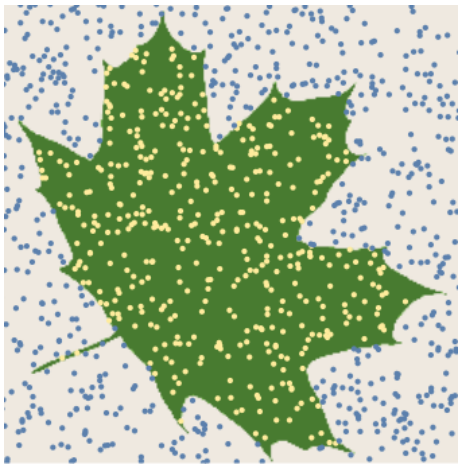
Quadrature:
A Variety of Topics

MATH2070: Numerical Methods in Scientific Computing I

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

---



A Monte Carlo estimate of the area of a maple leaf.

A Variety of Questions

- *What are some other common quadrature rules?*
- *How can random sampling be used for quadrature?*
- *How can an integral over a 2D domain be estimated?*
- *How can we integrate a function with a discontinuity of some kind?*

1 Simpson's rule

Simpson's rule is the 3-point member of the Newton-Cotes family of quadrature rules. The basic Simpson's rule improves the accuracy of the trapezoid rule by checking the midpoint:

$$S(f,a,b) = (b-a) * \frac{1}{6} * (f(a) + 4 * f(\frac{a+b}{2}) + f(b))$$

Just as with the trapezoidal rule, we can get an improved estimate of the integral by dividing $[a,b]$ into n subintervals (using $2*n+1$ points). Here is an example:

```
a = 0.0;
b = 2.0;
n = 10;
x = linspace ( a, b, 2 * n + 1 );

q = 0.0
for i = 1 : n
  q = q + hump(x(2*i-1)) + 4 * hump(x(2*i)) + hump(x(2*i+1));
```

```
9  end
10
11 q = ( b - a ) * q / 6.0 / n;
```

Listing 1: Simpson's quadrature rule for hump().

If you worry about such things, the summation can be replaced by a single efficient MATLAB statement:

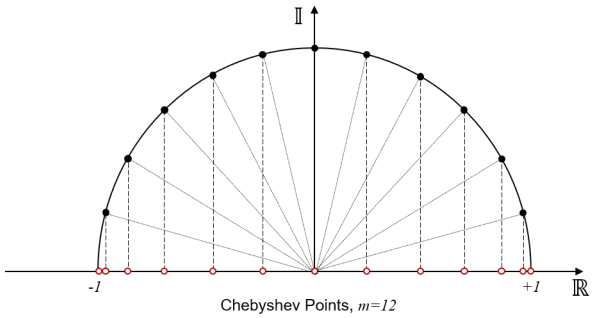
```
1   q =               hump(x(1) ) ...
2      + 4.0 * sum ( hump(x(2:2*n)) ) ...
3      + 2.0 * sum ( hump(x(3:2*n-1)) ) ...
4      +             hump(x(2*n+1));
```

Listing 2: Single statement Simpson summation.

The error using S1(f,a,b) is proportional to $(b-a)^5$; that is, if we apply the one step rule to an interval half the size, the error tends to be 1/32 as large. When we use a composite Simpson's rule $SN(f,a,b)$ on a fixed interval, and increase n, the error tends to decrease like h^4 where $h = \frac{(}{b} - a)n$.

2 Clenshaw-Curtis rule

Like the Gauss rules, a n-point Clenshaw-Curtis rule $Cn(f,a,b)$ uses a set of nodes that are not equally spaced, which tend to be more dense near the endpoints. The points and weights are easier to compute than for a Gauss rule.



Clenshaw-Curtis points are cosines of equally-spaced angles.

The polynomial exactness is less than for Gauss: the n-point Clenshaw-Curtis rule will exactly integrate polynomials up to degree $n-1$, but an n-point Gauss rule is exact for polynomials up to degree $2*n-1$.

However, it has been observed that sometimes this rule does as well as a Gauss rule when the integrand is something more complicated than a polynomial.

The function `clenshaw_curtis_m1p1(n)` returns a rule for $[-1,+1]$, while `clenshaw_curtis_ab(n,a,b)` works with $[a,b]$. As with other quadrature rules, a composite rule can be formed by breaking $[a,b]$ into subintervals.

3 A Monte Carlo rule

So far, we seem to have been very careful to choose the points and weights of our quadrature rules for efficiency and accuracy. Now we will see a rule that seems to break all the patterns we have followed. A Monte Carlo quadrature rule uses random sampling to estimate an integral. It has the form

$$MC(f,a,b) = (b-a) * \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

where the points x are chosen **randomly** within the interval $[a,b]$.

```
function hump_mc ( n )

  a = 0.0;
  b = 2.0;
  x = 2.0 * rand ( n, 1 );
  q_exact = hump_int ( a, b );

  q = ( b - a ) * ( 1.0 / n ) * sum ( hump(x) );

  return
end
```

Listing 3: Monte Carlo estimate of hump() integral.

4 A 16 point playoff

Suppose we wanted to estimate the integral of $f(x)$ over $[a,b]$, and we were allowed a budget of roughly 16 function evaluations. How do the various methods compare?
We can try to estimate the value of $\int_0^1 \frac{dx}{x^2+(1-x)^2}$ using full rules of order 16, or composite rules with a given number of subintervals. The results are:

```
Rule         Sub   Order   Error

Trapezoid     16     2     0.001302
Simpson        8     3     7.557e-08
Clenshaw       1    16     1.526e-09
Clenshaw       4     4     3.904e-06
Gauss          1    16     1.452e-12
Gauss          4     4     2.301e-08
Monte Carlo    1    32     0.06328
```

Listing 4: Results of playoff.

The full Gauss rule wins in this class, with the full Clenshaw-Curtis rule coming in a respectable second. The Monte-Carlo rule has very poor accuracy, but the Monte Carlo rule does not promise accuracy without the use of very many sample points.

5 Integrals involving hat functions

In class, we have discussed the notion of using "hat functions" as part of interpolation or approximation of data.
Hat functions play an important role in finite element analysis. Part of a finite element analysis involves computing an estimate Q of the value I of an integral of the form:

$$I(f*\phi_i(x),a,b) = \int_a^b f(x)\phi_i(x)\,dx \approx Q(f*\phi_i(x),a,b)$$

where $f(x)$ is some arbitrary (but presumably smooth) function, and $\phi_i(x)$ is the hat function associated with node x_i in the mesh:

$$a = x_0 < x_1 < ... < x_{i-1} < x_i < x_{i+1} < ... < x_n = b$$

Suppose we plan to use a 3-point Gauss rule to estimate the integral. But $G3(f*\phi_i(x),a,b)$, applying the rule over the whole interval, is a bad idea, since the integrand will be zero over most of $[a,b]$. A better idea is

$G3(f * \phi_i(x), x_{i-1}, x_{i+1})$. However, the hat function is not differentiable at x_i, and this means the accuracy of the Gauss rule will suffer. The best idea is to split the integral estimate over the two intervals:

$$Q(f * \phi_i(x), a, b) = G3(f * \phi_i(x), x_{i-1}, x_i) + G3(f * \phi_i(x), x_i, x_{i+1})$$

Here we estimate such an integral using a Gauss rule with ng points:

```
function hat_integral ( ng )

  a = 0.0;
  b = 2.0 * pi;
  n = 8;
  x = linspace ( a, b, n + 1 );

  i = 7;
  [ xg1, wg1 ] = gauss_ab ( ng, x(i-1), x(i) );
  [ xg2, wg2 ] = gauss_ab ( ng, x(i), x(i+1) );

  q = wg1' * ( f(xg1) .* hat ( x(i-1), x(i), x(i+1), xg1 ) ) ...
    + wg2' * ( f(xg2) .* hat ( x(i-1), x(i), x(i+1), xg2 ) );
```

Listing 5: Estimate integral involving a hat function.

Here, `hat(x1,x2,x3,x)` evaluates a hat function with the given nodes. If we are careful, we can write this as a one-statement formula, in such a way that it can be called with a vector x as input:

```
  function value = hat ( x1, x2, x3, x )
    value = ( x1 <= x & x < x2 ) .* ( x - x1 ) ./ ( x2 - x1 ) ...
          + ( x2 <= x & x < x3 ) .* (x3 - x  ) ./ ( x3 - x2 );
    return
  end
```

Listing 6: Evaluate hat function in vectorized way.

6 Product rules for product regions

A square or rectangle can be thought of as a product region of the form $[a, b] \times [c, d]$, and there are many other examples of product regions. In some cases, we can use our one-dimensional tools to construct corresponding quadrature rules for these higher-dimensional product regions.

Consider the task of estimating the integral of $f(x, y)$ over a rectangle $R = [a, b] \times [c, d]$. We might use Gauss rule of order $n_1 = 3$ in x and order $n_2 = 2$ in y to build a product Gauss rule:

$$G(n_1 \times n_2)(f(x, y), R) = G3(f(x, *), a, b) \times G2(f(*, y), c, d)$$

The new rule will have $n = n_1 * n_2 = 6$ points and weights. The points use every possible pairing of an x and a y node, and the weights will be the product of the corresponding 1D weights. If we denote the G3 and G2 weights by u and v, respectively, then we have:

$$G(n_1 \times n_2)(f(x, y), R) = (b - a) * (d - c) * \sum_{i=1}^{3} \sum_{j=1}^{2} u_i v_j f(x_i, y_j)$$

```
function rectangle_gauss ( n1, n2 )

  a = 1.0;
  b = 5.0;
  c = 2.0;
```

```
  d = 3.0;

  [ x, u ] = gauss_ab ( n1, a, b );
  [ y, v ] = gauss_ab ( n2, c, d );

  q = 0.0;
  for i = 1 : n1
    for j = 1 : n2
      q = q + u(i) * v(j) * f(x(i),y(j));
    end
  end

  fprintf ( 1, '  Using %d x %d points, integral estimate is %g\n', n1, n2, q );

  return
end
function value = f ( x, y )
  value = sin ( x ) .* exp ( x + y );
  return
end
```

Listing 7: Using a product Gauss rule on a rectangle.

7 MATLAB built-in functions `integral()` and `integral2()`

MATLAB has built-in functions of the form:

```
  q = integral ( f, xmin, xmax )
  q = integral2 ( f, xmin, xmax, ymin, ymax )
```

which numerically estimate the integral of $f(x)$ over the interval $[xmin, xmax]$ or the integral of $f(x, y)$ over the rectangle $[xmin, xmax] \times [ymin, ymax]$. We could check our previous Gauss estimate over the rectangle by

```
  q = integral2 ( @(x,y)f(x,y), 1.0, 5.0, 2.0, 3.0 );
```

The `integral2()` function also has some abilities to integrate over non-product regions such as a triangle, regions in which the y limit of integration is a function of x, and regions parameterized by polar coordinates.

8 Monte Carlo rules for irregular regions

An integral over a region can be thought of as the product of the area of the region times an "average" value of the integrand.

Obviously, we could get an average value of the integrand simply by randomly sampling n points. Suppose we wanted to estimate the integral of $\sin(x) * e^{x+y}$ over the unit circle. We could do that this way:

```
function circle_mc ( nmax )
  n = 0;
  q = 0.0;

  while ( n < nmax )
    x = rand();
    y = rand();
    if ( x^2 + y^2 <= 1.0 )
      n = n + 1;
      q = q + sin(x) * exp ( x + y );
    end
  end
```

```
13
14    q = pi * q / n;
15    fprintf ( 1, '  Using %d points, integral estimate is %g', n, q );
```

Listing 8: Estimate integral over circle.

This is known as a *rejection* method, since, in order to sample the circle, we start by sampling the unit square, and then reject those points that are outside the circle. There are usually methods that do a more efficient job of creating points that are guaranteed to be within your region.

Since the result will vary even if we call the function twice with the same value of *nmax*, we can try to get some confidence in the estimate by comparing the results as `nmax` is increased. We can also look at the variance in the results to see whether we are likely to be close to an accurate answer.

The circle is not a very irregular region, but this same Monte Carlo sampling idea can be applied to regions such as a polygon, or regions defined by some implicit mathematical constraint, as well as regions in higher dimensions.

9 Computing Assignment #9

We wish to approximate $\int_{y=0}^{1} \int_{x=-1}^{2} x\, e^{xy}\, dx\, dy$. The exact integral is $I = e^2 - e^{-1} - 3$. Write a function which uses an $m \times n$ Gauss product rule to return a value Q which estimates the integral. By experiment, find values of m and n so that the error $E = |I - Q| < 1.0e - 10$.

Turn in: your file *hw9.m* and your values of m and n by Friday, October 25.