1 Illustrations of numerical integration

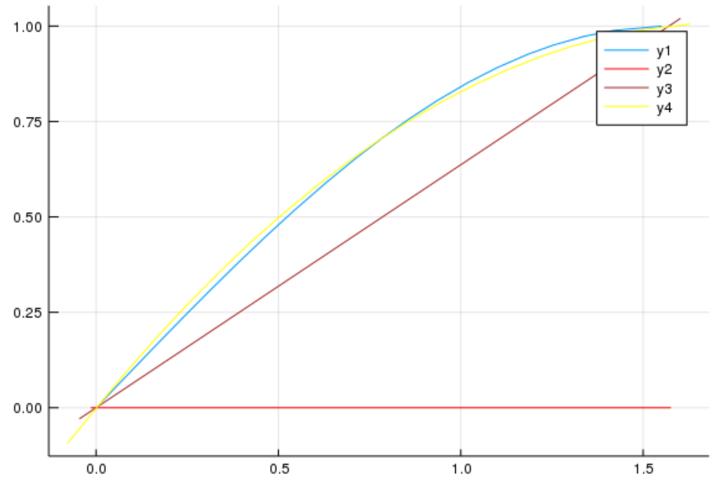
1.1 Newton Cotes formulas and errors

A Newton-Cotes formula uses an interpolating polynomial over $[a, b]$ to estimate f and in turn the integral of f over $[a, b]$. The nodes are evenly spaced, e.g.: `a`, `a,b`, `a,(a+b)/2, b`, ...

```
linspace(a,b,n=251) = range(a,stop=b, length=n)
function interpolating_nodes(a, b, n)
  n == 0 && return [a]
  collect(linspace(a,b,n+1))
end

function l(i, nodes)
  length(nodes) == 1 && return(x -> 1.0)
  x -> begin
    prod((x-nodes[j])/(nodes[i]-nodes[j]) for j in eachindex(nodes) if i
  end
end

function poly_interp(f, nodes)
  x -> sum(f(nodes[i]) * l(i, nodes)(x) for i in eachindex(nodes))
end

function quadrature(f, a, b, nodes)
        As = [quadgk(l(i, nodes), a, b)[1] for i in eachindex(nodes)]
        sum(f(nodes[i]) * As[i] for i in eachindex(nodes))
end

function newton_cotes(f, a, b, n)
  nodes = interpolating_nodes(a, b, n)
  quadrature(f, a, b, nodes)
end
```

```
newton_cotes (generic function with 1 method)
```

```
using Plots, QuadGK
f = sin
a, b = 0, pi/2
plot(f, a, b)
plot!(poly_interp(f, interpolating_nodes(a, b, 0)), color=:red)
plot!(poly_interp(f, interpolating_nodes(a, b, 1)), color=:brown)
plot!(poly_interp(f, interpolating_nodes(a, b, 2)), color=:yellow)
```



How accurate for the sine function

```
quadgk(f, a, b)   # 1.0
```

```
(0.9999999999999999, 1.1102230246251565e-16)
```

```
[newton_cotes(f, a, b, i) for i in 0:6] .- 1.0
```

```
7-element Array{Float64,1}:
 -1.0
 -0.2146018366025517
  0.0022798774922103693
  0.0010049233142788816
 -8.434527007272763e-6
 -4.7386138333216365e-6
  2.5837235240189216e-8
```

Should be exact for polynomials of degree n or less but not necessarily more:

```
a, b = 0, 1
function err(n)
  fn = x -> x^n  # x-> x^(n+1)
  nodes = interpolating_nodes(a, b, n)
  p = poly_interp(fn, nodes)
  newton_cotes(p, a, b, n) - quadgk(fn, a, b)[1]
end
[err(n) for n in 0:6]
```

```
7-element Array{Float64,1}:
  0.0
  0.0
  5.551115123125783e-17
  5.551115123125783e-17
 -2.7755575615628914e-17
 -2.7755575615628914e-17
 -5.551115123125783e-17
```

1.2 Gauss quadrature

Legendre polynomials satisfy $P_0 = 1$, $P_1(x) = x$, and $(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$.

```
using SymPy
@vars x
ps = Sym[1, x]
for n = 1:5
   pn, pn_1 = ps[end], ps[end-1]
   p =( (2n+1) * x*pn - n*pn_1 ) * (1// (n+1))
   push!(ps, simplify(p))
end
ps
```

$$\begin{bmatrix} 1 \\ x \\ \frac{3x^2}{2} - \frac{1}{2} \\ \frac{x\left(5x^2-3\right)}{2} \\ \frac{35x^4}{8} - \frac{15x^2}{4} + \frac{3}{8} \\ \frac{x\left(63x^4-70x^2+15\right)}{8} \\ \frac{231x^6}{16} - \frac{315x^4}{16} + \frac{105x^2}{16} - \frac{5}{16} \end{bmatrix}$$

We were told these were *orthogonal*:

```
w = 1
[integrate(ps[i] * ps[j] * w, (x, -1, 1)) for i in eachindex(ps), j in e
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We were told that these give exact quadrature for polynomials in Π_{2n+1}.

```
n = 5
a,b = -1, 1
```

```
pn = ps[n+1]  # 1 - based
nodes = solve(pn)  # solve p(x) == 0
```

$$\left[\begin{array}{r} 0 \\ -\sqrt{-\frac{2\sqrt{70}}{63}+\frac{5}{9}} \\ \sqrt{-\frac{2\sqrt{70}}{63}+\frac{5}{9}} \\ -\sqrt{\frac{2\sqrt{70}}{63}+\frac{5}{9}} \\ \sqrt{\frac{2\sqrt{70}}{63}+\frac{5}{9}} \end{array}\right]$$

```
function err(i)
   fn = x -> x^i
   Fn = x -> x^(i+1)/(i+1)
   quadrature(fn, a, b, N.(nodes)) - (Fn(b) - Fn(a))
end
n = length(nodes) - 1
[err(i) for i in 0:2n+1]
```

```
10-element Array{Float64,1}:
 -2.220446049250313e-16
 -1.1102230246251565e-16
  0.0
 -5.551115123125783e-17
  0.0
 -5.551115123125783e-17
  0.0
 -5.551115123125783e-17
  0.0
 -4.163336342344337e-17
```

But 10th degree polys are not necessarily exact:

```
fn = x -> x^10
Fn = x -> x^11/11
quadrature(fn, a, b, N.(nodes)) - (Fn(b) - Fn(a))
```

```
-0.002931812455622018
```

1.3 Error

Thm 4 on p497 has: if f is in $C^{2n}([a,b])$ where $g(x)$ is of degree n (so that there are n nodes) then (note $n-1$):

$$E = \int_a^b f(x)w(x)dx - \sum_{i-1}^{n-1} f(x_i)A_i = \frac{f^{(2n)}(\xi)}{(2n)!}\int_a^b q^2(x)w(x)dx = \frac{f^{(2n)}(\xi)}{(2n)!}\langle q,q\rangle_w$$

Here $q(x) = \Pi(x - x_i)$.

For this $n = 5$ we have the exact integral

```
q = prod(x-xi for xi in nodes)
integrate(q*q*w, (x, a, b))
```

$$\frac{128}{43659}$$

So for $f(x) = x^{10}$ we have $f^{(10)}(\xi)/10! = 1$ and so the error is

```
float(integrate(q*q*w, (x, a, b)))
```

```
0.0029318124556219796
```

Which matches what was previously found.

1.3.1 Simpson's error

If we used 5 points and simpson's formula, then we would apply simpsons over x_0, x_1, x_2 and x_2, x_3, x_4. How accurate would that be?

```
nodes = N.(nodes) # make floating point
quadrature(fn, nodes[1], nodes[3], nodes[1:3]) +
quadrature(fn, nodes[3], nodes[5], nodes[3:5]) - (Fn(1) - Fn(-1))
```

```
-0.118219124196878
```

So quite far off by comparison