Math 541 - Numerical Analysis

Lecture Notes – Quadrature – Part B

Joseph M. Mahaffy,
⟨jmahaffy@mail.sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://jmahaffy.sdsu.edu

Spring 2018

Outline

SDSU

Divide and Conquer with Simpson's Rule

The exact solution:

$$\int_0^4 e^x dx = e^4 - e^0 = 53.59815$$

Simpson's Rule with $h = 2$

$$\int_0^4 e^x dx \approx \frac{2}{3}(e^0 + 4e^2 + e^4) = 56.76958.$$

The error is **3.17143** (5.92%).

Divide-and-Conquer: Simpson's Rule with $h = 1$

$$\int_0^2 e^x dx + \int_2^4 e^x dx \approx \frac{1}{3}(e^0 + 4e^1 + e^2) + \frac{1}{3}(e^2 + 4e^3 + e^4) = 53.86385$$

The error is **0.26570**. (0.50%) Improvement by a factor of 10!

Divide and Conquer with Simpson's Rule

The exact solution:

$$\int_0^4 e^x dx = e^4 - e^0 = 53.59815$$

Divide-and-Conquer: Simpson's Rule with $h = 1/2$

$$\int_0^1 + \int_1^2 + \int_2^3 + \int_3^4 e^x dx \approx \frac{1}{6}(e^0 + 4e^{1/2} + e^1) + \frac{1}{6}(e^1 + 4e^{3/2} + e^2)$$
$$+\frac{1}{6}(e^2 + 4e^{5/2} + e^3) + \frac{1}{6}(e^3 + 4e^{7/2} + e^4) = 53.61622$$

The error has been reduced to **0.01807** (0.034%).

h	abs-error	err/h	err/h^2	err/h^3	err/h^4
2	3.17143	1.585715	0.792857	0.396429	0.198214
1	0.26570	0.265700	0.265700	0.265700	0.265700
1/2	0.01807	0.036140	0.072280	0.144560	0.289120

Divide and Conquer with Simpson's Rule

Extending the table...

h	abs-error	err/h	err/h^2	err/h^3	**err/h^4**	err/h^5
2	3.171433	1.585716	0.792858	0.396429	**0.198215**	0.099107
1	0.265696	0.265696	0.265696	0.265696	**0.265696**	0.265696
1/2	0.018071	0.036142	0.072283	0.144566	**0.289132**	0.578264
1/4	0.001155	0.004618	0.018473	0.073892	**0.295566**	1.182266
1/8	0.000073	0.000580	0.004644	0.037152	**0.297215**	2.377716
1/16	0.000004	0.000072	0.001162	0.018601	**0.297629**	4.762065

Clearly, the **err/h^4** column seems to converge (to a non-zero constant) as $h \to 0$. The columns to the left seem to converge to zero, and the err/h^5 column seems to grow.

This is ***numerical evidence*** that the composite Simpson's rule has a convergence rate of $\mathcal{O}\left(h^4\right)$. But, isn't Simpson's rule 5th order???

Generalized Composite Simpson's Rule

For an even integer n: Subdivide the interval $[a, b]$ into n subintervals, and apply Simpson's rule on each consecutive pair of sub-intervals. With $h = (b-a)/n$ and $x_j = a + jh$, $j = 0, 1, \ldots, n$, we have

$$\int_a^b f(x)dx = \sum_{j=1}^{n/2} \int_{x_{2j-2}}^{x_{2j}} f(x)dx$$

$$= \sum_{j=1}^{n/2} \left\{ \frac{h}{3} \Big[f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j}) \Big] - \frac{h^5}{90} f^{(4)}(\xi_j) \right\},$$

for some $\xi_j \in [x_{2j-2}, x_{2j}]$, if $f \in C^4[a, b]$.

Since all the interior "even" x_{2j} points appear twice in the sum, we can simplify the expression a bit...

Generalized Composite Simpson's Rule

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(x_0) - f(x_n) + \sum_{j=1}^{n/2}\left[4f(x_{2j-1}) + 2f(x_{2j})\right]\right]$$
$$-\frac{h^5}{90}\sum_{j=1}^{n/2} f^{(4)}(\xi_j).$$

The error term is:

$$E(f) = -\frac{h^5}{90}\sum_{j=1}^{n/2} f^{(4)}(\xi_j), \quad \xi_j \in [x_{2j-2}, x_{2j}]$$

The Error for Composite Simpson's Rule

If $f \in C^4[a,b]$, the **Extreme Value Theorem** implies that $f^{(4)}$ assumes its max and min in $[a,b]$. Now, since

$$\min_{x\in[a,b]} f^{(4)}(x) \le f^{(4)}(\xi_j) \le \max_{x\in[a,b]} f^{(4)}(x),$$

$$\left[\frac{n}{2}\right] \min_{x\in[a,b]} f^{(4)}(x) \le \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \le \left[\frac{n}{2}\right] \max_{x\in[a,b]} f^{(4)}(x),$$

$$\min_{x\in[a,b]} f^{(4)}(x) \le \left[\frac{2}{n}\right] \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \le \max_{x\in[a,b]} f^{(4)}(x),$$

By the **Intermediate Value Theorem** there exists $\mu \in (a,b)$ so that

$$f^{(4)}(\mu) = \frac{2}{n}\sum_{j=1}^{n/2} f^{(4)}(\xi_j) \quad \Leftrightarrow \quad \frac{n}{2} f^{(4)}(\mu) = \sum_{j=1}^{n/2} f^{(4)}(\xi_j)$$

The Error for Composite Simpson's Rule

We can now rewrite the error term:

$$E(f) = -\frac{h^5}{90}\sum_{j=1}^{n/2} f^{(4)}(\xi_j) = -\frac{h^5}{180} n f^{(4)}(\mu),$$

or, since $h = (b-a)/n \Leftrightarrow n = (b-a)/h$, we can write

$$E(f) = -\frac{(b-a)}{180} h^4 f^{(4)}(\mu).$$

Hence ***Composite Simpson's Rule*** has **degree of accuracy 3** (since it is exact for polynomials up to order 3), and the error is proportional to h^4 — ***Convergence Rate*** $\mathcal{O}\left(h^4\right)$.

Composite Simpson's Rule — Summary

Theorem (Composite Simpson's Rule)

*Let $f \in C^4[a,b]$, n be even, $h = (b-a)/n$, and $x_j = a + jh$, $j = 0, 1, \dots, n$. There exists $\mu \in (a,b)$ for which the **Composite Simpson's Rule** for n subintervals can be written with its error term as*

$$\int_a^b f(x)\,dx = \frac{h}{3}\left[f(a) - f(b) + \sum_{j=1}^{n/2}\left[4f(x_{2j-1}) + 2f(x_{2j})\right]\right] - \frac{(b-a)}{180}h^4 f^{(4)}(\mu).$$

Note: $x_0 = a$, and $x_n = b$.

Composite Simpson's Rule — MatLab

The MatLab code **Composite Simpson's Rule** allows varying the interval $[a, b]$ and the number of subdivisions $h = \frac{b-a}{2N}$.
The function $f(x)$ is inputted on `Line 4`.

```
1  function S = compsimp(a,b,N)
2  % Composite Simpson's Rule for function f(x)
3  % on [a,b] using 2N steps
4  f = @(x) exp(x);
5  h = (b-a)/(2*N);
6  i = 0:N-1;
7  xi = a+2*i*h;
8  xi1 = a+2*(i+0.5)*h;
9  xi2 = a+2*(i+1)*h;
10 S = (h/3)*sum(f(xi)+4*f(xi1)+f(xi2));
11 end
```

Composite Midpoint Rule — MatLab

The MatLab code for the **Composite Midpoint Rule** allows varying the interval $[a, b]$ and number of subdivisions $h = \frac{b-a}{N}$. The function $f(x)$ is inputted on `Line 4`.

```
1 function M = compmidpt(a,b,N)
2 % Composite Midpoint Rule for function f(x)
3 % on [a,b] using N steps
4 f = @(x) exp(x);
5 h = (b-a)/N;
6 i = 1:N;
7 ci = a+0.5*(2*i-1)*h;
8 M = h*sum(f(ci));
9 end
```

Composite Midpoint Rule — Convergence

The **Composite Midpoint Rule** is applied to

$$\int_0^4 e^x dx = 53.59815003$$

with various stepsizes to determine the order of convergence.

Recall that ***local convergence*** of the **Midpoint Rule** is $\mathcal{O}\left(h^3\right)$.

h	Approx	abs-error	err/h	**err/h^2**	err/h^3
2	45.607638	7.990513	3.995256	**1.997628**	0.998814
1	51.428356	2.169794	2.169794	**2.169794**	2.169794
1/2	53.043880	0.554270	1.108539	**2.217079**	4.434157
1/4	53.458826	0.139325	0.557299	**2.229193**	8.916770
1/8	53.563271	0.034879	0.279030	**2.232239**	17.857911
1/16	53.589427	0.008723	0.139562	**2.232994**	35.727905

Clearly, the **err/h^2** column seems to converge (to a non-zero constant) as $h \to 0$.
This is ***numerical evidence*** that the **Composite Midpoint Rule** has a convergence rate of $\mathcal{O}\left(h^2\right)$.

Composite Trapezoid Rule — MatLab

The MatLab code for the **Composite Trapezoid Rule** allows varying the interval $[a, b]$ and number of subdivisions $h = \frac{b-a}{N}$. The function $f(x)$ is inputted on `Line 4`.

```
1  function T = comptrap(a,b,N)
2  % Composite Trapezoid Rule for function f(x)
3  % on [a,b] using N steps
4  f = @(x) exp(x);
5  h = (b-a)/N;
6  i = 0:N-1;
7  xi = a+i*h;
8  xi1 = a+(i+1)*h;
9  T = (h/2)*sum(f(xi)+f(xi1));
10 end
```

Composite Trapezoid Rule — Convergence

The **Composite Trapezoid Rule** is applied to

$$\int_0^4 e^x dx = 53.59815003$$

with various stepsizes to determine the order of convergence.

Recall that ***local convergence*** of the **Trapezoid Rule** is $\mathcal{O}\left(h^3\right)$.

h	Approx	abs-error	err/h	**err/h^2**	err/h^3
2	70.376262	16.778112	8.389056	**4.194528**	2.097264
1	57.991950	4.393800	4.393800	**4.393800**	4.3938004
1/2	54.710153	1.112003	2.224006	**4.448012**	8.8960237
1/4	53.877017	0.278867	1.115467	**4.461867**	17.847467
1/8	53.667921	0.069771	0.558168	**4.465342**	35.722735
1/16	53.61560	0.017446	0.279135	**4.466168**	71.458680

Clearly, the **err/h^2** column seems to converge (to a non-zero constant) as $h \to 0$.
This is ***numerical evidence*** that the **Composite Trapezoid Rule** has a convergence rate of $\mathcal{O}\left(h^2\right)$.

Composite Boole's Rule — MatLab

The MatLab code for the **Composite Boole's Rule** allows varying the interval $[a, b]$ and number of subdivisions $h = \frac{b-a}{4N}$.
The function $f(x)$ is inputted on `Line 4`.

```
1  function B = compboole(a,b,N)
2  % Composite Boole's Rule for function f(x)
3  % on [a,b] using 4N steps
4  f = @(x) exp(x);
5  h = (b-a)/(4*N);
6  i = 0:N-1;
7  xi = a+4*i*h;
8  xi1 = a+4*(i+0.25)*h;
9  xi2 = a+4*(i+0.5)*h;
10 xi3 = a+4*(i+0.75)*h;
11 xi4 = a+4*(i+1)*h;
12 B = (2*h/45)*sum(7*f(xi)+32*f(xi1)+12*f(xi2)...
13     +32*f(xi3)+7*f(xi4));
14 end
```

Composite Boole's Rule — Convergence

The **Composite Boole's Rule** is applied to

$$\int_0^4 e^x dx = 53.59815003$$

with various stepsizes to determine the order of convergence.

Recall that ***local convergence*** of the **Boole's Rule** is $\mathcal{O}\left(h^7\right)$.

h	Approx	abs-error	err/h^5	**err/h^6**	err/h^7
1	53.670130	0.071980	0.071980	**0.071980**	0.071980
1/2	53.599712	0.001562	0.049998	**0.099996**	0.199991
1/4	53.598177	2.6809E-05	0.027453	**0.109811**	0.439242
1/8	53.598150	4.2920E-07	0.014064	**0.112511**	0.900087
1/16	53.598150	6.7474E-09	0.007075	**0.113202**	1.811232

Clearly, the **err/h^6** column seems to converge (to a non-zero constant) as $h \to 0$.
This is ***numerical evidence*** that the **Composite Boole's Rule** has a convergence rate of $\mathcal{O}\left(h^6\right)$.

Composite Simpson's Rule — Refined

The MatLab code for the **Composite Simpson's Rule** allows varying the interval $[a, b]$ and number of subdivisions $h = \frac{b-a}{2N}$. The function $f(x)$ is inputted on `Line 4`.

The stepsize h is subdivided in half until successive approximations for the integral are within a ***specified tolerance***.

```
1 function [h,S] = compsimptol(a,b,tol)
2 % Composite Simpson's Rule for function f(x)
3 % on [a,b] doubling steps til within tolerance
4 f = @(x) exp(x);
5 S0 = 0;
6 N = 1;
7 j = 0;
8 h = (b-a)/2;
```

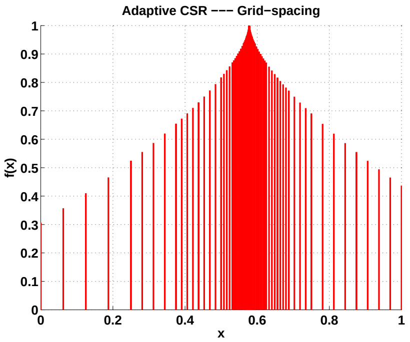
Composite Simpson's Rule — Refined

The code below completes the code from the previous slide

```
S = f(h)*(b-a);
while (abs(S-S0)>tol)
    S0 = S;
    h = (b-a)/(2*N);
    i = 0:N-1;
    xi = a+2*i*h;
    xi1 = a+2*(i+0.5)*h;
    xi2 = a+2*(i+1)*h;
    S = (h/3)*sum(f(xi)+4*f(xi1)+f(xi2));
    j = j + 1;
    N = 2^j;
end
```

More Advanced Numerical Integration Ideas

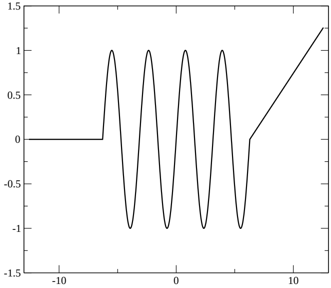
Adaptive



Introduction

The ***composite formulas*** require ***equally spaced nodes***.

This is not good if the function we are trying to integrate has both regions with large fluctuations, and regions with small variations.



We need many points where the function fluctuates, but few points where it is close to constant or linear.

Introduction — Adaptive Quadrature Methods

Idea Cleverly predict (or measure) the amount of variation and automatically add more points where needed.

We are going to discuss this in the context of Composite Simpson's rule, but the approach can be adopted for other integration schemes.

First we are going to develop a way to ***measure the error*** — a numerical estimate of the actual error in the numerical integration. **Note**: just knowing the structure of the error term is not enough! (We will however use the structure of the error term in our derivation of the numerical error estimate.)

Then we will use the error estimate to decide whether to accept the value from CSR, or if we need to refine further (recompute with smaller h).

Some Notation — One-step Simpson's Rule $S(f;a,b)$

Notation — "One-step" Simpson's Rule:

$$\int_a^b f(x)\,dx = S(f;\,a,b) - \underbrace{\frac{h_1^5}{90} f^{(4)}(\mu_1)}_{E(f;h_1,\mu_1)}, \quad \mu_1 \in (a,b),$$

where

$$S(f;\,a,b) = \frac{(b-a)}{6}\left[f(a) + 4f\left(\tfrac{a+b}{2}\right) + f(b)\right], \quad h_1 = \frac{(b-a)}{2}.$$

Composite Simpson's Rule (CSR)

With this notation, we can write CSR with $n = 4$, and $h_2 = (b-a)/4 = h_1/2$:

$$\int_a^b f(x)\,dx = S(f;\, a, \tfrac{a+b}{2}) + S(f\,;\, \tfrac{a+b}{2}, b) - E(f; h_2, \mu_2).$$

We can squeeze out an estimate for the error by noticing that

$$E(f; h_2, \mu_2) = \frac{1}{16}\left(\frac{h_1^5}{90} f^{(4)}(\mu_2)\right) = \frac{1}{16} E(f; h_1, \mu_2).$$

Now, **assuming** $f^{(4)}(\mu_1) \approx f^{(4)}(\mu_2)$, we do a little bit of algebra magic with our two approximations to the integral...

Wait! Wait! Wait! — I pulled a fast one!

$$E(f; h_2, \mu_2) = \frac{1}{32}\left(\frac{h_1^5}{90} f^{(4)}(\mu_2^1)\right) + \frac{1}{32}\left(\frac{h_1^5}{90} f^{(4)}(\mu_2^2)\right)$$

where $\mu_2^1 \in [a, \frac{a+b}{2}]$, $\mu_2^2 \in [\frac{a+b}{2}, b]$.

If $f \in C^4[a,b]$, then we can use our old friend, the **intermediate value theorem**:

$$\text{There exists} \quad \mu_2 \in [\mu_2^1, \mu_2^2] \subset [a,b] : \; f^{(4)}(\mu_2) = \frac{f^{(4)}(\mu_2^1) + f^{(4)}(\mu_2^2)}{2}.$$

So it follows that

$$E(f; h_2, \mu_2) = \frac{1}{16}\left(\frac{h_1^5}{90} f^{(4)}(\mu_2)\right).$$

Back to the Error Estimate...

Now we have

$$S(f;\, a, \tfrac{a+b}{2}) + S(f;\, \tfrac{a+b}{2}, b) - \frac{1}{16}\left(\frac{h_1^5}{90} f^{(4)}(\mu_2)\right)$$
$$= S(f;\, a, b) - \frac{h_1^5}{90} f^{(4)}(\mu_1).$$

Now use the assumption $f^{(4)}(\mu_1) \approx f^{(4)}(\mu_2)$ (and replace μ_1 and μ_2 by μ):

$$\frac{h_1^5}{90} f^{(4)}(\mu) \approx \frac{16}{15}\Big[S(f;\, a, b) - S(f;\, a, (a+b)/2) - S(f;\, (a+b)/2, b)\Big],$$

notice that $\frac{h_1^5}{90} f^{(4)}(\mu) = E(f;\, h_1, \mu) = 16E(f;\, h_2, \mu)$. Hence

$$E(f;\, h_2, \mu) \approx \frac{1}{15}\Big[S(f;\, a, b) - S(f;\, a, (a+b)/2) - S(f;\, (a+b)/2, b)\Big],$$

Finally, we have the error estimate in hand...

Using the estimate of $\frac{h_1^5}{90} f^{(4)}(\mu)$, we have

Error Estimate for CSR

$$\left| \int_a^b f(x)dx - S(f;\, a, (a+b)/2) - S(f;\, (a+b)/2, b) \right|$$
$$\approx \frac{1}{15} \left| S(f;\, a, b) - S(f;\, a, (a+b)/2) - S(f;\, (a+b)/2, b) \right|$$

Notice!!! $S(f;\, a, (a+b)/2) + S(f;\, (a+b)/2, b)$ approximates $\int_a^b f(x)dx$ **15 times better** than it agrees with the known quantity $S(f;\, a, b)$!!!

Example — Error Estimates

We will apply Simpson's rule to

$$\int_0^{\pi/2} \sin(x)\,dx = 1.$$

Here,

$$\mathbb{S}_1(\sin(x);\ 0,\pi/2) = S(\sin(x);\ 0,\pi/2)$$

$$= \frac{\pi}{12}\Big[\sin(0) + 4\sin(\pi/4) + \sin(\pi/2)\Big] = \frac{\pi}{12}\Big[2\sqrt{2}+1\Big]$$
$$= 1.00227987749221.$$

$$\mathbb{S}_2(\sin(x);\ 0,\pi/2) = S(\sin(x);\ 0,\pi/4) + S(\sin(x);\ \pi/4,\pi/2)$$

$$= \frac{\pi}{24}\Big[\sin(0) + 4\sin(\pi/8) + 2\sin(\pi/4) + 4\sin(3\pi/8) + \sin(\pi/2)\Big]$$
$$= 1.00013458497419.$$

Example — Error Estimates

The error estimate is given by

$$\begin{aligned}
&\frac{1}{15}\left[\mathbb{S}_1(\sin(x);\ 0,\pi/2) - \mathbb{S}_2(\sin(x);\ 0,\pi/2)\right] \\
&\quad = \frac{1}{15}\left[1.00227987749221 - 1.00013458497419\right] \\
&\quad = 0.00014301950120.
\end{aligned}$$

This is a very good approximation of the actual error, which is 0.00013458497419.

OK, we know how to get an error estimate. How do we use this to create an adaptive integration scheme???

Adaptive Quadrature

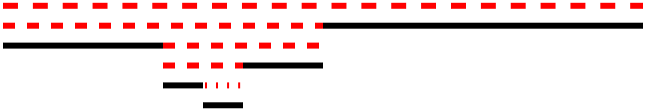
We want to approximate $\mathcal{I} = \int_a^b f(x)\,dx$ with an error less than ϵ (a specified tolerance).

`[1]` Compute the two approximations
$\mathbb{S}_1(f(x);\, a, b) = S(f(x);\, a, b)$, and
$\mathbb{S}_2(f(x);\, a, b) = S(f(x);\, a, \frac{a+b}{2}) + S(f(x);\, \frac{a+b}{2}, b)$.

`[2]` Estimate the error, if the estimate is less than ϵ, we are done. Otherwise...

`[3]` Apply steps `[1]` and `[2]` recursively to the intervals $[a, \frac{a+b}{2}]$ and $[\frac{a+b}{2}, b]$ with tolerance $\epsilon/2$.

Adaptive Quadrature, Interval Refinement Example 1



The funny figure above is supposed to illustrate a possible sub-interval refinement hierarchy. **Red** dashed lines illustrate failure to satisfy the tolerance, and **black** lines illustrate satisfied tolerance.

level	tol	interval
1	ϵ	$[\mathbf{a},\mathbf{b}]$
2	$\epsilon/2$	$[\mathbf{a},\mathbf{a}+\frac{\mathbf{b-a}}{\mathbf{2}}]$ $[\mathbf{a}+(\mathbf{b}-\mathbf{a})/\mathbf{2},\mathbf{b}]$
3	$\epsilon/4$	$[\mathbf{a},\mathbf{a}+\frac{\mathbf{b-a}}{\mathbf{4}}]$ $[\mathbf{a}+\frac{\mathbf{b-a}}{\mathbf{4}},\mathbf{a}+\frac{\mathbf{b-a}}{\mathbf{2}}]$
$\vdots$		

Adaptive Quadrature – MatLab

Below are the **MatLab** programs for **Adaptive Quadrature**

```
function [val,err] = ACSR(f,a,b,tol,level,h)

h1 = (b-a)/2;  x1 = (a:h1:b);  w1 = [1 4 1];
h2 = h1/2;     x2 = (a:h2:b);  w2 = [1 4 2 4 1];

S1 = h1/3*sum(f(x1).*w1);
S2 = h2/3*sum(f(x2).*w2);

err = abs(S1-S2)/15;

if( err < tol )
  fprintf('CSR succeeded at level %d on interval ...
      [%f,%f]\n',...
      level,a,b);
  val = S2;
```

Adaptive Quadrature – MatLab

```
  figure(2); hold on
  plot([a b],[level level],'o-','linewidth',3)
  hold off
  figure(3); hold on
  for k = 1:length(x2)
    plot([x2(k) x2(k)],[0,f(x2(k))],'r-')
  end
  hold off
else
  [lt, err_lt] = ACSR(f,a,(a+b)/2,tol/2,level+1,h);
  [rt, err_rt] = ACSR(f,(a+b)/2,b,tol/2,level+1,h);
  val = lt+rt;
  err = err_lt+err_rt;
end
```

Adaptive Quadrature – MatLab

Below is the graphic display program for **Adaptive Quadrature**

```
tol = 10^(-6);
a   = 0;
b   = 1;

f = @(x) 1-((x-pi/2/exp(1)).^2).^(1/3);

figure(1); xv = 0:0.001:1;
plot(xv,f(xv),'-','linewidth',3)
title('Adaptive CSR --- The ...
     Function','fontweight','bold','fontsize',14)
axis([0 1 0 1])
xlabel('x','fontweight','bold','fontsize',14)
ylabel('f(x)','fontweight','bold','fontsize',14)
grid on
```

Adaptive Quadrature – MatLab

```
h = figure(2); clf;
title('Adaptive CSR --- Refinement ... Levels','fontweight','bold','fontsize',14)
xlabel('x','fontweight','bold','fontsize',14)
ylabel('Refinement ... Level','fontweight','bold','fontsize',14)
grid on

[val,err] = ACSR(f,a,b,tol,1,h);

figure(h)
ax = axis;
axis([ax(1:2) 0 ax(4)])
hold off
fprintf('\n\nThe integral: %10.8f,  the error: %e ... \n\n',val,err);
```

Adaptive Quadrature, Interval Refinement Example 2

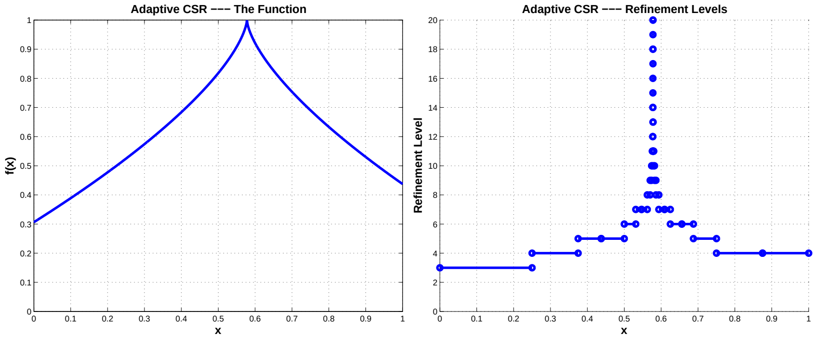


Figure: Application of adaptive CSR to the function $f(x) = 1 - \sqrt[3]{(x - \frac{\pi}{2e})^2}$. Here, we have required that the estimated error be less than 10^{-6}. The left panel shows the function, and the right panel shows the number of refinement levels needed to reach the desired accuracy. At completion we have the value of the integral being 0.61692712, with an estimated error of $3.93 \cdot 10^{-7}$.

Gaussian Quadrature

Idea: Evaluate the function at a set of **optimally chosen** points in the interval.

We will choose $\{x_0, x_1, \ldots, x_n\} \in [a, b]$ and coefficients c_i, so that the approximation

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} c_i f(x_i)$$

is exact for the largest class of polynomials possible.

We have already seen that the open Newton-Cotes formulas sometimes give us better "bang-for-buck" than the closed formulas (*e.g.* the mid-point formula uses only 1 point and is as accurate as the two-point trapezoidal rule). — Gaussian quadrature takes this one step further.

Quadrature Types — A Comparison

	Newton-Cotes		Gaussian
	Open	Closed	
Quadrature Points	Degree of Accuracy	Degree of Accuracy	Degree of Accuracy
1	**1** *	—	1
2	1	**1** †	3
3	3	**3** #	5
4	3	3	7
5	5	5	9

* — The mid-point rule.
† — Trapezoidal rule.
# — Simpson's rule.

The mid-point rule is the only optimal scheme we have see so far.

Gaussian Quadrature — Example

2-Point Formula

Consider a 3^{rd} **order polynomial**:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

This ***polynomial*** is constructed from ***4 linearly independent functions***, 1, x, x^2, and x^3, on the interval $x \in [-1, 1]$. (There are 4 arbitrary constants, a_i.)

We should be able to find an optimal two-point formula:

$$\int_{-1}^{1} f(x)\, dx = c_1 f(x_1) + c_2 f(x_2),$$

to solve this integral problem, since we have 4 parameters, c_1, c_2, x_1, and x_2

Thus, we seek optimal points x_i and optimal weights c_i, which give the value of the ***integral exactly up to polynomials of degree 3***

Gaussian Quadrature — Example

2-Point Formula

Suppose we want to find this optimal two-point formula:

$$\int_{-1}^{1} f(x)\,dx = c_1 f(x_1) + c_2 f(x_2).$$

Since we have 4 parameters to play with, we can generate a formula that is ***exact up to polynomials of degree 3***. We get the following 4 equations:

$$\begin{aligned} \int_{-1}^{1} 1\,dx &= 2 &= c_1 + c_2 \\ \int_{-1}^{1} x\,dx &= 0 &= c_1x_1 + c_2x_2 \\ \int_{-1}^{1} x^2\,dx &= \tfrac{2}{3} &= c_1x_1^2 + c_2x_2^2 \\ \int_{-1}^{1} x^3\,dx &= 0 &= c_1x_1^3 + c_2x_2^3 \end{aligned}$$

$$\begin{aligned} c_1 &= 1 \\ c_2 &= 1 \\ x_1 &= -\frac{\sqrt{3}}{3} \\ x_2 &= \frac{\sqrt{3}}{3} \end{aligned}$$

Higher Order Gaussian Quadrature Formulas

We could obtain higher order formulas by adding more points, computing the integrals, and solving the resulting non-linear system of equations... but it gets very painful, very fast.

The **Legendre Polynomials** come to our rescue!

The Legendre polynomials $P_n(x)$ are ***orthogonal*** on $[-1, 1]$ with respect to the weight function $w(x) = 1$, *i.e.*,

$$\int_{-1}^{1} P_n(x) P_m(x)\, dx = \alpha_n \delta_{n,m} = \begin{cases} 0 & m \neq n \\ \alpha_n & m = n. \end{cases}$$

If $P(x)$ is a polynomial of degree less than n, then

$$\int_{-1}^{1} P_n(x) P(x)\, dx = 0.$$

A Quick Note on Legendre Polynomials

We will see Legendre polynomials in **more detail later**. For now, all we need to know is that they satisfy the property

$$\int_{-1}^{1} P_n(x)P_m(x)\,dx = \alpha_n \delta_{n,m}.$$

and the first few Legendre polynomials are

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_2(x) &= x^2 - 1/3 \\ P_3(x) &= x^3 - 3x/5 \\ P_4(x) &= x^4 - 6x^2/7 + 3/35 \\ P_5(x) &= x^5 - 10x^3/9 + 5x/21. \end{aligned}$$

It turns out that the **roots** of the Legendre polynomials are the nodes in Gaussian quadrature.

Higher Order Gaussian Quadrature Formulas

Theorem

Suppose that $\{x_1, x_2, \ldots, x_n\}$ *are the roots of the* n^{th} *Legendre polynomial* $P_n(x)$ *and that for each* $i = 1, 2, \ldots, n$*, the coefficients* c_i *are defined by*

$$c_i = \int_{-1}^{1} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}\, dx.$$

If $P(x)$ *is any polynomial of degree less than* $2n$*, then*

$$\int_{-1}^{1} P(x)\, dx = \sum_{i=1}^{n} c_i P(x_i).$$

Proof of the Theorem

Let us first consider a polynomial, $P(x)$ with degree less than n. $P(x)$ can be rewritten as an $(n-1)^{st}$ Lagrange polynomial with nodes at the roots of the n^{th} Legendre polynomial $P_n(x)$. This representation is exact, since the error term involves the n^{th} derivative of $P(x)$, which is zero. Hence,

$$\int_{-1}^{1} P(x)\,dx = \int_{-1}^{1} \left[\sum_{i=1}^{n} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} P(x_i) \right] dx$$

$$= \sum_{i=1}^{n} \left[\int_{-1}^{1} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}\,dx \right] P(x_i) = \sum_{i=1}^{n} c_i P(x_i),$$

which verifies the result for polynomials of degree less than n.

Proof of the Theorem

If the polynomial $P(x)$ of degree $[n, 2n)$ is divided by the n^{th} Legendre polynomial $P_n(x)$, we get:

$$P(x) = Q(x)P_n(x) + R(x)$$

where both $Q(x)$ and $R(x)$ are of degree less than n.

[1] Since $\deg(Q(x)) < n$, ***orthogonality*** gives:

$$\int_{-1}^{1} Q(x)P_n(x)\,dx = 0.$$

[2] Further, since x_i is a root of $P_n(x)$:

$$P(x_i) = Q(x_i)P_n(x_i) + R(x_i) = R(x_i).$$

Proof of the Theorem

[3] Now, since $\deg(R(x)) < n$, the first part of the proof implies

$$\int_{-1}^{1} R(x)\,dx = \sum_{i=1}^{n} c_i R(x_i).$$

Putting [1], [2] and [3] together we arrive at

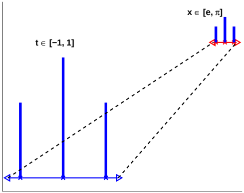
$$\begin{aligned}\int_{-1}^{1} P(x)\,dx &= \int_{-1}^{1} \left[Q(x)P_n(x) + R(x)\right] dx \\ &= \int_{-1}^{1} R(x)\,dx = \sum_{i=1}^{n} c_i R(x_i) \\ &= \sum_{i=1}^{n} c_i P(x_i),\end{aligned}$$

which shows that the formula is exact for all polynomials $P(x)$ of degree less than $2n$. $\square$

Gaussian Quadrature beyond the interval $[-1, 1]$

By a simple linear transformation,

$$t = \frac{2x - a - b}{b - a} \Leftrightarrow x = \frac{(b-a)t + (b+a)}{2},$$



we can apply the Gaussian Quadrature formulas to any interval

$$\int_a^b f(x)\,dx = \int_{-1}^{1} f\left(\frac{(b-a)t + (b+a)}{2}\right) \underbrace{\frac{(b-a)}{2}}_{\text{Rescale summation weights by this factor.}} dt.$$

Example

Degree	$P_n(x)$	Roots / Quadrature points
2	$x^2 - 1/3$	$\{-1/\sqrt{3},\ 1/\sqrt{3}\}$
3	$x^3 - 3x/5$	$\{-\sqrt{3/5},\ 0,\ \sqrt{3/5}\}$
4	$x^4 - 6x^2/7 + 3/35$	$\{-0.86114,\ -0.33998,\ 0.33998,\ 0.86114\}$

Table: Quadrature points on "standard interval:"

$$\int_0^{\pi/4} (\cos(x))^2\, dx = \frac{1}{4} + \frac{\pi}{8} = 0.642699081698724$$

Degree	"Standard" Quadrature points $\in [-1, 1]$	(Unscaled) Weight Coefficients
2	-0.57735, 0.57735	1, 1
3	-0.77459, 0, 0.77459	0.55556, 0.88889, 0.55556
4	-0.86113, -0.33998, 0.33998, 0.86113	0.34785, 0.65215, 0.65215, 0.34785
Degree	**Translated Quadrature points**	**Rescaled Weight Coefficients**
2	0.16597, 0.61942	0.39269, 0.39269
3	0.08851, 0.39270, 0.69688	0.21816, 0.34906, 0.21816
4	0.05453, 0.25919, 0.52621, 0.73087	0.13660, 0.25609, 0.25609, 0.13660

Table: Quadrature points translated to interval of interest; with weight coefficients.

Example

$$\int_0^{\pi/4} (\cos(x))^2\,dx = \frac{1}{4} + \frac{\pi}{8} = 0.642699081698724$$

Degree	Translated Quadrature points	Rescaled Weight Coefficients
2	0.16597, 0.61942	0.39269, 0.39269
3	0.08851, 0.39270, 0.69688	0.21816, 0.34906, 0.21816
4	0.05453, 0.25919, 0.52621, 0.73087	0.13660, 0.25609, 0.25609, 0.13660

Table: Quadrature points translated to interval of interest; with weight coefficients.

Degree	Integral approximation	Error
2	0.642317235049753	0.0003818466489...
3	0.642701112090729	0.0000020303920...
4	0.642699075999924	0.0000000056988...

Table: Approximation and Error, for GQ.

MatLab for Gaussian Quadrature

The **MatLab code** below allows the user to insert a function, $f(x)$ over an interval $x \in [a, b]$ for $n = 2$, 3, or 4 points.

```
function S = gauss234(a,b,n)
% Gaussian Quadrature for n = 2,3,4 pts
% User inputs function f and interval [a,b]
%f = @(x) x.^5 + 5*x.^3 + 2*x + 3;
f = @(x) 2.4*x.^2.*cos(2.4*x);
if (n == 2)
    gr = [-1/sqrt(3),1/sqrt(3)];
    wt = [1,1];
elseif (n == 3)
    gr = [-sqrt(3/5),0,sqrt(3/5)];
```

MatLab for Gaussian Quadrature

```
    wt = [5/9,8/9,5/9];
elseif (n == 4)
    tr = (2/7)*sqrt(6/5);
    tw = sqrt(30)/36;
    gr = [-sqrt((3/7)+tr),-sqrt((3/7)-tr),...
          sqrt((3/7)-tr),sqrt((3/7)+tr)];
    wt = [0.5-tw,0.5+tw,0.5+tw,0.5-tw];
else
    S = fprintf('Selected n inappropriate');
    return
end
gx = ((b-a)*gr + (b+a))/2;
wx = wt*(b-a)/2;
S = sum(wx.*f(gx));
```

More Quadrature Points?!

It turns out it is not that difficult to write a piece of (matlab) code which computes the Lagrange polynomials and their roots; however numerical roundoff causes some issues with the coefficients, after some "hand cleaning" we get:

$$\begin{aligned}
L_5(x) &= x^5 - \frac{10}{9}x^3 + \frac{5}{21}x \\
L_6(x) &= x^6 - \frac{15}{11}x^4 + \frac{5}{11}x^2 - \frac{5}{231} \\
L_7(x) &= x^7 - \frac{21}{13}x^5 + \frac{105}{143}x^3 - \frac{35}{429}x \\
L_8(x) &= x^8 - \frac{28}{15}x^6 + \frac{14}{13}x^4 - \frac{28}{143}x^2 + \frac{7}{1287} \\
L_9(x) &= x^9 - \frac{36}{17}x^7 + \frac{126}{85}x^5 - \frac{84}{221}x^3 + \frac{17}{656}x \\
L_{10}(x) &= x^{10} - \frac{45}{19}x^8 + \frac{630}{323}x^6 - \frac{210}{323}x^4 + \frac{106}{1413}x^2 - \frac{1}{733}
\end{aligned}$$

SDSU

More Quadrature Points?!

It is, of course, tempting to use many quadrature points, but the quality of the points has to be considered. Here, using the points given by matlab's `roots` command: