Day 22. Quadrature, cont'd

culminating in adaptive quadrature with Gauss-Kronrod rules

Recall that we wish to accurately approximate $\int_a^b f$

in the common situation where we do not have access to an antiderivative F, F' = f,
hence cannot used the Fund. Thm. of Calculus.

We will use a "quadrature rule"

$$Q_{[a,b]}(f) = q\left(\{f(x_j)\}\right) = \sum_{j=0}^{m} \omega_j f(x_j) = (b-a)\sum_{j=0}^{m} \alpha_j f(x_j)$$

A quadrature rule is said to be "of polynomial degree m" if

$$Q_{[a,b]}(p) = \int_a^b p \quad \forall\, p \in P_m \,.$$

If the m+1 $\{x_j\}_{j=0}^m$ are already chosen and distinct,
then the requirement of Q being of polynomial degree m consitutes m+1 conditions on the m+1 weights $\{\alpha_j\}_{j=0}^m$
and we can determine those weights in this way.

$$Q(1) \overset{\text{require}}{=} \int_a^b 1\,dx \qquad (b-a)\sum_j 1\,\alpha_j = b-a$$

$$Q(x) = \int_a^b x\,dx \qquad (b-a)\sum_j x_j\alpha_j = \frac{b^2}{2} - \frac{a^2}{2}$$

$$Q(x^2) = \int_a^b x^2\,dx \qquad (b-a)\sum_j x_j^2\alpha_j = \frac{b^3}{3} - \frac{a^3}{3}$$

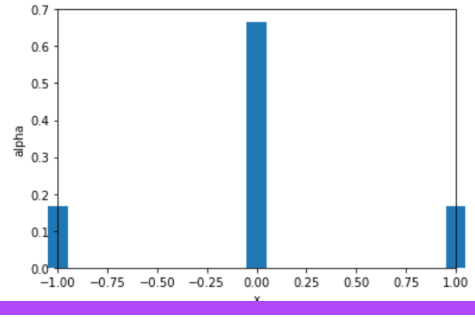
$$\vdots$$

$$Q(x^m) = \int_a^b x^m\,dx \qquad (b-a)\sum_j x_j^m\alpha_j = \frac{b^{m+1}}{m+1} - \frac{a^{m+1}}{m+1}$$

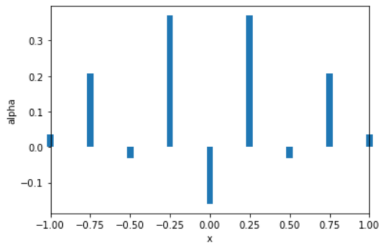
m+1 linear equations in m+1 variables $\{\alpha_j\}$.
Since the matrix is non-singular for distinct {x_j},
with m+1 sample points we can achieve polynomial degree at least m.

Let's look at some examples ...

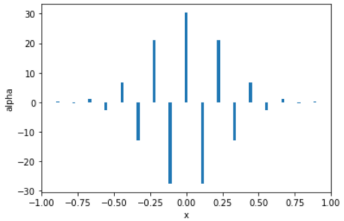
```
m = 2
closed Newton-Cotes  x =  [-1 0 1]

alpha = (1/6, 2/3, 1/6)

quadrature rule          has polynomial degree m   = 2 by construction
quadrature rule actually has polynomial degree m+1 = 3?  True
quadrature rule actually has polynomial degree m+2 = 4?  False
```



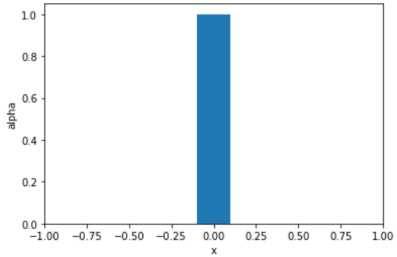

```
m = 8
closed Newton-Cotes  x =  [-1 -3/4 -1/2 -1/4 0 1/4 1/2 3/4 1]

alpha = (989/28350, 2944/14175, -464/14175, 5248/14175, -454/2835, 5248/14175, -464/14175, 2
944/14175, 989/28350)

quadrature rule          has polynomial degree m   = 8 by construction
quadrature rule actually has polynomial degree m+1 = 9?  True
quadrature rule actually has polynomial degree m+2 = 10?  False
```

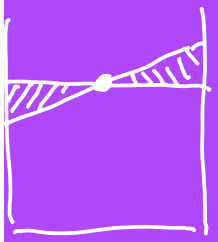



```
m = 18
closed Newton-Cotes  x =  [-1 -8/9 -7/9 -2/3 -5/9 -4/9 -1/3 -2/9 -1/9 0 1/9 2/9 1/3 4/9 5/9
2/3 7/9
 8/9 1]

alpha = (203732352169/15209113920000, 6162434073/50697046400, -214182958293/1013940928000, 1
61769065751/158428270000, -176535961191/63371308000, 308573105553/45265220000, -23648622603
3/18106088000, 664657884333/31685654000, -69854658519033/2534852320000, 11533183608517/38022
7848000, -69854658519033/2534852320000, 664657884333/31685654000, -236486226033/18106088000,
308573105553/45265220000, -176535961191/63371308000, 161769065751/158428270000, -21418295829
3/1013940928000, 6162434073/50697046400, 203732352169/15209113920000)

quadrature rule          has polynomial degree m   = 18 by construction
quadrature rule actually has polynomial degree m+1 = 19?  True
quadrature rule actually has polynomial degree m+2 = 20?  False
```



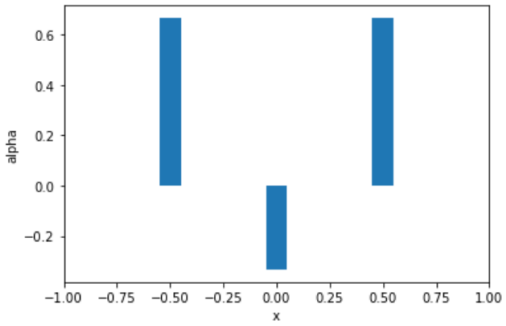

```
m = 0
open Newton-Cotes  x =  [0]

alpha = (1,)

quadrature rule          has polynomial degree m   = 0 by construction
quadrature rule actually has polynomial degree m+1 = 1?  True
quadrature rule actually has polynomial degree m+2 = 2?  False
```





Large alternating positive and negative weights can lead to catastrophic loss of significance due to subtraction of near-equals.

```
m = 2
open Newton-Cotes  x =  [-1/2 0 1/2]

alpha = (2/3, -1/3, 2/3)

quadrature rule          has polynomial degree m   = 2 by construction
quadrature rule actually has polynomial degree m+1 = 3?  True
quadrature rule actually has polynomial degree m+2 = 4?  False
```



We see the results with uniformly spaced nodes are not terribly satisfactory,
especially in regard to having negative weights.

But who said the nodes should be uniformly spaced?

How about considering node locations to be another set of degrees of freedom?
Then we'd have 2m+2 variables:
perhaps we could then achieve polynomial degree 2m+1 !?

Turns out this IS possible, and the corresponding optimal rules are called Gaussian quadrature.

Example: the midpoint rule, aka 1-point open Newton-Cotes, is also the 1-point Gaussian rule:

$m=0$, $x_0 = \frac{a+b}{2}$, $\alpha_0 = 1$ has polynomial degree $2 \cdot 0 + 1 = 1$. $= m+1$.

And the 4-point Gaussian rule has degree 2(3)+1 = 7. Wow!!

Let's develop the 2-point (m=1) Gaussian quadrature rule on [-1,1],
by requiring that it has polynomial degree 2m+1 = 3.

$$G_2(f) = \omega_0 f(x_0) + \omega_1 f(x_1)$$

It seems reasonable to guess that by symmetry, $x_0 = -x_1$ and $\omega_0 = \omega_1$.

Then, requiring exact results for each element of the monomial basis for P_3,
the constraints are:

$$G_2(1) = \omega_1 \cdot 1 + \omega_2 \cdot 1 = 2\omega_1 \overset{\text{need}}{=} \int_{-1}^{1} 1\,dx = 2 \Rightarrow \boxed{\omega_1 = 1,\ \omega_0 = 1}$$

So $G_2(f) = f(-x_1) + f(x_1)$.

$$G_2(x) = -x_1 + x_1 = 0 \overset{\text{need}}{=} \int_{-1}^{1} x\,dx = 0 \quad \checkmark \text{ ok regardless of } x_1$$

$$G_2(x^2) = (-x_1)^2 + x_1^2 = 2x_1^2 \overset{\text{need}}{=} \int_{-1}^{1} x^2\,dx = \frac{2}{3} \Rightarrow x_1^2 = \frac{1}{3},\ x_1 = \frac{1}{\sqrt{3}}.$$

Thus $\boxed{G_2(f) = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)}$.

And $G_{\frac{1}{\sqrt{3}}}(x^3) = -x_1^3 + x_1^3 = 0$ need $\int_{-1}^{1} x^3\,dx = 0$ ✓ 

Let's compare with the inferior 2-point rule called the "trapezoid rule":

$$T(f) = f(-1) + f(1).$$

Let's test the 2-point Gaussian quadrature rule we just derived

(to integrate a non-polynomial function)

```
from numpy import sqrt, exp
```

```
def G2(f):
    return f(-1/sqrt(3)) + f(1/sqrt(3))

def T(f):
    return f(-1) + f(1)

myf = exp         # here is the function we want to integrate

e = exp(1)
exact = e - 1/e

gauss_approx = G2(myf)
trap_approx  = T(myf)

gauss_error = (gauss_approx-exact)/exact
trap_error  = (trap_approx -exact)/exact

print(trap_error)
```

```
0.31303528549933135
```

```
print(gauss_error)
```

```
-0.003278714921135362
```

Gauss is 100 times better than Trapezoid in this example.

How to obtain the Gauss rules for larger m?

Ingredients are a set of polynomials (p0, p1, p2, ...) with the following properties:

(1) pn has degree n
(2) pi is orthogonal to pj if i does not equal j, with respect to the inner product $(p_i, p_j) = \int_{-1}^{1} p_i p_j$
(3) pn has n distinct roots in [-1,1]

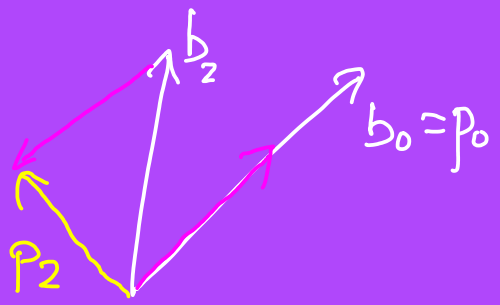
We could develop such a set as follows (choose some basis and orthogonalize it using Gram-Schmidt):

Say, let $p_0(x) = 1$

$p_1(x) = x$

$p_2(x) = x^2$ nope, not orthogonal to p_0

$= x^2 - \frac{1}{3}$



etc., obtiaining the "Legendre polynomials".

A convenient representation of the Legendre polynomials (I claim) is

$$P_n(x) = \frac{1}{2^n n!}\left(x^2-1\right)^{n^{(n)}}$$

Does this satisfy properties (1),(2),(3) ?

(1) pn is of degree n ? Yes! It's a polynomial of degree 2n diff'd n times.

(2) Orthogonality ?

$$\int_{-1}^{1} P_i P_j = \int_{-1}^{1} \left(x^2-1\right)^{i^{(i)}} \left(x^2-1\right)^{j^{(j)}} dx, \quad i \neq j \ \& \ \text{wolog } i > j.$$

Integrating by parts: $\int u'v = uv\Big| - \int uv'$

$$\int_{-1}^{1} P_i P_j = \underbrace{\left(x^2-1\right)^{i^{(i-1)}} \left(x^2-1\right)^{j^{(j)}}\Big|_{-1}^{1}}_{=0 \text{ because } (x^2-1)^i \text{ has a multiple root of order } i \text{ at } x=\pm 1.} - \int_{-1}^{1} \left(x^2-1\right)^{i^{(i-1)}} \left(x^2-1\right)^{j^{(j+1)}}$$

$$= -\int_{-1}^{1} \left(x^2-1\right)^{i^{(i-1)}} \left(x^2-1\right)^{j^{(j+1)}} dx$$

$\vdots$ $j+1$ times in all

$$= \pm \int_{-1}^{1} \left(x^2-1\right)^{i^{(i-(j+1))}} \underbrace{\left(x^2-1\right)^{j^{(2j+1)}}}_{\equiv 0} dx = 0 \ \checkmark \ \text{☺}$$

(3) n distinct roots in [-1,1] ? ... Yes: Google it > Stack Exchange

So what ? We will show that if

> we use the roots of p_(m+1) as the nodes, and
> choose the weights to obtain degree m

we will in fact have a rule of degree 2m+1 - the Gauss(-Legendre) rule.

Let P be a pdy of deg $2m+1$.

Can always express P as $P = S p_{m+1} + R$

where S, R are polynomials, $\deg S = m < m+1$, $\deg R < m+1$

See example below

$$p_{m+1} \overline{)\, P} \quad S \qquad R$$

Observation (1)

$G(P) = G(R)$ Why?

$G(P) = G(S\, p_{m+1}) + G(R)$

$= 0$ because sample points of G are precisely the zeros of p_{m+1}.

Observation (2)

$G(R) = \int R$ Why?

R has deg $\le m$ and $G(p) = \int p \;\; \forall\, p \in P_m$.

Observation (3)

$\int R = \int P$ Why?

$\int P = \int S p_{m+1} + \int R$

0 because S, with deg m, is orthogonal to p_{m+1}.

Summarizing,

$$G(P) \overset{(1)}{=} G(R) \overset{(2)}{=} \int R \overset{(3)}{=} \int P \quad \checkmark$$

Example: $m=1$, $m+1=2$, $2m+1=3$ $\qquad \deg R < m+1$

$P(x) = x^3 + 4x^2 + 3x + 5$

$P_2(x) = x^2 - \frac{1}{3}$

$$\begin{array}{r|l} & x + 4 \\ \hline x^2 - \frac{1}{3} & x^3 + 4x^2 + 3x + 5 \\ & x^3 \qquad\quad - \frac{1}{3}x \\ \hline & 4x^2 + \frac{10}{3}x + 5 \\ & 4x^2 \qquad\quad - \frac{4}{3} \\ \hline & \frac{10}{3}x + \frac{19}{3} = R(x) \end{array} \qquad = Q(x)$$

$$P(x) = \underbrace{(x+4)}_{\deg < 2} P_2(x) + \underbrace{\frac{10}{3}x + \frac{19}{3}}_{\deg < 2}$$

Error bound for quadrature rules

Recall Theorem 4.6 p215:

If $f \in C^{m+1}[a,b]$ then $R(x) = f(x) - p(x) = \dfrac{f^{(m+1)}(\xi)}{(m+1)!} \prod_{j=0}^{m} (x - x_j)$.

It follows that

$$|E(f)| = |Q(f) - \int f| \le \frac{\|f^{(m+1)}\|_\infty}{(m+1)!} \int_a^b \prod_j (x - x_j)\, dx$$

$$\le H^{m+2} \beta_{m+1} \|f^{(m+1)}\|_\infty \qquad \text{where } H = b - a$$

$$\beta_{m+1} = \int_0^1 (z - z_j)\, dz$$

Thus $E(f) \to 0$ as $H \to 0$.

and we can reduce "H" by dividing our domain [a,b] into small subintervals or "panels", applying the rule to each panel and adding the results.

In practice, though, ... we estimate our error numerically, as follows ...

Suppose we have two rules, such as G7 and "K15",

where deg(K15) >> deg(G7).

Then $G_7(f) - K_{15}(f)$ is a reasonable estimate of the error in G7(f).

K in K15 above stands for Kronrod.

The (2m+1)-point Kronrod rule is the (2m+1)-point rule of maximal polynomial degree that re-uses the sample points of the m-point Gaussian rule.

Let's take a look at G7-K15 ...

Let's test the 7,15-point Gauss-Kronrod pair of rules

```
from numpy import *

# nodes and weights for the interval [-1,1] copied-and-pasted from Wikipedia

knodes = array([
-0.991455371120812639 2,
-0.949107912342758524 5,
-0.864864423359769072 8,
-0.741531185599394439 9,
-0.586087235467691130 3,
-0.405845151377397166 9,
-0.207784955007898467 6,
 0.,
 0.207784955007898467 6,
 0.405845151377397166 9,
 0.586087235467691130 3,
 0.741531185599394439 9,
 0.864864423359769072 8,
 0.949107912342758524 5,
 0.991455371120812639 2])

gnodes = knodes[1::2]

kweights = array([
0.022935322010529224964,
0.063092092629978553 29,
0.104790010322250183 84,
0.140653259715525918 75,
0.169047266392679028 3,
0.190350578064785409 9,
0.204432940075298892 41,
0.209482141084727828 0,
0.204432940075298892 41,
0.190350578064785409 9,
0.169047266392679028 3,
0.140653259715525918 75,
0.104790010322250183 84,
0.063092092629978553 29,
0.022935322010529224964])

gweights = array([
0.129484966168869693270611432679082018328587402259 95, \
0.279705391489276667901467771423779582486925065226 6, \
0.381830050505118944950369775488975133878365083533 9, \
0.417959183673469387755102040816326530612244897959 2, \
0.381830050505118944950369775488975133878365083533 9, \
0.279705391489276667901467771423779582486925065226 6, \
0.129484966168869693270611432679082018328587402259 9   ])
```

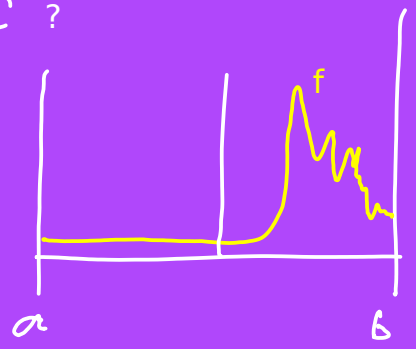
```
def g7(f,a,b):
    global gnodes,gweights
    center = (a+b)/2.
    halfwidth = (b-a)/2.
    nodes = center + halfwidth*gnodes
    weights = halfwidth*gweights
    return dot( weights, f(nodes) )

def k15(f,a,b):
    global knodes,kweights
    center = (a+b)/2.
    halfwidth = (b-a)/2.
    nodes = center + halfwidth*knodes
    weights = halfwidth*kweights
    return dot( weights, f(nodes) )

# test them
def myf(x): return  sin(x) #x**10
def myF(x): return -cos(x) # for this example we can provide an antiderivative to obtain the integral exactly

a,b = 2,5  # randomly chosen interval

exact = myF(b) - myF(a)
print( 'Exact     :',exact )

Gest  = g7(myf,a,b)
print( "Gauss 7   :", Gest, ', error:', abs(Gest-exact)/exact )

Kest  = k15(myf,a,b)
print( "Kronrod 15:", Kest, ', error:', abs(Kest-exact)/exact )
```

```
Exact     : -0.6998090220103687
Gauss 7   : -0.6998090220106843 , error: -4.510322044637135e-13
Kronrod 15: -0.6998090220103687 , error: -0.0
```

Wow!!

Adaptive quadrature

How to attain an integral approximation with error tolerance ε ?



Apply first directly to [a,b] and obtain error estimate.

If error estimate $> \varepsilon$,

then apply rule separately

on $\left[a, \frac{a+b}{2}\right]$ with tolerance $\frac{\varepsilon}{2}$,

and on $\left[\frac{a+b}{2}, b\right]$ with tolerance $\frac{\varepsilon}{2}$,

and add the results.

Do this recursively until total error estimate $< \varepsilon$.

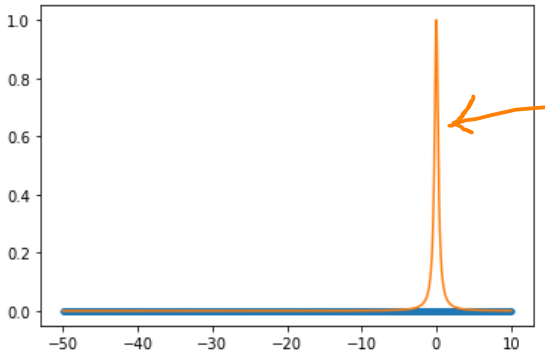
Let's try it out ...

Adaptive quadrature with G7K15

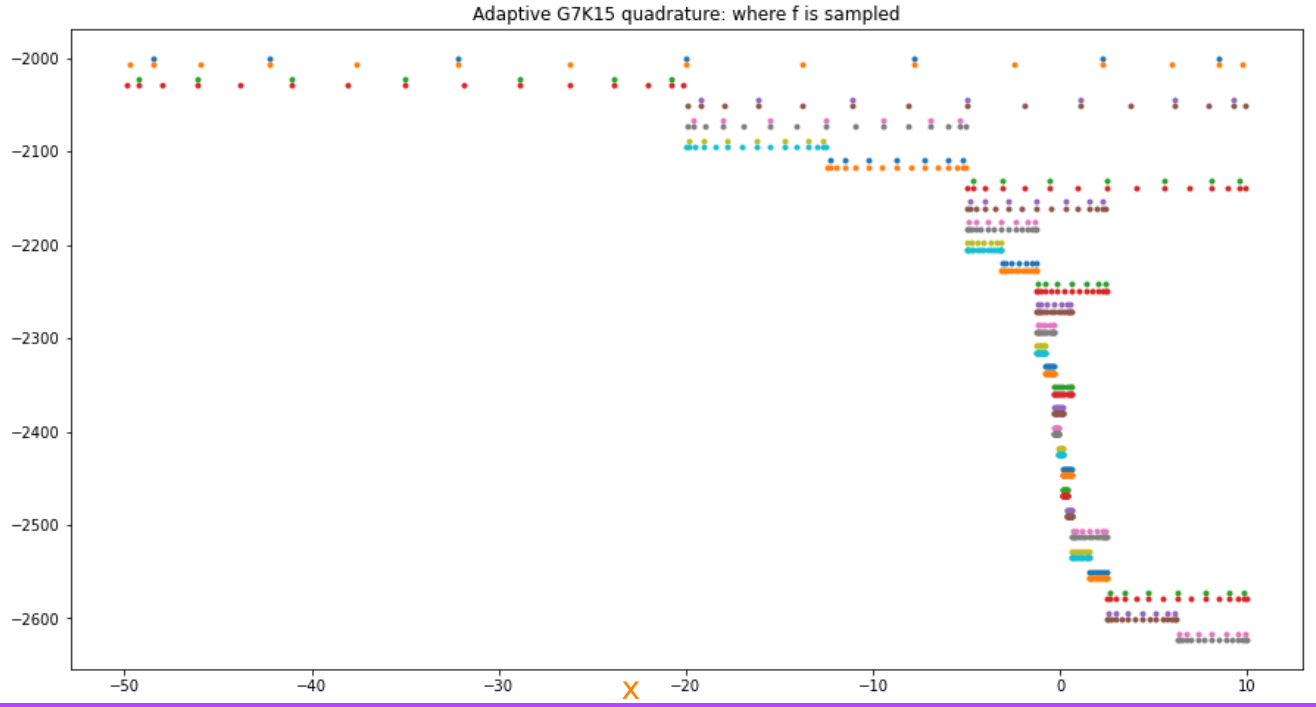
```
import numpy as np
from gk import g7,k15
import matplotlib.pyplot as plt

count = 0

def runge(x):
    global count
    plt.plot(x,-count*np.ones_like(x),'.')   # plot the points where we've been asked to evaluate the function
    count += len(x)
    return 1/(1+12*x**2)

f = runge

def F(x):  # antiderivative of Runge
    s = np.sqrt(12)
    return np.arctan(s*x)/s

def quad( f, a, b ):
    g = g7 ( f, a, b )
    k = k15( f, a, b )
    error_estimate = 200*abs(g-k)**1.5
    return k, error_estimate

def adaptive( f, a, b, tol ):
    i,e = quad( f, a, b )
    if e <= tol:
        return i, e
    else:
        midpt = (a+b)/2
        i1, e1 = adaptive( f, a    , midpt, tol/2 )
        i2, e2 = adaptive( f, midpt, b    , tol/2 )
        return i1+i2, e1+e2

a = -50
b =  10
exact = F(b) - F(a)
xx = linspace(a,b,2000); plt.figure(); plt.plot(xx,f(xx))
print( "exact ", exact )
plt.figure(figsize=(15,8))
i,e = adaptive( f, a, b, 1e-10 )
print( "approx", i, ", error estimate", e )
print( "actual error", i-exact )

plt.title('Adaptive G7K15 quadrature: where f is sampled')
plt.savefig('temp.pdf')
```

```
exact  0.896902014293353
approx 0.8969020142933529 , error estimate 2.647744277426301e-12
actual error -1.1102230246251565e-16
```

accurate to machine precision!



Integrand (Runge function) has a big spike in a small region of our interval [-50,10].



Closing question:

"How is it that the Gaussian rules are so remarkably accurate for non-polynomial functions?"

Consider the 2-point (m=1) Gaussian rule. There is a unique polynomial of degree 1 or less that interpolates the sampled values of the function f for which we want to evaluate $\int_{-1}^{1}$.

But there is an entire 2D family of functions within P_3 that interpolate the data, and if we choose the Gaussian sample points they all have the same integral (because we know the 2-point Gauss rule gets the integral exactly for all $p \in P_3$).

Thus there is an entire 2D family of interpolating cubics with the same integral, and it is plausible that some of them approximate f quite well, yielding an accurate approximation of its integral. Illustration below.

Let's take a look at an example ...