the integrand is of the form

$$f(x) = \frac{f_{\rm num}(x)}{f_{\rm denom}(x)}\,. \tag{6.2.3}$$

- Then suppose that both $f_{\rm num}(x) = 0$ and $f_{\rm denom}(x) = 0$ at some value $x = x_0$.
- However, the value of $f(x)$ is finite at $x = x_0$.
- What this means in pure mathematics is that *the limit as $x \to x_0$ is finite:*

$$\lim_{x\to x_0} \frac{f_{\rm num}(x)}{f_{\rm denom}(x)} = (\text{finite limit})\,. \tag{6.2.4}$$

- Note that, even in pure mathematics, we cannot put $x = x_0$ directly; we must *take a limit.*
- Unfortunately, a computer cannot "take a limit" (just as it cannot do so in numerical differentiation).
- A computer will evaluate $f(x)$ at $x = x_0$ as $0/0$ and the numerical computation of the integral will fail.
- Hence the integrand is **finite but difficult to evaluate numerically.**
- We shall declare such integrals to also be improper integrals.

6.3 Integral with discontinuous integrand

- As long as the integrand is finite in the domain of intehration, it is acceptable if the integrand is discontinuous. For example consider the step function with a discontinuity at $x = 0$:

$$f_{\text{step}}(x) = \begin{cases} 0 & (x < 0)\,, \\ 1 & (x \geq 1)\,. \end{cases} \tag{6.3.1}$$

  Let $a < 0$ and $b > 0$. Then the following is a proper integral

$$I = \int_a^b f_{\text{step}}(x)\,dx\,. \tag{6.3.2}$$

  We can compute its value by integrating from a to 0 and then 0 to b.

- Let $f_{\text{disc}}(x)$ be a bounded function with a discontinuity at $x = c$, where $a < c < b$. Then we can numerically compute the overall integral as a sum of two terms, from a to c abd c to b:

$$I = \int_a^b f_{\text{disc}}(x)\,dx = \left(\int_a^c f_{\text{disc}}(x)\,dx\right) + \left(\int_c^b f_{\text{disc}}(x)\,dx\right). \tag{6.3.3}$$

- Let $f_{\text{disc}}(x)$ be a bounded function with a **finite number of discontinuities** at $x = c_i$, $(i = 1, 2, \dots, i_{\max})$, where

$$a < c_1 < c_2 < \cdots < c_{i_{\max}} < b\,. \tag{6.3.4}$$

  Then we can numerically compute the overall integral as a sum of multiple terms in an obvious way

$$\begin{aligned} I &= \int_a^b f_{\text{disc}}(x)\,dx \\ &= \left(\int_a^{c_1} f_{\text{disc}}(x)\,dx\right) + \left(\int_{c_1}^{c_2} f_{\text{disc}}(x)\,dx\right) + \cdots + \left(\int_{c_{i_{\max}}}^b f_{\text{disc}}(x)\,dx\right). \end{aligned} \tag{6.3.5}$$

- Now suppose $f(x)$ is a bounded function with an **infinite number of discontinuities** in the interval from a to b. For example let $a = 0$ and $b = 1$ and suppose f is discontinuous at $1/j$, where $j = 2, 3, 4 \dots$.

  1. We cannot handle such a function numerically.
     The integral of such a function is an **improper integral**.

6.4 More foolishness

6.4.1 Infinitely rapid oscillations

- Consider the integral
$$I = \int_0^1 \sin\Bigl(\frac{1}{x}\Bigr)\, dx\,. \tag{6.4.1}$$
- The value of the integral is obviously finite, because $|\sin(1/x)| \le 1$.
- However, computationally we cannot deal with the above integrand.
- The above integrand is mathematically not well defined at $x = 0$.
- This is an improper integral.
- Put $t = 1/x$, then we obtain
$$I = \int_1^\infty \frac{\sin(t)}{t^2}\, dt\,. \tag{6.4.2}$$
This is obviously an improper integral.

6.4.2 Finite but hard to compute integrands

- Consider the integrals
$$I_1 = \int_0^1 e^{-1/x}\, dx\,, \qquad I_2 = \int_0^1 e^{-1/x^2}\, dx\,. \tag{6.4.3}$$
- The values of both integrals are finite. The limits of both integrands are zero as $x \to 0$ through positive values
$$\lim_{x\to 0^+} e^{-1/x} = 0\,, \qquad \lim_{x\to 0^+} e^{-1/x^2} = 0\,. \tag{6.4.4}$$
(For e^{-1/x^2} we do not need to restrict $x > 0$ to take the limit.)
- However, computationally we cannot compute the values of $e^{-1/x}$ or e^{-1/x^2} at $x = 0$.
- Some special treatment is required to calculate both of the above integrals.
- We shall consider both of the above to be improper integrals.

6.5 General remarks

- It is obvious that there is a large variety of “pathological” functions.
- The functions may be finite (bounded) throughout the domain of integration from a to b, but even so, they behave badly in one way or another.
- Hence they yield improper integrals.
- It is too difficult (for us) to list all the possible cases.
- We shall asume that the functions we deal with are all bounded and continuous, or have at most a finite set of discontinuities. If they have a finite set of discontinuities, we employ eq. (6.3.5).
- Hence we shall assume below that $f(x)$ is continuous and bounded in the domain $a \leq x \leq b$, and behaves “sufficiently well” (although that is hard to define precisely).

6.6 One-dimensional proper integrals

- Without loss of generality, we may assume $a < b$.
- We subdivide the domain of integration into n subintervals, with end-points $x_0 < x_1 < x_2 < \cdots < x_n$, where $x_0 = a$ and $x_n = b$.
- The subintervals do not have to be of equal length, but in most cases they are. (The algorithms are simpler that way.) We subdivide the domain of integration into n equal subintervals, of length $h = (b - a)/n$. Then

$$x_0 = a\,, \qquad x_1 = a + h\,, \quad \ldots \quad x_j = a + jh\,, \quad \ldots \quad x_n = a + nh = b\,. \tag{6.6.1}$$

- We compute the function values at the $n + 1$ points $f(x_0)$, $f(x_1)$, $\ldots$, $f(x_n)$.
- We now have to use this data to estimate the value of the integral. There are numerous algorithms to do so.

6.7 Lagrange polynomial

- Although this lecture is about the numerical evaluation of integrals $\int_a^b f(x)\,dx$, we digress to the helpful topic of the **Lagrange polynomial**.

- Suppose we have a set of points (x_i, y_i), $i = 1, 2, \dots, n$, where the x_i are all distinct (i.e. there is no vertical scatter in the data). Without loss of generality we may assume the x_i are sorted from the smallest to the largest, so $x_1 < x_2 < \cdots < x_n$.

- We can always find a polynomial of sufficiently high degree which passes through all the points.

- What we seek is the polynomial of *lowest* degree which passes through all the points. This polynomial is unique. It is called the **Lagrange polynomial**.

- First define the following polynomial

$$\ell_1(x) = \frac{(x-x_2)(x-x_3)\dots(x-x_n)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_n)}\,. \tag{6.7.1}$$

  Note the following:

  1. The denominator does not vanish, so the polynomial $\ell_1(x)$ is well defined.
  2. At $x = x_1$, then $\ell_1(x_1) = 1$.
  3. At all the other points $x = x_2$, $x = x_3$, $\dots$, $x = x_n$, then $\ell_1(x_i) = 0$, $i \neq 1$.
  4. This is called a **Lagrange basis polynomial**. It equals 1 at $x = x_1$ and 0 at all the other points $x = x_i$, $i \neq 1$.

- We can easily construct a second Lagrange basis polynomial $\ell_2(x)$, which equals 1 at $x = x_2$ and 0 at all the other points $x = x_i$, $i \neq 2$. The polynomial is obviously

$$\ell_3(x) = \frac{(x-x_1)(x-x_3)\dots(x-x_n)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_n)}\,. \tag{6.7.2}$$

  This time we omit a factor $(x - x_2)$ in the numerator, and the denominator has an obvious pattern.

- We can similarly construct a third Lagrange basis polynomial $\ell_3(x)$, which equals 1 at $x = x_3$ and 0 at all the other points $x = x_i$, $i \neq 3$.

- The pattern is obvious. We can construct the Lagrange basis polynomial $\ell_j(x)$, which equals 1 at $x = x_j$ and 0 at all the other points $x = x_i$, $i \neq j$.

- The overall Lagrange polynomial $L(x)$ is then given by the following weighted sum of n Lagrange basis polynomials

$$\begin{aligned} L(x) &= y_1\,\ell_1(x) + y_2\,\ell_2(x) + y_3\,\ell_3(x) + \cdots + y_n\,\ell_n(x) \\ &= \sum_{j=1}^{n} y_j\,\ell_j(x)\,. \end{aligned} \tag{6.7.3}$$

1. Notice that $L(x) = y_1$ at $x = x_1$, because $\ell_1(x_1) = 1$ and all the other basis polynomials vanish $\ell_j(x_1) = 0$ for $j \neq 1$.
2. Similarly $L(x) = y_2$ at $x = x_2$, because $\ell_2(x_2) = 1$ and all the other basis polynomials vanish $\ell_j(x_2) = 0$ for $j \neq 2$.
3. And so on: $L(x) = y_3$ at $x = x_3$, etc.

- For only two points $n = 2$, the Lagrange polynomial equals the standard formula for a straight line through (x_1, y_1) and (x_2, y_2):

$$L(x) = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1} \,. \tag{6.7.4}$$

- For three points $n = 3$, the Lagrange polynomial is a quadratic

$$L(x) = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} \,. \tag{6.7.5}$$

  If the three points lie on a straight line, the Lagrange polynomial will simplify to that straight line.

6.8 Trapezoid Rule

- Consider the subinterval $[x_i, x_{i+1}]$.
- The function values at the end points are $f(x_i)$ and $f(x_{i+1})$.
- We say the length of the interval $(= h)$ is small and so we approximate the function inside the subinterval by a **straight line.**
- The formula for a straight line $y_{\rm st}(x)$ in the (x, y) plane, passing through two points $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$ is, for $x_i \neq x_{i+1}$,

$$y_{\rm st}(x) = f(x_i) + (x - x_i)\,\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}\,. \tag{6.8.1}$$

- **We approximate the value of the integral in the subinterval by integrating using the straight line**

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \simeq \int_{x_i}^{x_{i+1}} y_{\rm st}(x)\,dx\,. \tag{6.8.2}$$

- We can also obtain the formula for the straight line using the Lagrange polynomial from eq. (6.7.4). The answer is equivalent to eq. (6.8.1)

$$L_{\rm st}(x) = -f(x_i)\,\frac{x - x_{i+1}}{h} + f(x_{i+1})\,\frac{x - x_i}{h}\,. \tag{6.8.3}$$

- The approximated value of the integral is equivalent to eq. (6.8.2)

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \simeq \int_{x_i}^{x_{i+1}} L_{\rm st}(x)\,dx\,. \tag{6.8.4}$$

- Using either eq. (6.8.2) or eq. (6.8.4), performing the integration yields the result

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \simeq \frac{h}{2}\,\big[\,f(x_i) + f(x_{i+1})\,\big]\,. \tag{6.8.5}$$

- Summing over all the subintervals yields the following approximate value for the integral

$$I \simeq \frac{h}{2}\,\big[\,f(x_0) + f(x_1)\,\big] + \frac{h}{2}\,\big[\,f(x_1) + f(x_2)\,\big] + \cdots + \frac{h}{2}\,\big[\,f(x_{n-1}) + f(x_n)\,\big]\,. \tag{6.8.6}$$

  We can collect terms and simplify to obtain

$$I = h\left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{f(x_n)}{2}\right] + O\biggl(\frac{|b-a|^3}{n^2}\,f''\biggr)\,. \tag{6.8.7}$$

  The last term is an error estimate. The notation "f''" denotes the value of $f''(x)$ at some unknown point in the interval $a \le x \le b$.

- **The formula eq.** (6.8.7) **is called the trapezoid rule.**

6.8.1 Code example

- A simple function to implement the trapezoid rule is as follows.
- We employ and extend the math library from the root finding lectures.
- The abstract base class is the same as before. We use the functon $f(x)$.

```
class MathFunction {
                        // same as before
};

class MathLibraryCPP {
public:
  // new function
  static double trapezoid(const MathFunction &mf, double a, double b, int n);

  // root finding algorithms ...
  // etc
};

double MathLibraryCPP::trapezoid(const MathFunction &mf, double a, double b, int n)
{
  if (n <= 0) return 0;
  double h = (b - a)/n;
  double sum = 0.5*(mf.f(a) + mf.f(b));
  for (int i = 1; i < n; ++i) {
    double x = a + i*h;
    sum += mf.f(x);
  }
  return sum*h;
}
```

- It is a simple rule, very popular.

6.9 Extended trapezoid rule

- We can play a clever trick with the trapezoid rule.
- Notice that if we have estimated the numerical value of the integral using n subintervals, then by placing new points at the midpoint of each subinterval, we can obtain $2n$ subintervals.
- Next note this: ***we can re-use the values of $f(x)$ that have already been computed.*** We only need to compute $f(x)$ at n new points.
- Consider the following example, where we set $a = 0$ and $b = 1$ for simplicity. At each row we only need to compute the values of f in that row:

$n = 2:$	$f(0)$				$f(\frac{1}{2})$				$f(1)$
$n = 4:$			$f(\frac{1}{4})$				$f(\frac{3}{4})$		
$n = 8:$		$f(\frac{1}{8})$		$f(\frac{3}{8})$		$f(\frac{5}{8})$		$f(\frac{7}{8})$	

- In this way, we can apply the trapezoid rule recursively. We can set a tolerance parameter `tol` and stop the recursion when the computed value of the integral changes by less than `tol`.
- We can formalize the procedure as follows. Let us define a set of stepsizes $h_j = (b-a)/2^j$, so

$$h_0 = b - a\,, \qquad h_1 = \frac{b-a}{2}\,, \qquad h_2 = \frac{b-a}{4}\,, \qquad h_3 = \frac{b-a}{8}\,, \cdots \tag{6.9.1}$$

  Let $E(j)$ be the result of the computation at step j. We begin with

$$E(0) = \frac{h_0}{2}\,\big[f(a) + f(b)\big]\,. \tag{6.9.2}$$

  Then for $E(1)$ we add one new point, at the midpoint of halfway fraction of the interval, so

$$E(1) = \tfrac{1}{2}E(0) + h_1 f(a + h_1)\,. \tag{6.9.3}$$

  For $E(2)$, we add two new points at the $\frac{1}{4}$ and $\frac{3}{4}$ fractions of the interval, so

$$E(2) = \tfrac{1}{2}E(1) + h_2\big[\,f(a + h_2) + f(a + 3h_2)\,\big]\,. \tag{6.9.4}$$

  Next we add four new points at the $\frac{1}{8}$, $\frac{3}{8}$, $\frac{5}{8}$ and $\frac{7}{8}$ fractions of the interval. Proceeding in this way, the general formula is

$$E(j) = \tfrac{1}{2}E(j-1) + h_j \sum_{i=1}^{2^j-1} f(a + (2i-1)h_j) \qquad (j = 1, 2, \dots)\,. \tag{6.9.5}$$

  This can easily be coded in a loop, where `double E[]` is an array of type `double`.

6.9.1 Code example

- A simple function to implement the extended trapezoid rule is as follows.
- We only display the new function in the math library.
- The output is a **vector** of length `jmax+1`.
- The function return type is `int` because we test for bad input.
- We return 1 for failure and 0 for success.

```
class MathFunction {
                        // same as before
};

class MathLibraryCPP {
public:
  // new function
  static int ext_trapezoid(const MathFunction &mf, double a, double b,
                           int jmax, std::vector<double> &E);

  // etc
};

int MathLibraryCPP::ext_trapezoid(const MathFunction &mf, double a, double b,
                                  int jmax, std::vector<double> &E)
{
  E.clear();
  if (jmax < 0) return 1; // fail
  E.resize(jmax+1);
  int n = 1;
  double h = (b - a)/n;
  E[0] = 0.5*h*(mf.f(a) + mf.f(b));
  for (int j = 1; j <= jmax; ++j) {
    h /= 2.0;
    n *= 2;
    double sum = 0;
    for (int i = 1; i < n; i += 2) {
      double x = a + i*h;
      sum += mf.f(x);
    }
    E[j] = 0.5*E[j-1] + sum*h;
  }
  return 0; // success
}
```

6.10 Simpson's Rule

- Instead of a straight line, let us approximate the function by a quadratic.
- For a quadratic we require three points, say x_i, x_{i+1} and x_{i+2}.
- Note that this means we approximate the function over a subinterval of length $2h$.
- The Lagrange polynomial inside the interval $[x_i, x_{i+2}]$ is a quadratic. Using eq. (6.7.5) yields

$$\begin{aligned} L_{\text{quad}}(x) = f(x_i)\,&\frac{(x-x_{i+1})(x-x_{i+2})}{2h^2} \\ &- f(x_{i+1})\,\frac{(x-x_i)(x-x_{i+2})}{h^2} \\ &+ f(x_{i+2})\,\frac{(x-x_i)(x-x_{i+1})}{2h^2}\,. \end{aligned} \tag{6.10.1}$$

- As before, we approximate the value of the integral in the subinterval by integrating the Lagrange polynomial:

$$\int_{x_i}^{x_{i+2}} f(x)\,dx \simeq \int_{x_i}^{x_{i+2}} L_{\text{quad}}(x)\,dx\,. \tag{6.10.2}$$

Performing the integration yields the result

$$\int_{x_i}^{x_{i+2}} f(x)\,dx \simeq \frac{h}{3}\left[\,f(x_i) + 4f(x_{i+1} + f(x_{i+2})\,\right]. \tag{6.10.3}$$

- The way I have written the approximation, n must be an even number, so that each integration is over a subinterval of length $2h$.
- Summing over all the subintervals yields the following approximate value for the integral

$$I \simeq \frac{h}{3}\left[\,f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\,\right]. \tag{6.10.4}$$

We can collect terms as follows

$$I \simeq \frac{h}{3}\left[f(x_0) + f(x_n) + 4\sum_{i=1,3,\dots,n-1} f(x_i) + 2\sum_{i=2,4,\dots,n-2} f(x_i)\right] + O\left(\frac{|b-a|^5}{n^4}\,f''''\right). \tag{6.10.5}$$

The last term is an error estimate. The notation "f''''" denotes the value of $f''''(x)$ at some unknown point in the interval $a \le x \le b$.

- **The formula eq.** (6.10.5) **is called Simpson's rule.**

6.10.1 Code example

- Simpson's rule can be coded as follows.
- Simpson's rule is easy to code, offers good accuracy and is widely used in practice.
- We only display the new function in the math library.
- **Note that the value of n must be even, for Simpson's rule.**

```
class MathFunction {
                        // same as before
};

class MathLibraryCPP {
public:
  // new function
  static double Simpson(const MathFunction &mf, double a, double b, int n);

  // etc
};

double MathLibraryCPP::Simpson(const MathFunction &mf, double a, double b, int n)
{
  if (n <= 0) return 0;
  double h = (b - a)/n;
  double sum1 = mf.f(a) + mf.f(b);
  double sum2 = 0;
  for (int i = 1; i < n; i += 2) {
    double x = a + i*h;
    sum2 += mf.f(x);
  }
  double sum3 = 0;
  for (int i = 2; i < n; i += 2) {
    double x = a + i*h;
    sum3 += mf.f(x);
  }
  return (sum1 + 4.0*sum2 + 2.0*sum3)*h/3.0;
}
```

6.11 Newton-Cotes formulas

- The use of Lagrange interpolating polynomials of higher degree yields a whole class of numerical integration formulas, which go by the general name of **Newton-Cotes** formulas.
- The trapezoid rule is a two-point Newton-Cotes formula.
- Simpson's rule is a three-point Newton-Cotes formula.
- The other Newton-Cotes formulas are boring and will not be discussed.

6.12 Simpson's Rule: cubic polynomials

- Sometimes, one can approximate a complicated function by a polynomial (hopefully of relatively low degree). Then the integral of the original function can be approximated by integrating the polynomial instead.
- One of the features of Simpson's rule is that, even though it evaluates the function $f(x)$ at only three points (equally spaced) x_i, x_{i+1} and x_{i+2} (over an interval of length $2h$), it yields the ***exact value of the integral*** for all polynomials up to ***degree 3***.
- Naïvely we would expect that with function values at three equally spaced points, we would only obtain the exact result for polynomials up to degree two. However, because of mathematical cancellations, Simpson's rule works for polynomials up to degree three.
- A polynomial with only one nonzero term is called a **monic polynomial**. We can verify the above property of Simson's rule by evaluating the integrals of various monic polynomials. A general polynomial of degree three (a ***cubic*** polynomial) can be composed of a linear sum of the four monic polynomials $f_0(x) = 1$, $f_1(x) = x$, $f_2(x) = x^2$, $f_3(x) = x^3$. (Note that $f_0(x)$ is just a constant.) Hence we only have to evaluate the integrals of these four monic polynomials individually.
- The domain of integration $[a, b]$ can obviously be transformed to $[0, 2]$ by shifting and scaling the x coordinate. Hence, without loss of generality, we can set $h = 1$ and $x_0 = 0$, $x_1 = 1$ and $x_2 = 2$. The values of the integrals are

$$\int_{x_0}^{x_2} f_0(x)\,dx = \int_0^2 1\,dx = 2\,, \tag{6.12.1a}$$

$$\int_{x_0}^{x_2} f_1(x)\,dx = \int_0^2 x\,dx = 2\,, \tag{6.12.1b}$$

$$\int_{x_0}^{x_2} f_2(x)\,dx = \int_0^2 x^2\,dx = \frac{8}{3}\,, \tag{6.12.1c}$$

$$\int_{x_0}^{x_2} f_3(x)\,dx = \int_0^2 x^3\,dx = 4\,. \tag{6.12.1d}$$

- Let us evaluate the integrals using Simpson's rule. Then

$$\int_{x_0}^{x_2} f_0(x)\,dx = \frac{f_0(0) + 4f_0(1) + f_0(2)}{3} = \frac{1+4+1}{3} = 2\,, \tag{6.12.2a}$$

$$\int_{x_0}^{x_2} f_1(x)\,dx = \frac{f_1(0) + 4f_1(1) + f_1(2)}{3} = \frac{0+4+2}{3} = 2\,, \tag{6.12.2b}$$

$$\int_{x_0}^{x_2} f_2(x)\,dx = \frac{f_2(0) + 4f_2(1) + f_2(2)}{3} = \frac{0+4+4}{3} = \frac{8}{3}\,, \tag{6.12.2c}$$

$$\int_{x_0}^{x_2} f_3(x)\,dx = \frac{f_3(0) + 4f_3(1) + f_3(2)}{3} = \frac{0+4+8}{3} = 4\,. \tag{6.12.2d}$$

- Hence for all cubic polynomials, Simpson's rule yields the exact value of the integral over any finite domain $[a, b]$.

- Next let us evaluate the integral of the fourth degree monic polynomial $f_4(x) = x^4$. The value of the integral is

$$\int_{x_0}^{x_2} f_4(x)\,dx = \int_0^2 x^4\,dx = \frac{32}{5} = 6.4\,. \tag{6.12.3}$$

  Simpson's rule yields

$$\int_{x_0}^{x_2} f_4(x)\,dx = \frac{f_4(0) + 4f_4(1) + f_4(2)}{3} = \frac{0+4+16}{3} = \frac{20}{3} = 6.6666\cdots \neq 6.4\,. \tag{6.12.4}$$

  Hence for a fourth degree polynomial (a ***quartic*** polynomial), Simpson's rule yields only an approximate value. The same is true for all polynomials of higher degree.

- Newton-Cotes formulas of higher degree yield the exact values for the integrals of polynomials of higher degree, but this is not of interest to us.

- What ***is*** of interest to us is to employ points which are ***not equally spaced***. This will give us additional flexibility to obtain the exact values for the integrals of polynomials of higher degree. This will be done later.

6.13 Midpoint rule

- Before we rush to study more complicated formulas, let us consider the following very simple formula. It is even simpler than the trapezoid rule.
- The formula is called the **midpoint rule**.
- As before, we subdivide the interval $[a, b]$ into n equal subintervals each of length $h = (b-a)/n$.
- In the subinterval $[x_i, x_{i+1}]$, instead of drawing a straight line (or a quadratic, etc), we simply approximate the function by its value at the midpoint $m_i = (x_i + x_{i+1})/2 = x_i + \frac{1}{2}h$. Hence we compute $f(m_i)$, so the integral is approximated by

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \simeq h\,f(m_i)\,. \tag{6.13.1}$$

- This yields the simple formula

$$\begin{aligned} I = \int_a^b f(x)\,dx &\simeq h\,\big[\,f(m_0) + f(m_1) + \cdots f(m_{n-1})\,\big] \\ &= h\left[\sum_{i=0}^{n-1} f(m_i)\right] \quad + O\biggl(\frac{|b-a|^3}{n^2}\,f''\biggr)\,. \end{aligned} \tag{6.13.2}$$

  The last term is an error estimate. The notation "f''" denotes the value of $f''(x)$ at some unknown point in the interval $a \le x \le b$.
- The order of accuracy is about the same as the trapezoid rule, and the number of function evaluations is n as opposed to $n+1$.

6.13.1 Code example

- A simple code example to implement the midpoint rule is as follows
- It is a simple rule, very popular.
- We only display the new function in the math library.

```
class MathFunction {
                        // same as before
};

class MathLibraryCPP {
public:
  // new function
  static double midpoint(const MathFunction &mf, double a, double b, int n);

  // etc
};

double MathLibraryCPP::midpoint(const MathFunction &mf, double a, double b, int n)
{
  if (n <= 0) return 0;
  double h = (b - a)/n;
  double sum = 0;
  double x = a + 0.5*h;
  for (int i = 0; i < n; ++i) {
    sum += mf.f(x);
    x += h;
  }
  return sum*h;
}
```

6.14 Midpoint rule: improper integrals

- The midpoint rule can be useful to evaluate improper integrals where the function $f(x)$ diverges at the endpoints $x = a$ and/or $x = b$, but some mathematical theorem says that the value of the integral is finite. The trapezoid rule or Simpson's rule cannot be used because they require the function to be evaluated at both endpoints. However, the midpoint rule does not evaluate the function $f(x)$ at the endpoints, and so it will work.

- A simple example is $f(x) = 1/\sqrt{x}$, with $a = 0$. Then we know from calculus that

$$\int_0^b \frac{1}{\sqrt{x}}\,dx = \Big[2\sqrt{x}\,\Big]_0^b = 2\sqrt{b}\,. \tag{6.14.1}$$

  The value of the integral is finite even though $f(x) \to \infty$ as $x \to 0$. The midpoint rule computes the value of the integral as follows. Note that $a = 0$, $h = b/n$ and $m_i = (i + \frac{1}{2})h$, so

$$\int_0^b \frac{1}{\sqrt{x}}\,dx \simeq \frac{b}{n}\sum_{i=0}^{n-1} \frac{1}{\sqrt{(i+\frac{1}{2})h}}\,. \tag{6.14.2}$$

- More generally, the midpoint rule will work for a function $f(x) = g(x)/\sqrt{x}$, where $g(x)$ is a continuous function and finite at $x = 0$ and the domain of integration is $a = 0$ and $b > 0$.

- What if the function $f(x)$ diverges at an interior point c, where $a < c < b$, but a mathematical theorem says that the value of the integral is finite? We can split the integral into two domains $[a, c]$ and $[c, b]$ and employ the midpoint rule in each domain

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx\,. \tag{6.14.3}$$

  We compute each integral separately, using the midpoint rule. The number of subintervals need not be the same in the two domains.

- **What if the function $f(x)$ is finite but equals $0/0$ at an interior point c, where $a < c < b$?** Once again we can split the integral into two domains $[a, c]$ and $[c, b]$ and employ the midpoint rule in each domain

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx\,. \tag{6.14.4}$$

  We compute each integral separately, using the midpoint rule. This is exactly how one can evaluate the integral in eq. (6.2.1) using the midpoint rule:

$$I = \int_{-\pi}^{\pi} \frac{\sin(x)}{x}\,dx = \int_{-\pi}^{0} \frac{\sin(x)}{x}\,dx + \int_0^{\pi} \frac{\sin(x)}{x}\,dx\,. \tag{6.14.5}$$

- Note that if the value of the integral is *infinite*, the midpoint rule will not help. The midpoint rule is not a magic cure to numerically integrate all functions which diverge at the endpoints or interior points of the domain of integration.

6.15 Romberg integration: Part 1

- We can use the trapezoid rule to play a trick to obtain higher accuracy using only "small" values of n. This trick is called **Romberg integration.**
- **Strictly speaking, the formulas below are applicable only if $f(x)$ is infinitely differentiable for all $a \le x \le b$. (But even then there are caveats.)**
- **The matter will be discussed more fully in Lecture 6b (which is an advanced topic, not for examination).**
- The remainder term of the trapezoid rule has a nice property. Recall from eq. (6.8.7)

$$I = h\left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{f(x_n)}{2}\right] + O\biggl(\frac{|b-a|^3}{n^2}\,f''\biggr)\,. \tag{6.15.1}$$

- The trapezoid rule has the remarkable property that the remainder term has a simple form.
  The remainder term can be expressed as a series in powers of $1/n$.
  All the odd order terms (odd powers of $1/n$) vanish.
  There is also a known formula for the even order terms (even powers of $1/n$).
  The answer is

$$I = h\left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{f(x_n)}{2}\right] - c_2h^2 - c_4h^4 - c_6h^6 - \cdots \tag{6.15.2}$$

  The coefficients c_k, $k = 2, 4, \dots$ are proportional to numbers called **Bernoulli numbers B_k**

$$\begin{aligned} c_2 &= \frac{B_2}{2!}(f'(b) - f'(a))\,, \\ c_4 &= \frac{B_4}{4!}(f'''(b) - f'''(a))\,, \\ &\vdots \\ c_k &= \frac{B_k}{k!}(f^{(k-1)}(b) - f^{(k-1)}(a)) \qquad\qquad (k = 2, 4, 6, \dots)\,. \end{aligned} \tag{6.15.3}$$

- However, the formula eq. (6.15.2) looks mathematically frightening.
  In particular, what are the Bernoulli numbers?
- Hence let us analyze eq. (6.15.2) systematically.
  1. **The Bernoulli numbers are just constants.** They are a family of numbers just like the Fibonacci numbers, except that they are not all integers. We do not really need to know their values, just that they are constants.
  2. The other important feature of the series for the remainder term in eq. (6.15.2) is that the derivatives f', f''', etc. are all evaluated ***only at the endpoints*** a ***and*** b***.*** They are *not* evaluated at unknown interior points.
  3. **Hence the values of c_2, c_4, etc. in eq.** (6.15.2) **do not depend on the number of subintervals n, or equivalently on the value of h.** This is very important.

- We can therefore use eq. (6.15.2) to play the following trick, to obtain higher accuracy using only a "few" subintervals. Let the exact value of the integral be $I_{\rm ex}$. With an obvious notation, define the numerical approximation $I(h)$ with a stepsize h as

$$I(n) = h\left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{f(x_n)}{2}\right]. \tag{6.15.4}$$

Then from eq. (6.15.2),

$$I_{\rm ex} = I(h) - c_2h^2 - c_4h^4 - \cdots \tag{6.15.5}$$

Now use a stepsize $h/2$ and we obtain

$$I_{\rm ex} = I(\tfrac{1}{2}h) - \frac{c_2h^2}{4} - \frac{c_4h^4}{16} - \cdots \tag{6.15.6}$$

We use eqs. (6.15.5) and (6.15.6) to cancel the remainder term in h^2 as follows

$$\begin{aligned} 4I_{\rm ex} - I_{\rm ex} &= \left[4I(\tfrac{1}{2}h) - c_2h^2 - \frac{c_4h^4}{4} - \cdots\right] - \left[I(h) - c_2h^2 - c_4h^4 - \cdots\right] \\ 3I_{\rm ex} &= 4I(\tfrac{1}{2}h) - I(h) + \frac{3}{4}c_4h^4 + \cdots \end{aligned} \tag{6.15.7}$$

Therefore

$$I_{\rm ex} = \frac{4}{3}I(\tfrac{1}{2}h) - \frac{1}{3}I(h) + \frac{3}{4}c_4h^4 + \cdots. \tag{6.15.8}$$

- **Using $I(h)$ and $I(\frac{1}{2}h)$, we obtained a numerical approximation of accuracy $O(h^4)$.**

- This is the basic idea of Romberg integration: we form linear combinations of numerical approximations to cancel the error terms to obtain higher order accuracy.

- **Example:** As a simple but instructive demonstration, let us use $n = 1$, and for simplicity set $a = 0$ and $b = 2$. Then $h = (b - a)/n = 2$, so

$$I(2) = f(0) + f(2). \tag{6.15.9}$$

Next use $n = 2$, so

$$I(1) = \frac{1}{2}\big(f(0) + 2I(1) + f(2)\big). \tag{6.15.10}$$

Now compute the linear combination

$$\begin{aligned} \frac{4}{3}I(1) - \frac{1}{3}I(2) &= \frac{2}{3}\big(f(0) + 2I(1) + f(2)\big) - \frac{1}{3}\big(f(0) + f(2)\big) \\ &= \frac{1}{3}\big(f(0) + 4f(1) + f(2)\big). \end{aligned} \tag{6.15.11}$$

This is Simpson's rule!
Using Romberg integration, we have rederived Simpson's rule.

6.16 Romberg integration: Part 2

- The Romberg integration calculation can be systematized. To avoid a mess, we need some more notation. As with the extended trapezoid rule, we define a set of stepsizes

$$h_j = \frac{b-a}{2^j} \qquad (j = 0, 1, 2, \dots)\,. \tag{6.16.1}$$

- Then we define a set of numerical approximations $R(j,k)$ (R for "Romberg"). We begin with $k = 0$. The value of $R(j,0)$ is exactly $E(j)$ from the extended trapezoid rule in eq. (6.9.5)

$$R(j,0) = E(j) \qquad (j = 0, 1, 2, \dots)\,. \tag{6.16.2}$$

- This is good by itself; we are using interpolation to improve the accuracy as the alue of j increases. However, so far this is simply the extended trapezoid rule.

- Next, recall that we can construct linear combinations of the $E(j)$ to cancel the $O(h^2)$ remainder term, to obtain accuracy of $O(h^4)$. This is the $k = 1$ part of the $R(j,k)$ array:

$$R(j,1) = \frac{1}{3}\big(4R(j,0) - R(j-1,0)\big) \qquad (j = 1, 2, 3, \dots)\,. \tag{6.16.3}$$

  In the case of Simpson's rule, we derived above that (put $j = 1$)

$$R(1,1) = \frac{1}{3}\big(4R(1,0) - R(0,0)\big) = \frac{1}{3}\big(4E(1) - E(0)\big)\,. \tag{6.16.4}$$

  The same idea works for $j > 1$. (We must begin with $j = 1$.)

- Next, we can take linear combination of the $k = 1$ terms to cancel the $O(h^4)$ remainder term, to obtain accuracy of $O(h^6)$. This is the $k = 2$ part of the $R(j,k)$ array. We must have $j \ge 2$:

$$R(j,2) = \frac{1}{4^2-1}\big(4^2R(j,1) - R(j-1,1)\big) \qquad (j = 2, 3, 4, \dots)\,. \tag{6.16.5}$$

- The procedure continues to higher values of k. We must have $j \ge k$. We can equivalently say $k = 0, 1, \dots, j$:

$$R(j,k) = \frac{1}{4^k-1}\big(4^kR(j,k-1) - R(j-1,k-1)\big) \qquad (0 \le k \le j)\,. \tag{6.16.6}$$

  The accuracy of the approximation is $O(h^{2k+2})$.

- This is the Romberg integration scheme. It forms a triangular array of approximations, with $j = 0, 1, 2, \dots$ and $0 \le k \le j$ as follows

$$\begin{array}{llll}
R(0,0) & & & \\
R(1,0) & R(1,1) & & \\
R(2,0) & R(2,1) & R(2,2) & \\
R(3,0) & R(3,1) & R(3,2) & R(3,3) \\
\text{etc.} & & &
\end{array}$$

- Notice that only the left column ($k = 0$) contains actual integration (the extended trapezoid rule to compute $E(j)$). All the other columns ($k \ge 1$) contain only linear combinations to cancel the remainder terms.

- A simple code fragment to implement Romberg integration is as follows.
  We assume that sufficient memory has been allocated for the arrays `E[]` and `**Rom`.
  Alternatively one can use `std::vector<double>` and `std::vector<std::vector<double>>`.

```
// perform validation checks, jmax > 0 etc
int jmax = ...;  // input parameter
double E[...];   // array of length at least jmax
double **Rom;    // memory must be allocated

// extended trapezoid rule is called to compute E[j] for j=0, ..., jmax-1
// set Rom[0][0]
double *R0 = Rom[0];
R0[0] = E[0];

for (int j = 1; j < jmax; ++j) {
  double *tmp1 = Rom[j];
  double *tmp2 = Rom[j-1];
  tmp1[0] = E[j];
  for (int k = 1; k <= j; ++k) {
    double pow4k = (1 << (2*k));  // pow4k = 4^k
    tmp1[k] = (pow4k*tmp1[k-1] -tmp2[k-1]) /(pow4k-1.0);
  }
}
```