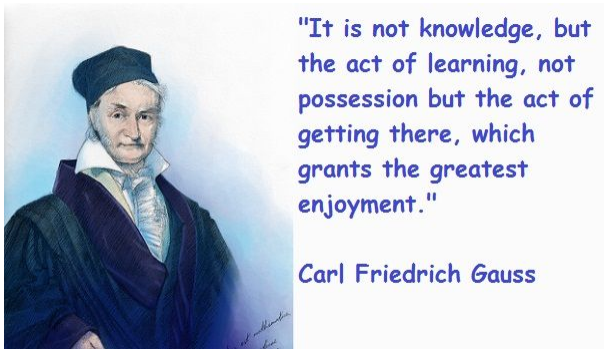
Quadrature:
Gauss rules

MATH2070: Numerical Methods in Scientific Computing I

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

---



Gauss advocates an adaptive approach to life.

Quadrature Using Gauss Rules

Estimate $I(f,a,b) \equiv \int_a^b f(x)\,dx$ *using an* n*-point Gauss rule* $Gn(f,a,b) = \sum_{i=1}^{n} \rho_i\, f(\xi_i)$

1 Gauss Quadrature Rules

While it may seem natural to use equally spaced sample points in the Newton-Cotes rules, this results in two disadvantages:

- these rules only achieve about half the precision possible;
- as n increases, half the weights become negative, and all large, so that the rule becomes unstable (significant loss of accuracy);

Gauss quadrature rules specify both the sample points and the weights; by doing so they essentially double the accuracy obtained by Newton-Cotes, and completely avoid the instability problem.

The table below lists points and weights for the first 4 Gauss rules. You should notice some interesting patterns in this table.

i	ξ	ρ
	n = 1	
1	0.0	2.0
	n = 2	
1	-0.57735026	1.0
2	+0.57735026	1.0
	n = 3	
1	-0.77459666	0.55555555
2	0.00000000	0.88888888
3	0.77459666	0.55555555
	n = 4	
1	-0.86113631	0.34785484
2	-0.33998104	0.65214515
3	+0.77459666	0.65214515
4	+0.86113631	0.34785484

2 Exercise: Verify properties of Gauss Rules

A Gauss rule of order n consists of n abscissas ξ_i and n weights ρ_i, such that:

1. Each ξ_i satisfies $-1 < \xi_i < 1$, $\xi_i = -\xi_{n+1-i}$, and $\sum \xi_i = 0$;
2. Each ρ_i satisfies $0 < \rho_i$, $\rho_i = \rho_{n+1-i}$, and $\sum \rho_i = 2$;
3. $I(f,-1,+1) \approx Gn(f,-1,+1) \equiv \sum \rho_i f(\xi_i)$;
4. $I(f,-1,+1) = Gn(f,-1,+1)$ **exactly**, if $f(x)$ is any polynomial of degree $2n-1$ or less;

The function `[ xi, rho ] = gauss ( n )` returns the points and weights of an n-point Gauss quadrature rule $Gn(f,-1,+1)$. We can use this function to verify some of the properties for a particular value of n:

```
n = 7
[ xi, rho ] = gauss ( n )

xi_sum = sum ( xi )
rho_sum = sum ( rho )
xi_sym = xi(1:n) + xi(n:-1:1)
rho_sym = rho(1:n) - rho(n:-1,1)

f = @(x)cos(x);
f_anti = @(x)sin(x);
gn = rho' * f(xi)
in = f_anti(+1.0) - f_anti(-1.0)

f = @(x) 13*x.^12-6*x.^5+5*x.^4-7;
f_anti = @(x) x.^13-x.^6+x.^5-7*x;
gn = rho' * f(xi)
in = f_anti(+1.0) - f_anti(-1.0)
```

Listing 1: gauss_properties.m

3 Example: Integrate quartic() over $[-1,+1]$

The *quartic(x)* function is:

$$f(x) = 2x^4 - 4x^2 + x + 20$$

The function *quartic_int(a,b)* returns the value of the definite integral of $f(x)$ over $[a,b]$. We can use these functions to test out a 2-point Gauss rule:

```
n = 2;

q_exact = quartic_int ( -1.0, +1.0 );

[ xi, rho ] = gauss ( n );
q_gauss = rho' * quartic(xi);

e = q_exact - q_gauss
fprintf ( 1, ' %g  %g  %g\n', q_exact, q_gauss, e );
```

Listing 2: Two-point Gauss quadrature integral estimate.

We can make a useful function `quartic_gauss_m1p1()` that lets us vary the number of points, so we can see how the error behaves as we increase n:

```
function quartic_gauss_m1p1 ( n )

  a = -1.0;
  b = +1.0;
  q_exact = quartic_int ( -1.0, +1.0 );

  [ xi, rho ] = gauss ( n );
  q_gauss = rho' * quartic(xi);

  e = q_exact - q_gauss;

  return
end
```

Listing 3: quartic_gauss_m1p1.m

Try this function for $n = 1, 2, 3$; explain what has happened to the error when we reach the 3-point rule.

4 Example: Remapping a Gauss Rule

The Gauss rule is set up for integrals over $[-1,+1]$. What happens if we want to integrate over $[a,b]$ instead? To generate the correspond rule, $Gn(f,a,b)$, the values ξ and ρ must be transformed to new values x and w, as follows:

$$x \leftarrow a + \frac{b-a}{2}(\xi + 1)$$
$$w \leftarrow \frac{b-a}{2}\rho$$

The program *gauss_ab()* can do this for us automatically:

```
function [ x, w ] = gauss_ab ( n, a, b )
  [ xi, rho] = gauss ( n );
  x = a + ( xi + 1.0 ) * ( b - a ) / 2.0;
  w = ( b - a ) * rho / 2.0;
  return
end
```

Listing 4: gauss_ab returns a Gauss rule for a general interval.

The values x will lie in $[a,b]$, and the w values will add up to $b-a$. The

5 Exercise: Integrate quartic() over $[2, 5]$

Now we can try using a Gauss rule to estimate an integral over a general interval:

```
n = 2;
a = 2.0;
b = 5.0;
[ x, w ] = gauss_ab ( n, a, b );
q_gauss = w' * quartic(x);
```

Listing 5: Estimate the integral of a quartic polynomial over a general interval.

Try these commands, and compare your quadrature result with the exact value returned by `quartic_int(a,b)`. Because the function is a polynomial of degree 4, what value of n should allow us to compute the exact value of the integral?

6 Exercise: Integrate e^{-x^2} over $[0, b]$

We know that $\int_0^b e^{-x^2}\,dx = 0.5 * \sqrt{\pi}\,\mathrm{erf}(b)$, where `erf(x)` is the standard mathematical shorthand for the **error function**. The corresponding MATLAB function is also called `erf(x)`. The integrand e^{-x^2} is not a polynomial, so we cannot expect that Gauss quadrature will return an exact estimate, no matter how many points we use.

Estimate the value of $\int_0^3 e^{-x^2}\,dx$. Using your knowledge of the exact integral, seek a value of n large enough that your absolute error is less than 1.0E-10.

```
n = ?;
a = 0.0;
b = 3.0;
[ x, w ] = gauss_ab ( n, a, b );
q_gauss = ?
q_exact = ?
e = abs ( q_exact - q_gauss )
```

Listing 6: Estimating an exponential integral.

7 Example: Estimate the quadrature error for hump(x) over $[0, 2]$

The hump() function is not a polynomial, involves rational functions, and has a strong variation over the domain. We expect that a Gauss rule can drive the error to zero, but we don't know, in advance, what n to use.

Pretend that we don't actually know the value of the integral. Then if we compute a sequence of estimates for increasing values of n, we can estimate the error as the difference between our current integral estimate and the previous one. If we have been given an error tolerance, then we can stop when our estimated error is less than this tolerance.

```
tol = 0.1

a = 0.0;
b = 2.0;

n = 0;
q = 0.0;

while ( true )
  q_prev = q;
```

```
    n = n + 1;
    [ xi, rho ] = gauss ( n );
    x = a + ( xi + 1 ) * ( b - a ) / 2.0;
    w = rho * ( b - a ) / 2.0;
    q = w' * hump ( x );
    err_est = abs ( q - q_prev );
    if ( err_est < tol )
      break
    end
  end
  fprintf ( 1, '  %2d  %16.12f  %e\n', n, q, err_est );
```

What is the least value of n that we need so that the estimated error is less than 1.0E-10?

8 A composite Gauss rule

We have seen that we can seek higher accuracy by increasing the value of n in a Gauss rule. This approach can break down if the integrand function does not have $2*n-1$ continuous derivatives. Also, as n gets very large (say, beyond 50) the Gauss nodes near the endpoints begin to get too close together, and there can be a loss of accuracy. An alternative is to use a composite rule, that is, to divide the interval $[a,b]$ into m equal subintervals, apply an n-point Gauss rule on each, and sum the results. This is known as a **composite Gauss rule**.

9 Exercise: Integrate $\frac{1}{2+\cos(x)}$ over $[0,2\pi]$

We know that $\int_0^{2\pi} \frac{1}{2+\cos(x)}\,dx \approx 3.62759872846844$. Because the integrand involves a rational function, and the cosine, we expect that convergence may be slow.

```
function [ q, e ] = cosfrac_composite ( m, n )

  a = 0.0;
  b = 2.0 * pi;
  q = 0.0;

  s = linspace ( a, b, m + 1 );

  for i = 1 : m
    [ x, w ] = gauss_ab ( n, s(i), s(i+1) );
    qs = w' * cosfrac ( x );
    q = q + qs;
  end

  e = cosfrac_int() - q;

  return
end
```

Listing 7: Estimate cosine fraction integral.

Let's say we have a "budget" of 30 function evaluations. Then we can try every combination of m and n such that $m \times n = 30$. If we run the code for each of these cases, we get a surprising variation in the results!

10 No Computing Assignment for this Lab!