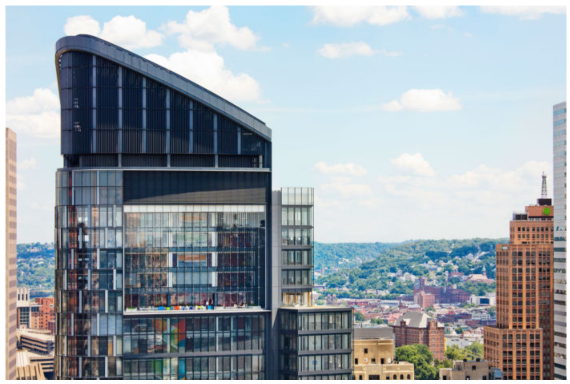
Quadrature:
The Trapezoid rule

MATH2070: Numerical Methods in Scientific Computing I

Location: http://people.sc.fsu.edu/~jburkardt/classes/math2070_2019/quadrature_trapezoid/quadrature_trapezoid.pdf



Trapezoids are not just for approximate integration! (Pittsburgh's PNC Tower)

Trapezoidal quadrature

Given a continuous function $f(x)$, and a domain $a \leq x \leq b$, estimate $I(f,a,b) \equiv \int_a^b f(x)\,dx$ using one or more trapezoids.

1 An estimate using one trapezoid

Suppose we want to estimate the integral $I(f,a,b)$ of a function $f(x)$ over the interval $[a,b]$, using a limited number of sample values. The trapezoid rule suggests the following approximation $T(f,a,b)$:

$$I(f,a,b) \approx T(f,a,b) = (b-a) * (f(a) + f(b))/2$$

It is not hard to see that this approximation is exact if the function $f(x)$ happens to be a constant or linear function. Otherwise, we know from our polynomial approximation result that we can compare $f(x)$ to the linear interpolant $p_2(x)$

$$f(x) - p_2(x) = \frac{f''(\xi)}{3!}(x-a)(x-b) \text{ for some } \xi \in [a,b]$$

and this implies, after integration, that

$$I(f,a,b) - T(f,a,b) = \frac{f''(\xi)}{12}(b-a)^3$$

This suggests that, for a fixed function $f(x)$, the quadrature error decreases cubically as $(b-a)$ decreases.

2 Example: Does the error drop cubically with interval size?

To verify the error behavior, we compare the exact and estimated integrals of $\int_{x1}^{x2} exp(x)\,dx$ as we repeatedly halve the size of the interval. In this case, we expect that at each step, the error will decrease by a factor $r = \frac{1}{8} = 0.125$.

```
 k      x1          x2          int           quad          error      rate

 0   -1.000000   1.000000   2.3504        3.08616       7.3e-01
 1   -0.500000   0.500000   1.04219       1.12763       8.0e-02   0.1161
 2   -0.250000   0.250000   0.505225      0.515707      1.0e-02   0.1227
 3   -0.125000   0.125000   0.250652      0.251956      1.3e-03   0.1244
 4   -0.062500   0.062500   0.125081      0.125244      1.6e-04   0.1249
 5   -0.031250   0.031250   0.0625102     0.0625305     2.0e-05   0.1250
 6   -0.015625   0.015625   0.0312513     0.0312538     2.5e-06   0.1250
 7   -0.007812   0.007812   0.0156252     0.0156255     3.1e-07   0.1250
 8   -0.003906   0.003906   0.00781252    0.00781256    3.9e-08   0.1250
 9   -0.001953   0.001953   0.00390625    0.00390626    4.9e-09   0.1250
10   -0.000977   0.000977   0.00195313    0.00195313    6.2e-10   0.1250
```

Listing 1: Output from decrease_h.m

3 Estimate the error using two trapezoids

Let the notation $T2(f,a,b)$ indicate that we are approximating the integral of $f(x)$ over $[a,b]$ using **two** trapezoids. Define $x_0 = a$, $x_1 = \frac{a+b}{2}$, $x_2 = b$, and write

$$\begin{aligned} T2(f,a,b) &= T1(f,x_0,x_1) + T1(f,x_1,x_2) \\ &= \frac{(x_1 - x_0)}{2} * (f(x_0) + f(x_1)) + \frac{(x_2 - x_1)}{2} * (f(x_1) + f(x_2)) \\ &= (x_2 - x_0) * (\frac{1}{2} f(x_0) + f(x_1) + \frac{1}{2} f(x_2))/2 \end{aligned}$$

We could estimate quadrature error as the difference

$$E(f,a,b,T1) = \int_a^b f(x)\,dx - T1(f,a,b) \approx T2(f,a,b) - T1(f,a,b)$$

but, as explained in Professor Layton's notes, more accurate estimates are:

$$\begin{aligned} E1(f,a,b,T1) &= \int_a^b f(x)\,dx - T1(f,a,b) \approx e1 = \frac{4}{3} * (T2(f,a,b) - T1(f,a,b)) \\ E2(f,a,b,T2) &= \int_a^b f(x)\,dx - T2(f,a,b) \approx e2 = \frac{1}{3} * (T2(f,a,b) - T1(f,a,b)) \end{aligned}$$

which reflects the expectation that using two trapezoids of half the width (T2) produces an estimate whose error is reduced by a factor of $\frac{1}{4}$ from the T1 estimate.

4 Example: Estimating the error

Consider again the problem of estimating the quadrature error when we are approximating $\int_a^b exp(x)\,dx$ using $T1(f,a,b)$. Let `E1, E2` represent the exact errors in `T1(f,a,b)` and T2(f,a,b), and let `e1, e2` stand for the corresponding error estimates. For a variety of values of $[a,b]$, we compute and compare the true and estimated errors:

```
  a        b         E1            e1            E2            e2

  0.00     0.20    -0.000738    -0.000737    -0.000184    -0.000184
  0.10     0.40    -0.002896    -0.002894    -0.000725    -0.000724
  0.20     0.60    -0.007988    -0.007983    -0.002001    -0.001996
  0.30     0.80    -0.018168    -0.018149    -0.004556    -0.004537
  0.40     1.00    -0.036575    -0.036520    -0.009185    -0.009130
  0.50     1.20    -0.067698    -0.067560    -0.017027    -0.016890
  0.60     1.40    -0.117846    -0.117535    -0.029695    -0.029384
  0.70     1.60    -0.195774    -0.195120    -0.049434    -0.048780
  0.80     1.80    -0.313488    -0.312198    -0.079339    -0.078050
  0.90     2.00    -0.487310    -0.484891    -0.123641    -0.121223
```

Listing 2: Output from t2_minus_t1.m

5 Using n trapezoids

It should be clear that the two trapezoid integral estimate is likely to be more accurate than when only one trapezoid is used, since the error estimate drops by a factor of 4. This suggests that we might be able to get further error reductions by repeatedly doubling the number of trapezoids. When a quadrature rule is used to estimate an integral by dividing it into subintervals and summing the integral estimates, this is known as a **composite rule**. For the trapezoid rule that uses $n+1$ equally spaced points $x_0, x_1, ..., x_n$, (and hence n trapezoids), the rule `Tn(f,a,b)` can be written simply as:

$$I(f,a,b) \approx Tn(f,a,b) = (b-a) \cdot (0.5 * f(x_0) + f(x_1) + ... + f(x_{n-1}) + 0.5 * f(x_n)) \,/\, n$$

The first factor represents the width of the interval. The second represents an estimated average value for $f(x)$ over that interval. This means that the coefficients of the sample values of $f(x)$ should add up to 1.

6 Exercise: A Composite trapezoid rule

Use a trapezoid rule `T8` to estimate the integral of the *hump()* function over the interval $[0,2]$. Use the following pseudocode as a guide.

```
a = 0.0
b = 2.0
n = 8
x = n+1 equally spaced values between a and b

q = 0.5 * fhumpx(1))
loop2 i = 2 to n
  q = q + hump ( x(i) )
end loop2
q = q + 0.5 * hump(x(n+1))

q = ( b - a ) * q / n

e = hump_int(a,b) - q
```

Listing 3: Pseudocode for T8 quadrature of hump().

7 Exercise: A Sequence of composite trapezoid rules

Use a sequence of trapezoid rules `T1, T2, T4, ..., T1024` to estimate the integral of the *hump()* function over the interval $[0,2]$. Use the following pseudocode as a guide.

```
a = 0
b = 2
q = 0
loop1 nlog = 0 to 10
  n = 2^nlog
  qold = q
  x = n + 1 equally spaced values between a and b
  q = 0.5 * f(x(1))
  loop2 i = 2 to n
    q = q + f ( x(i) )
  end loop2
  q = q + 0.5 * f(x(n+1))
  q = ( b - a ) * q / n
  e = hump_int(a,b) - q
  print n, q, e
end loop1
```

Listing 4: Pseudocode for sequence of trapezoid quadratures of hump().

Here's how your output should start:

```
     n      Tn(hump,a,b)            Error

     1     0.3212981744421901          29
     2      16.16064908722109        13.17
     4      16.17515212981744        13.15
    ...     ...                      ...
  1024      ?                        ?
```

Listing 5: First three results for hump_trap.m.

8 An algorithm for adaptive integral estimation

By comparing T1(f,a,b) and T2(f,a,b), we can estimate the quadrature error that we make. Suppose that we wish to estimate the integral of f over an interval $[a, b]$ with an error of no more than `tol`. We can do so adaptively, by breaking the interval up into a sequence of subintervals. But rather than being equal subintervals, we start at the left endpoint, and consider a "small" interval of width h, setting $x_0 = a, x_1 = a + h/2, x_2 = x_0 + h$. If the error estimate for $T2(f, x_0, x_2)$ is more than $\frac{tol*h}{b-a}$, we cut h in half and retry the step. If it is less than $\frac{tol*h}{b-a}$, we add this to our running estimate for the integral and prepare for the next step. If the error estimate is less than $\frac{8*tol*h}{b-a}$, we are actually justified in trying a stepsize of $2 * h$ on the next step, otherwise we use h again.

A version of such an adaptive quadrature scheme was discussed in class.

9 Pseudocode for adaptive trapezoid quadrature

```
pseudocode for adaptive quadrature

%  Set a small initial h

h = ( b - a ) / 100.0

n = 0
q = 0.0
x0 = a

```

```
Loop1 to estimate integral from x0 to x0 + h

 if b <= x0 exit with success

%  Estimate the integral and the error.
%  If the error is small enough, accept the estimate, and advance x0.
%  Otherwise, decrease h and try again.
%
  Loop2 to reduce h if necessary

    if n_max <= n ) exit with error

%  Don't go past b!

    if ( b < x0 + h )
      h = ( b - x0 )
      x1 = x0 + h / 2.0
      x2 = b
    else
      x1 = x0 + h / 2.0
      x2 = x0 + h

%  Compute integral and error estimates using 1 and 2 trapezoids.

    q1 = h * (       f(x0)         +       f(x2) ) / 2.0
    q2 = h * ( 0.5 * f(x0) + f(x1) + 0.5 * f(x2) ) / 2.0
    e1 = abs ( 4.0 * ( q2 - q1 ) / 3.0 )
    e2 = abs (       ( q2 - q1 ) / 3.0 )

%  Decide if h can be increased, or is about right, or needs to be reduced.

    if ( 8.0 * e2 <= tol * h / ( b - a ) )
      h = h * 2.0
      q = q + q2
      x0 = x2
      n = n + 1
      break
    elseif ( e2 <= tol * h / ( b - a ) )
      h = h
      q = q + q2
      x0 = x2
      n = n + 1
      break
    else
      h = h / 2.0
      if h too small then exit error

    end loop1

  end loop2

end
```

10 Example: Adaptive quadrature of *hump()* over [0,2]

Consider the quadrature of the *hump()* function. The function has a very sharp variation in [0.2,0.4] and a mild variation in [0.6,1.1]. We can imagine that the trapezoid rule would have some trouble around these bending areas. When we run a simple version of the adaptive code, we get a good estimate for the integral, and we can see that the program took smaller steps in the problem areas.

```
hump_adapt:
```

```
  Use adaptive trapezoid integration to estimate
  the integral of hump(x) from 0 to 2.

At x0 =  0.02, try smaller h = 0.01
At x0 =  0.12, try smaller h = 0.005
At x0 = 0.245, try bigger h = 0.01
At x0 =  0.245, try smaller h = 0.005
At x0 =  0.265, try smaller h = 0.0025
At x0 = 0.3475, try bigger h = 0.005
At x0 = 0.3575, try bigger h = 0.01
At x0 =  0.3675, try smaller h = 0.005
At x0 = 0.5425, try bigger h = 0.01
At x0 = 0.7725, try bigger h = 0.02
At x0 =  0.8325, try smaller h = 0.01
At x0 = 0.9925, try bigger h = 0.02
At x0 = 1.3925, try bigger h = 0.04
At x0 = 1.6725, try bigger h = 0.08
At x0 = 2, try bigger h = 0.015

  Number of subintervals = 185
  Integral estimate = 29.3281
  Exact integral =    29.3262
  Error =             0.00190283
  Error tolerance =   0.01
```

Listing 6: Output from hump_adapt.m

Although the adaptivity seemed to work reasonably well for *hump()*, the adaptivity would be much more necessary in cases where $f(x)$ was highly oscillatory, so that the curve cannot be well approximated by straight line segments unless they are very small.

11 Assignment #7

Consider the function

$$f(x) = e^x \sin(x)$$

over the interval $[a, b] = [0, 2\pi]$.

Write a program *hw7.m* which

1. Uses trapezoid rules $Tn(f, a, b)$ of order $n = 2^{nlog}$ for $nlog = 0, 1, 2, ..., 10$ to estimate the integral $I(f, a, b)$;
2. Evaluates the error $En(f, a, b) = I(f, a, b) - Tn(f, a, b)$ (Work out the formula for $I(f, a, b)$!);
3. Prints $n, Tn(f, a, b), En(f, a, b)$ for the 11 values of n;

Turn in: your file *hw7.m* by Friday, October 11.