# Lesson 19: Numerical integration

```
> restart;
```

## Left, Right, Midpoint and Trapezoid

Suppose we want to approximate $J = \int_a^b f(x)\,\mathrm{d}x$ numerically. We defined several ways of doing this: left and right Riemann sums, the Midpoint Rule and the Trapezoid Rule.

```
> h := n -> (b-a)/n;
```

$$h := n \rightarrow \frac{b-a}{n}$$

```
> X := (k,n) -> a + k*h(n);
```

$$X := (k, n) \rightarrow a + k\, h(n)$$

```
> LeftSum:= n -> add(f(X(k-1,n))*h(n), k=1..n);
```

$$LeftSum := n \rightarrow add(f(X(k-1, n))\, h(n), k = 1\, ..n) \tag{1.1}$$

```
> RightSum:= n -> add(f(X(k,n))*h(n), k=1..n);
```

$$RightSum := n \rightarrow add(f(X(k, n))\, h(n), k = 1\, ..n) \tag{1.2}$$

```
> M := n -> add(f((X(k-1,n)+X(k,n))/2)*h(n), k=1..n);
```

$$M := n \rightarrow add\left(f\left(\frac{1}{2}\, X(k-1, n) + \frac{1}{2}\, X(k, n)\right) h(n), k = 1\, ..n\right)$$

```
> Mformal := n -> Sum(f((X(k-1,n)+X(k,n))/2)*h(n), k=1..n);
  eval(Mformal(n),{a=0,b=1});
```

$$Mformal := n \rightarrow \sum_{k=1}^{n} f\left(\frac{1}{2}\, X(k-1, n) + \frac{1}{2}\, X(k, n)\right) h(n)$$

$$\sum_{k=1}^{n} \frac{f\left(\frac{1}{2}\, \frac{k-1}{n} + \frac{1}{2}\, \frac{k}{n}\right)}{n}$$

```
> T := n -> add((f(X(k-1,n)) + f(X(k,n)))/2 * h(n), k=1..n);
  Tformal := n -> Sum((f(X(k-1,n)) + f(X(k,n)))/2 * h(n), k=1..
  n);
```

$$T := n \rightarrow add\left(\frac{1}{2}\, (f(X(k-1, n)) + f(X(k, n)))\, h(n), k = 1\, ..n\right)$$

$$Tformal := n \rightarrow \sum_{k=1}^{n} \frac{1}{2}\, (f(X(k-1, n)) + f(X(k, n)))\, h(n)$$

I took the integral $J = \int_0^1 \frac{1}{x+1}\,\mathrm{d}x$, which should be $\ln(2)$, and calculated the error (the difference between the true value and the approximation) for left and right sums, Midpoint and Trapezoid Rules with different values of $n$ from 1 to 20.

```
> a := 0: b := 1: f := x -> 1/(x + 1):
  J := int(f(x),x=a..b);
```

$$J := \ln(2) \tag{1.3}$$

```
> LeftSumErrors := [seq([n, evalf(J - LeftSum(n))], n = 1.. 20)
  ];
  RightSumErrors := [seq([n, evalf(J - RightSum(n))], n = 1..
  20)];
```

*LeftSumErrors* := [[1, -0.3068528194], [2, -0.1401861527], [3, -0.0901861527], [4, -0.0663766289], [5, -0.0524877400], [6, -0.0433968309], [7, -0.0369865745], [8, -0.0322246698], [9, -0.0285481992], [10, -0.0256242226], [11, -0.0232432702], [12, -0.0212669856], [13, -0.0196003189], [14, -0.0181758175], [15, -0.0169442904], [16, -0.0158690216], [17, -0.0149220519], [18, -0.0140817158], [19, -0.0133309650], [20, -0.0126562012]]

*RightSumErrors* := [[1, 0.1931471806], [2, 0.1098138473], [3, 0.0764805139], [4, 0.0586233711], [5, 0.0475122600], [6, 0.0399365024], [7, 0.0344419969], [8, 0.0302753302], [9, 0.0270073564], [10, 0.0243757774], [11, 0.0222112753], [12, 0.0203996811], [13, 0.0188612195], [14, 0.0175384682], [15, 0.0163890429], [16, 0.0153809784], [17, 0.0144897128], [18, 0.0136960620], [19, 0.0129848244], [20, 0.0123437988]] **(1.4)**

```
> MidpointErrors := [seq([n, evalf(J - M(n))], n = 1 .. 20)];
```

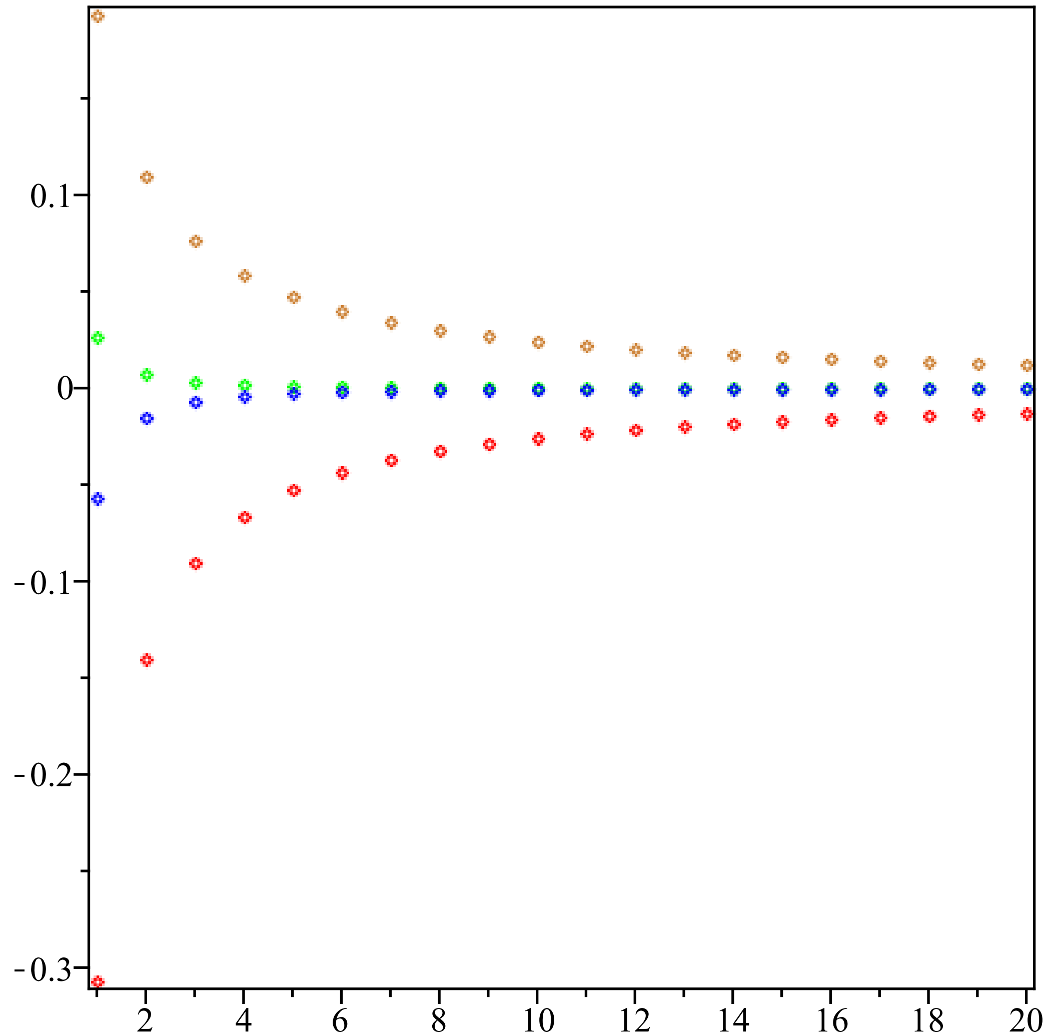
*MidpointErrors* := [[1, 0.0264805139], [2, 0.0074328949], [3, 0.0033924908], [4, 0.0019272894], [5, 0.0012392949], [6, 0.0008628597], [7, 0.0006349395], [8, 0.0004866266], [9, 0.0003847676], [10, 0.0003118202], [11, 0.0002577997], [12, 0.0002166855], [13, 0.0001846727], [14, 0.0001592614], [15, 0.0001387542], [16, 0.0001219663], [17, 0.0001080498], [18, 0.0000963856], [19, 0.0000865128], [20, 0.0000780824]] **(1.5)**

```
> TrapezoidErrors := [seq([n, evalf(J - T(n))], n = 1 .. 20)];
```

*TrapezoidErrors* := [[1, -0.0568528194], [2, -0.0151861527], [3, -0.0068528194], [4, -0.0038766289], [5, -0.0024877400], [6, -0.0017301643], [7, -0.0012722888], [8, -0.0009746698], [9, -0.0007704214], [10, -0.0006242226], [11, -0.0005159975], [12, -0.0004336523], [13, -0.0003695497], [14, -0.0003186747], [15, -0.0002776238], [16, -0.0002440216], [17, -0.0002161696], [18, -0.0001928269], [19, -0.0001730703], [20, -0.0001562012]] **(1.6)**

The errors all decrease in size, of course, as $n$ increases, but the Midpoint and Trapezoid errors decrease faster than the Left and Right Sum errors.

```
> with(plots):
  display(pointplot(LeftSumErrors,colour=red),
  pointplot(RightSumErrors,colour=gold),
  pointplot(MidpointErrors,colour=green), pointplot
  (TrapezoidErrors,colour=blue),axes=box);
```

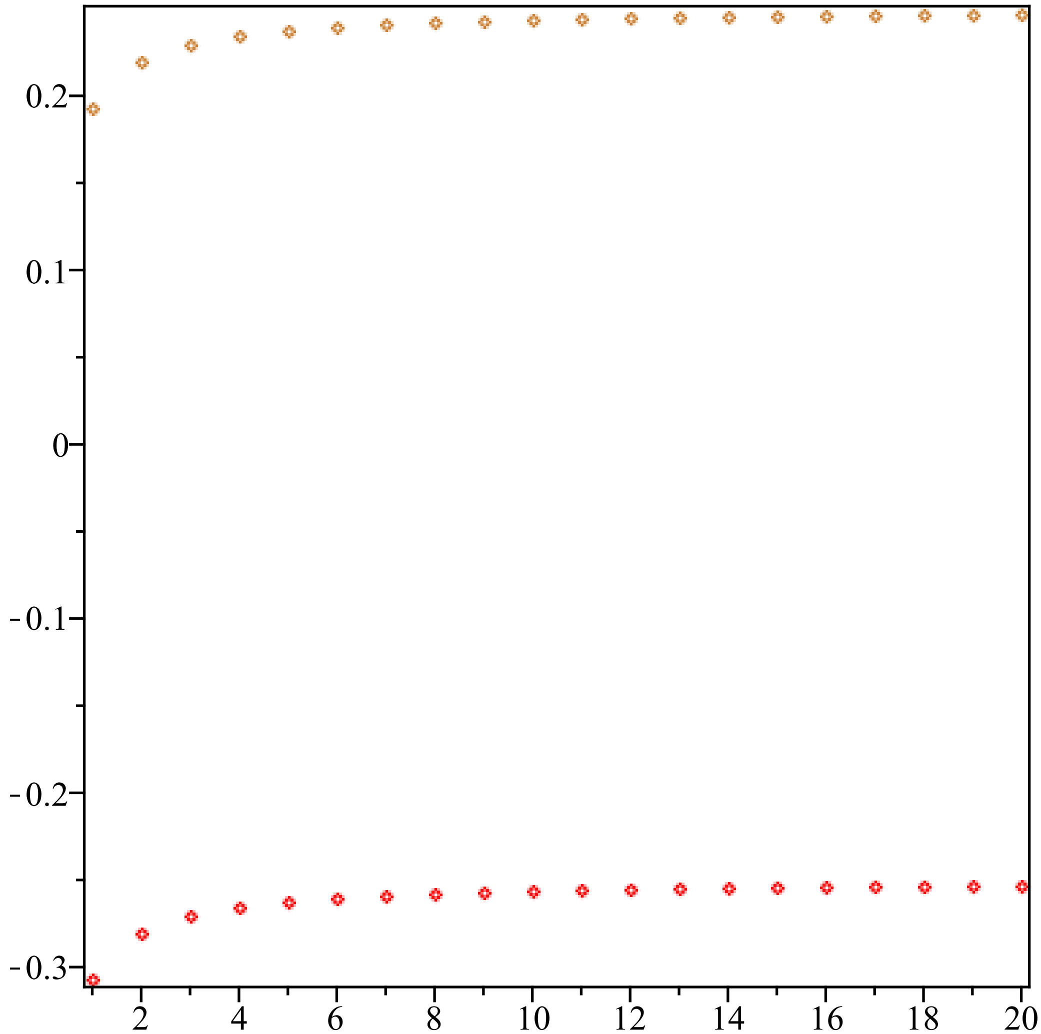



It turns out the errors in both Left and Right Sum methods are approximately proportional to $n^{-1}$, while for Midpoint and Trapezoid they are approximately proportional to $n^{-2}$. To see that, one way is to plot $n$ or $n^2$ times the error.

```
> LSn := [seq([n, evalf(J-LeftSum(n))*n], n=1..20)];
  RSn := [seq([n, evalf(J-RightSum(n))*n], n=1..20)];
  display(pointplot(LSn,colour=red),pointplot(RSn,colour=gold),
  axes=box);
```

*LSn* := [[1, -0.3068528194], [2, -0.2803723054], [3, -0.2705584581], [4, -0.2655065156], [5, -0.2624387000], [6, -0.2603809854], [7, -0.2589060215], [8, -0.2577973584], [9, -0.2569337928], [10, -0.2562422260], [11, -0.2556759722], [12, -0.2552038272], [13, -0.2548041457], [14, -0.2544614450], [15, -0.2541643560], [16, -0.2539043456], [17, -0.2536748823], [18, -0.2534708844], [19, -0.2532883350], [20, -0.2531240240]]

*RSn* := [[1, 0.1931471806], [2, 0.2196276946], [3, 0.2294415417], [4, 0.2344934844], [5, 0.2375613000], [6, 0.2396190144], [7, 0.2410939783], [8, 0.2422026416], [9, 0.2430662076], [10, 0.2437577740], [11, 0.2443240283], [12, 0.2447961732], [13, 0.2451958535], [14, 0.2455385548], [15, 0.2458356435], [16, 0.2460956544], [17, 0.2463251176], [18, 0.2465291160], [19, 0.2467116636], [20, 0.2468759760]]

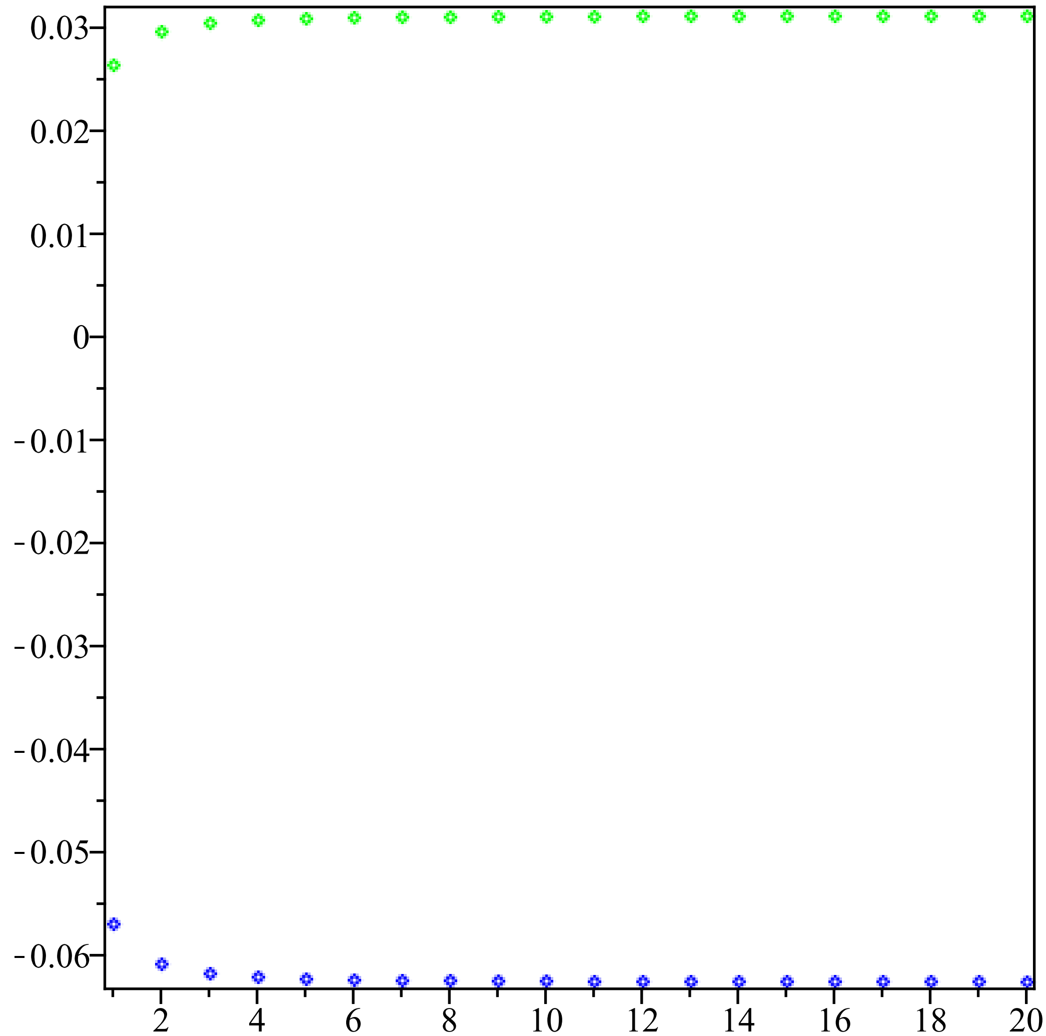



```
> Mnsq := [seq([n, evalf(J-M(n))*n^2],n=1..20)];
  Tnsq := [seq([n, evalf(J-T(n))*n^2],n=1..20)];
```

*Mnsq* := [[1, 0.0264805139], [2, 0.0297315796], [3, 0.0305324172], [4, 0.0308366304], [5, 0.0309823725], [6, 0.0310629492], [7, 0.0311120355], [8, 0.0311441024], [9, 0.0311661756], [10, 0.0311820200], [11, 0.0311937637], [12, 0.0312027120], [13, 0.0312096863], [14, 0.0312152344], [15, 0.0312196950], [16, 0.0312233728], [17, 0.0312263922], [18, 0.0312289344], [19, 0.0312311208], [20, 0.0312329600]]

*Tnsq* := [[1, -0.0568528194], [2, -0.0607446108], [3, -0.0616753746], [4, -0.0620260624], [5, -0.0621935000], [6, -0.0622859148], [7, -0.0623421512], [8, -0.0623788672], [9, -0.0624041334], [10, -0.0624222600], [11, -0.0624356975], [12, -0.0624459312], [13, -0.0624538993], [14, -0.0624602412], [15, -0.0624653550], [16, -0.0624695296], [17, -0.0624730144], [18, -0.0624759156], [19, -0.0624783783], [20, -0.0624804800]]

```
> display(pointplot(Mnsq,colour=green), pointplot(Tnsq,colour=
  blue),axes=box);
```




The theoretical results for the Left Sum and Right Sum are as follows:
If $f'$ is continuous, the error in Left Sum is $\frac{f'(t)\,(b-a)^2}{2\,n}$ for some $t$ between $a$ and $b$, the error in Right Sum is $-\frac{f'(t)\,(b-a)^2}{2\,n}$ (for a different $t$ in the same interval). Here's how that works in the case of the Left Sum (the Right Sum is similar).

Let the minimum and maximum values of $f'$ on our interval be $c$ and $d$. Then at any point $x$ of the subinterval $[x_{k-1}, x_k]$ we have $f(x_{k-1}) + c\,(x - x_{k-1}) \le f(x) \le f(x_{k-1}) + d\,(x - x_{k-1})$
Integrating this over the subinterval,

$$f(x_{k-1})\,h + \frac{c\,h^2}{2} \le \int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x \le f(x_{k-1})\,h + \frac{d\,h^2}{2}$$

Now add these for all $n$ subintervals, remembering that $h = \frac{(b-a)}{n}$.

$$L(n) + \frac{c\,(b-a)^2}{2\,n} \le J \le L(n) + \frac{d\,(b-a)^2}{2\,n}$$

(where $L(n)$ is the left sum), i.e.

$$\frac{c\,(b-a)^2}{2\,n} \le J - L(n) \le \frac{d\,(b-a)^2}{2\,n}$$

and since the continuous function $f'$ takes on all values between its minimum and its maximum somewhere in the interval, $J - L(n) = \frac{f'(t)\,(b-a)^2}{2\,n}$ for some $t$ in the interval.

Here's the theoretical result for the Midpoint and Trapezoid Rules. Assume $f''$ is continuous on the interval $[a, b]$. Then the error in the Midpoint Rule is $\frac{f''(t)\,(b-a)^3}{24\,n^2}$ for some $t$ between $a$ and $b$, and the error in the Trapezoid Rule is $-\frac{f''(t)\,(b-a)^3}{12\,n^2}$ (for a different $t$ in the same interval). If the assumption that $f''$ is continuous is not true, these error estimates might not be true.

Here's an example of a function whose second derivative is not continuous, in fact it doesn't have a first derivative at 0.

```
> f := x -> sqrt(x);
```

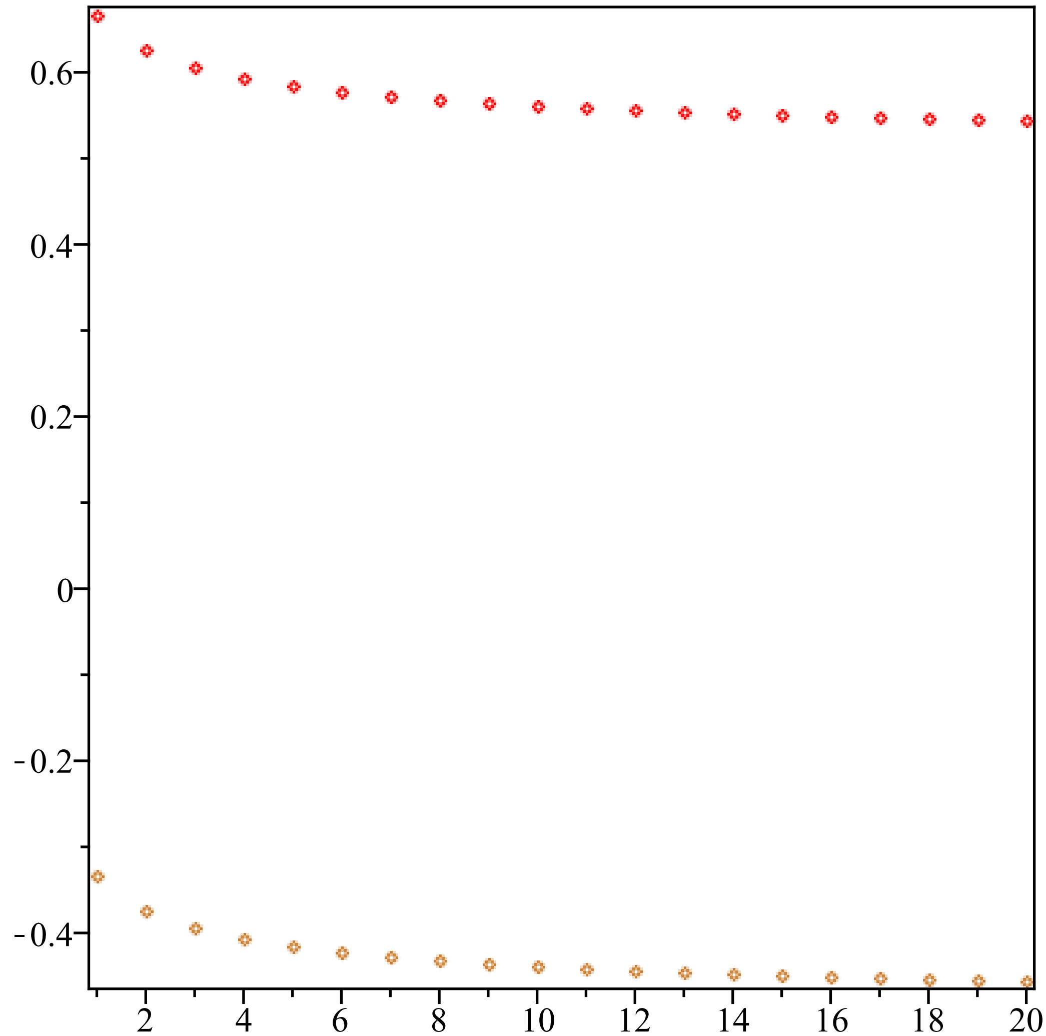
$$f := x \rightarrow \sqrt{x}$$

```
> J := int(f(x),x=a..b);
  LSn := [seq([n, evalf(J-LeftSum(n))*n], n=1..20)];
  RSn := [seq([n, evalf(J-RightSum(n))*n], n=1..20)];
  display(pointplot(LSn,colour=red),pointplot(RSn,colour=gold),
  axes=box);
```

$$J := \frac{2}{3}$$

$LSn$ := [[1, 0.6666666667], [2, 0.6262265524], [3, 0.6061531497], [4, 0.5935344820], [5, 0.5846403465], [6, 0.5779271490], [7, 0.5726227374], [8, 0.5682915608], [9, 0.5646664926], [10, 0.5615732515], [11, 0.5588925673], [12, 0.5565396620], [13, 0.5544523739], [14, 0.5525839302], [15, 0.5508983688], [16, 0.5493675208], [17, 0.5479689424], [18, 0.5466845279], [19, 0.5454994673], [20, 0.5444015080]]

$RSn$ := [[1, -0.3333333333], [2, -0.3737734476], [3, -0.3938468505], [4, -0.4064655180], [5, -0.4153596535], [6, -0.4220728512], [7, -0.4273772622], [8, -0.4317084392], [9, -0.4353335086], [10, -0.4384267487], [11, -0.4411074330], [12, -0.4434603380], [13, -0.4455476268], [14, -0.4474160705], [15, -0.4491016292], [16, -0.4506324787], [17, -0.4520310582], [18, -0.4533154702], [19, -0.4545005329], [20, -0.4555984920]]

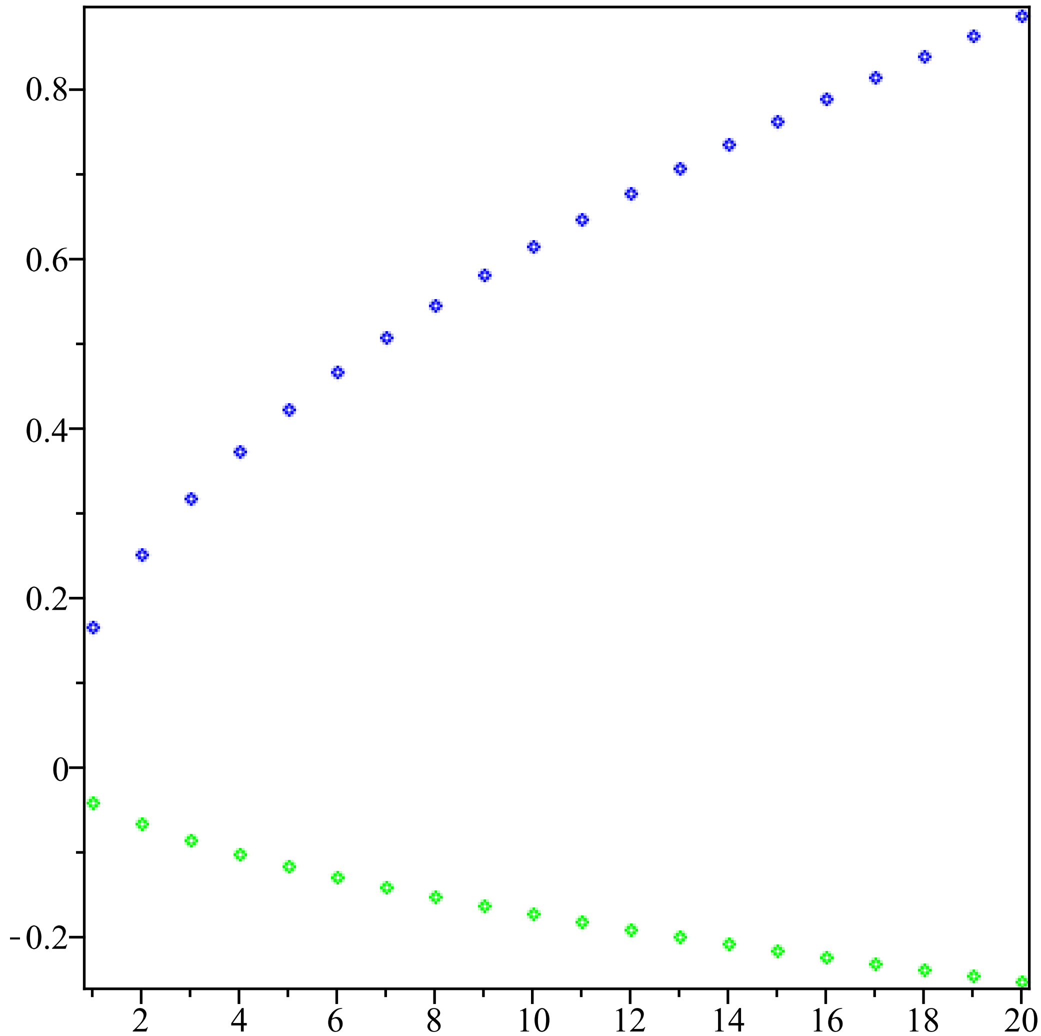



In this case it looks like the errors in left and right Riemann sums are behaving more or less as they should (as a more detailed error analysis would show), but:

```
> Mnsq := [seq([n, evalf(J-M(n))*n^2],n=1..20)];
  Tnsq := [seq([n, evalf(J-T(n))*n^2],n=1..20)];
  display(pointplot(Mnsq,colour=green), pointplot(Tnsq,colour=
  blue),axes=box);
```

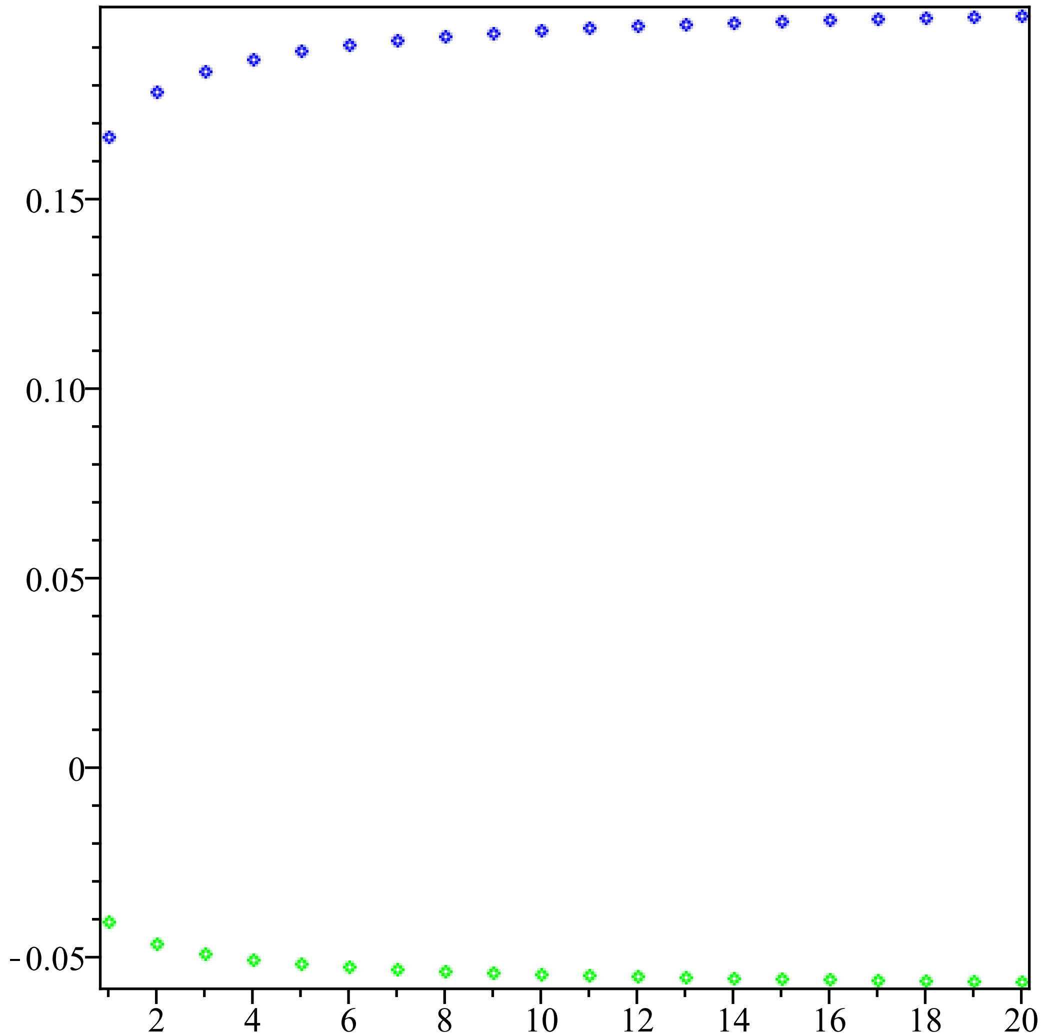
*Mnsq* := [[1, -0.0404401143], [2, -0.0653841412], [3, -0.0846780021], [4, -0.1009716848], [5, -0.1153354775], [6, -0.1283249232], [7, -0.1402716532], [8, -0.1513923456], [9, -0.1618376679], [10, -0.1717174520], [11, -0.1811146053], [12, -0.1900936483], [13, -0.1987058161], [14, -0.2069927639], [15, -0.2149887038], [16, -0.2227222810], [17, -0.2302178017], [18, -0.2374958848], [19, -0.2445745723], [20, -0.2514692560]]

*Tnsq* := [[1, 0.1666666667], [2, 0.2524531048], [3, 0.3184594488], [4, 0.3741379280], [5, 0.4232017325], [6, 0.4675628916], [7, 0.5083591632], [8, 0.5463324864], [9, 0.5819984298], [10, 0.6157325130], [11, 0.6478182301], [12, 0.6784759339], [13, 0.7078808749], [14, 0.7361750298], [15, 0.7634755485], [16, 0.7898803174], [17, 0.8154719936], [18, 0.8403214849], [19, 0.8644898768], [20, 0.8880301440]]




These don't appear to be approaching constant values. In fact, it turns out the dependence on $n$ is not like $n^{-2}$, but $n^{-1.5}$.

```
> display(pointplot([seq([n,evalf(J-M(n))*n^1.5],n=1..20)],
  colour=green),
  pointplot([seq([n,evalf(J-T(n))*n^1.5],n=1..20)],colour=
  blue),axes=box);
```




# Big-O notation

Suppose you want to approximate a quantity $J$, and you can use approximations $A(n)$ depending on a positive integer $n$ (e.g. the number of intervals used). Typically, you know something about how the error $J - A(n)$ depends on $n$. For example, with the Trapezoid rule (when $f''$ is continuous) we know $|J - T(n)|$ is less than some constant times $n^{-2}$. We could write this as $J = T(n) + \mathrm{O}\left(n^{-2}\right)$. This $\mathrm{O}\left(n^{-2}\right)$ term means "something whose absolute value is less than some constant times $n^{-2}$ when $n$ is large". Similarly, for the left Riemann sum, $J = L(n) + \mathrm{O}\left(n^{-1}\right)$. Here's the definition of O:

Let $f(n)$ and $g(n)$ be functions of $n$. We say $f(n) = \mathrm{O}(g(n))$ as $n \to \infty$ if there exist constants $M$ and $N$ such that $|f(n)| \le M|g(n)|$ whenever $n > N$.

Note that O is not a function, it's a way of expressing relationships between functions.

There's a similar notation for functions of $x$ as $x \to 0$: $f(x) = \mathrm{O}(g(x))$ as $x \to 0$ if there exist constants $M$ and $\varepsilon > 0$ such that $|f(x)| \le M|g(x)|$ whenever $|x| < \varepsilon$.

# Simpson's Rule

When $f''$ is continuous,the Midpoint Rule's error is approximately -1/2 times the Trapezoid Rule's error. This might suggest that a combination of the two rules would cancel out the error (at least approximately) and produce a much better approximation. The appropriate combination is 2/3* Midpoint + 1/3*Trapezoid. This is called Simpson's Rule.

```
> f:= 'f': 2/3*M(1)+1/3*T(1);
```

$$\frac{2}{3} f\left(\frac{1}{2}\right) + \frac{1}{6} f(0) + \frac{1}{6} f(1)$$

I'm going to call this the Simpson's Rule approximation for $n = 2$ (rather than $n = 1$): it uses the same three equally-spaced points as the Trapezoid Rule for $n = 2$. We'll only use $S(n)$ when $n$ is even.

```
> S := n -> add((1/3*f(X(2*k-2,n)) + 4/3*f(X(2*k-1,n)) + 1/3*f
  (X(2*k,n))) * h(n), k=1..n/2);
```

$$S := n \rightarrow add\left(\left(\frac{1}{3} f(X(2k-2, n)) + \frac{4}{3} f(X(2k-1, n)) + \frac{1}{3} f(X(2k, n))\right) h(n), k = 1..\frac{1}{2} n\right)$$

```
> S(2);
```

$$\frac{2}{3} f\left(\frac{1}{2}\right) + \frac{1}{6} f(0) + \frac{1}{6} f(1)$$

```
> S(6);
```

$$\frac{1}{18} f(0) + \frac{2}{9} f\left(\frac{1}{6}\right) + \frac{1}{9} f\left(\frac{1}{3}\right) + \frac{2}{9} f\left(\frac{1}{2}\right) + \frac{1}{9} f\left(\frac{2}{3}\right) + \frac{2}{9} f\left(\frac{5}{6}\right) + \frac{1}{18} f(1) \qquad \textbf{(3.1)}$$

The theoretical value for the error in Simpson's Rule is $\frac{\mathrm{D}^{(4)}(f)(t)\,(b-a)^5}{180\, n^4}$. Thus it is $\mathrm{O}(n^{-4})$.

```
> f := x -> 1/(x+1): J:= ln(2): Digits:= 20;
  SimpsonErrors := [seq([2*j, evalf(J - S(2*j))], j = 1 .. 20)]
  ;
```

$$Digits := 20$$

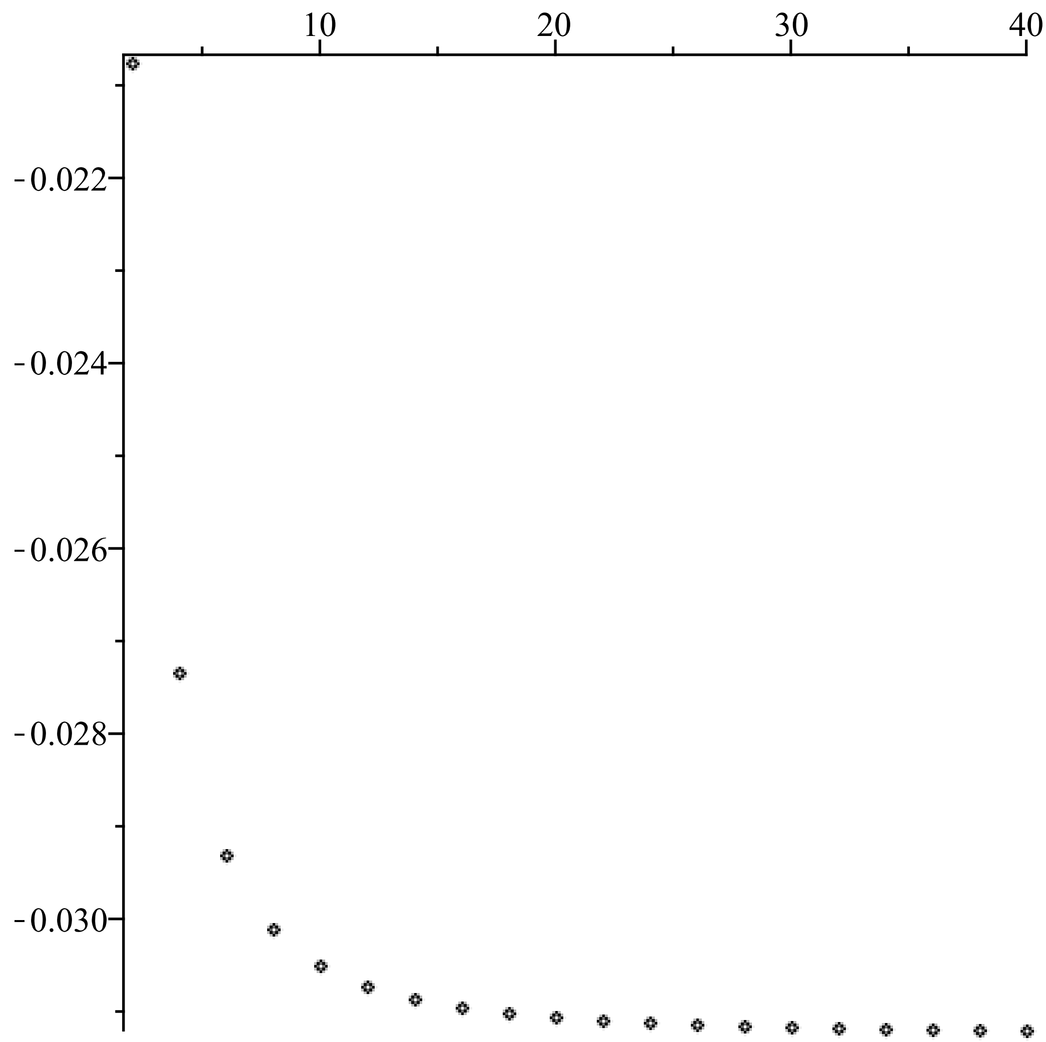
$SimpsonErrors := [[2, -0.0012972638844991350 2], [4, -0.00010678769402294455], [6, -0.000022612609847860 37], [8, -0.0000073500945853451 1], [10, -0.0000030501289850699 1], [12, -0.0000014816491557205 2], [14, -8.0331514405317\ 10^{-7}], [16, -4.7225947373165\ 10^{-7}], [18, -2.9542106159912\ 10^{-7}], [20, -1.9410517080955\ 10^{-7}], [22, -1.3271826881005\ 10^{-7}], [24, -9.378437233684\ 10^{-8}], [26, -6.813308618135\ 10^{-8}], [28, -5.068023853983\ 10^{-8}], [30, -3.847360662788\ 10^{-8}], [32, -2.972987763032\ 10^{-8}], [34, -2.333445859223\ 10^{-8}], [36, -1.856967590158\ 10^{-8}], [38, -1.496116472277\ 10^{-8}], [40, -1.218801071711\ 10^{-8}]]$

```
> Sn4 := [seq([2*j, (2*j)^4*evalf(J - S(2*j))], j = 1 .. 20)];
```

$Sn4 := [[2, -0.020756222151986160 32], [4, -0.027337649669873804 80], [6, -0.029305942362827039 52], [8, -0.030105987421573570 56], [10, -0.030501289850699100 00], [12, -0.030723476893020702 72], [14, -0.030860154573946578 72], [16, -0.030949996870477414 40], [18,$

-0.03101212136242922112], [20, -0.03105682732952800000], [22,
-0.03109005077836707280], [24, -0.03111540391642742784], [26,
-0.03113518519080859760], [28, -0.03115091269993774848], [30,
-0.03116362136858280000], [32, -0.03117403616609042432], [34,
-0.03118267705730626928], [36, -0.03118992475910819328], [38,
-0.03119606316537774672], [40, -0.03120130743580160000]]

```
> with(plots): pointplot(Sn4);
```




To get an idea of how much better this is than Left Sum, Right Sum, Midpoint or Trapezoid, let's see what $n$ would be needed to have an error less than $10^{-10}$ in absolute value (without roundoff error).
We'll keep Digits = 20, otherwise roundoff error would be a problem.

For the Left Sum with this function $f$, the error was approximately $-\frac{.25}{n}$. So:

```
> solve(.25/n = 10^(-10));
```

$$2.500000000\ 10^{9} \qquad \textbf{(3.2)}$$

I won't actually calculate a sum with 2.5 billion terms. It might take a while, and roundoff error would be quite severe.

For the Trapezoid Rule with this function $f$, the error was approximately $-\frac{0.06}{n^2}$. So:

```
> solve(0.06/n^2 = 10^(-10));
```

$$-24494.897427831780982, 24494.897427831780982$$

We'd need $n \geq 24495$.

```
> evalf(T(24495)-J);
```

$$1.0416579425\ 10^{-10}$$

Well, that's a bit more than $10^{-10}$. The $-\frac{.06}{n^2}$ was, after all, only an approximation. Let's try making $n$ a little larger.

```
> evalf(T(25000)-J);
```

$$9.999999998\ 10^{-11} \tag{3.3}$$

```
> evalf(T(25500)-J);
```

$$9.611687810\ 10^{-11} \tag{3.4}$$

For the Midpoint Rule, the error was approximately $\frac{0.03}{n^2}$. So:

```
> solve(0.03/n^2 = 10^(-10));
```

$$-17320.508075688772935, 17320.508075688772935$$

We'd need $n \geq 17321$. Again,

```
> evalf(M(17321)-J);
```

$$-1.0416074992\ 10^{-10}$$

Again, it really needs to be a little larger.

```
> evalf(M(18000)-J);
```

$$-9.645061722\ 10^{-11} \tag{3.5}$$

For Simpson's Rule, the error was approximately $-\frac{0.031}{n^4}$. So:

```
> solve(0.031/n^4 = 10^(-10));
```

$$132.69068114098672131, 132.69068114098672131\,\mathrm{I}, -132.69068114098672131, -132.69068114098672131\,\mathrm{I}$$

We'd need n >= 133, actually 134 since n must be even.

```
> evalf(J - S(134));
```

$$-9.691039697\ 10^{-11} \tag{3.6}$$

Here's a different point of view about the three different rules. If you were integrating a polynomial, when would the rule give the correct answer (neglecting round-off error)?

The Midpoint and Trapezoid rules give the correct answers for polynomials of degree up to 1, but in general not for higher degrees.

```
> f := unapply(add(c[j]*x^j, j=0..5),x);
```

$$f := x \rightarrow c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5 \tag{3.7}$$

```
> J:= int(f(x),x=a..b);
```

$$\tag{3.8}$$

$$J := c_0 + \frac{1}{2} c_1 + \frac{1}{3} c_2 + \frac{1}{4} c_3 + \frac{1}{5} c_4 + \frac{1}{6} c_5 \quad \textbf{(3.8)}$$

```
> J - T(2);
```

$$-\frac{1}{24} c_2 - \frac{1}{16} c_3 - \frac{13}{160} c_4 - \frac{19}{192} c_5 \quad \textbf{(3.9)}$$

```
> J - M(2);
```

$$\frac{1}{48} c_2 + \frac{1}{32} c_3 + \frac{51}{1280} c_4 + \frac{73}{1536} c_5 \quad \textbf{(3.10)}$$

The thing to notice is that there is no $c_0$ or $c_1$. So if the polynomial has degree $\le 1$, the error would be 0.

Simpson's Rule is exact up to degree 3.

```
> J - S(2);
```

$$-\frac{1}{120} c_4 - \frac{1}{48} c_5$$

We could have used this to derive Simpson's Rule in another way. Suppose we didn't know the coefficients, but we knew the general form of the rule we wanted.

```
> rule:= n -> add((d[0]*f(X(2*k-2,n)) + d[1]*f(X(2*k-1,n)) + d
  [2]*f(X(2*k,n))) * h(n), k=1..n/2);
```

$$rule := n \rightarrow add\left( \left(d_0 f(X(2k-2,n)) + d_1 f(X(2k-1,n)) + d_2 f(X(2k,n))\right) h(n), k = 1..\frac{1}{2} n \right)$$

```
> J - rule(2);
```

$$c_0 + \frac{1}{2} c_1 + \frac{1}{3} c_2 + \frac{1}{4} c_3 + \frac{1}{5} c_4 + \frac{1}{6} c_5 - \frac{1}{2} d_0 c_0 - \frac{1}{2} d_1 \left(c_0 + \frac{1}{2} c_1 + \frac{1}{4} c_2 + \frac{1}{8} c_3 + \frac{1}{16} c_4 + \frac{1}{32} c_5\right) - \frac{1}{2} d_2 \left(c_0 + c_1 + c_2 + c_3 + c_4 + c_5\right) \quad \textbf{(3.11)}$$

If we want the rule to give the right answer for polynomials of degree up to 2, we need the coefficients of $c_0$, $c_1$ and $c_2$ here to cancel out.

```
> eqns := { seq(coeff(%, c[j]), j=0..2) };
```

$$eqns := \left\{ \frac{1}{2} - \frac{1}{4} d_1 - \frac{1}{2} d_2, \frac{1}{3} - \frac{1}{8} d_1 - \frac{1}{2} d_2, 1 - \frac{1}{2} d_0 - \frac{1}{2} d_1 - \frac{1}{2} d_2 \right\}$$

I'm only doing this for $j$ up to 2, not 3:
there are only three parameters, $d_0$ to $d_2$, so I want three equations. The fact that it is exact for $x^3$ too is an added bonus.

```
> R:= solve(eqns);
```

$$R := \left\{ d_0 = \frac{1}{3}, d_1 = \frac{4}{3}, d_2 = \frac{1}{3} \right\}$$

```
> f:= 'f': eval(rule(2),R) = S(2);
```

$$\frac{1}{6} f(0) + \frac{2}{3} f\left(\frac{1}{2}\right) + \frac{1}{6} f(1) = \frac{1}{6} f(0) + \frac{2}{3} f\left(\frac{1}{2}\right) + \frac{1}{6} f(1) \quad \textbf{(3.12)}$$

# Newton-Cotes Rules

This can be generalized. Let's say I want a rule that uses some linear combination of $f(x_j)$ for $0 \le j \le k$.

```
> rule:= k -> add(d[j]*f(X(j,k)), j=0..k) * h(k);
```

$$rule := k \to add\left(d_j f(X(j,k)), j = 0..k\right) h(k)$$

```
> rule(1); rule(2); rule(3);
```

$$d_0 f(0) + d_1 f(1)$$

$$\frac{1}{2} d_0 f(0) + \frac{1}{2} d_1 f\left(\frac{1}{2}\right) + \frac{1}{2} d_2 f(1)$$

$$\frac{1}{3} d_0 f(0) + \frac{1}{3} d_1 f\left(\frac{1}{3}\right) + \frac{1}{3} d_2 f\left(\frac{2}{3}\right) + \frac{1}{3} d_3 f(1)$$

With $k+1$ degrees of freedom in choosing the coefficients $d_j$ I can hope to get the integrals of $k+1$ functions $x^j$ for $0 \le j \le k$ correct. The result is called a Newton-Cotes rule of order $k$.

```
> eqns:= k -> {seq(eval(int(f(x),x=0..1) - rule(k), f = unapply
  (x^j, x)), j=0..k)};
```

$$eqns := k \to \left\{ seq\left( \left( \int_0^1 f(x)\, dx - rule(k) \right) \Bigg|_{f = unapply(x^j, x)}, j = 0..k \right) \right\}$$

```
> eqns(2);
```

$$\left\{ \frac{1}{2} - \frac{1}{4} d_1 - \frac{1}{2} d_2, \frac{1}{3} - \frac{1}{8} d_1 - \frac{1}{2} d_2, 1 - \frac{1}{2} d_0 - \frac{1}{2} d_1 - \frac{1}{2} d_2 \right\}$$

```
> solve(%);
```

$$\left\{ d_0 = \frac{1}{3}, d_1 = \frac{4}{3}, d_2 = \frac{1}{3} \right\}$$

```
> eqns(3);
```

$$\left\{ \frac{1}{2} - \frac{1}{9} d_1 - \frac{2}{9} d_2 - \frac{1}{3} d_3, \frac{1}{3} - \frac{1}{27} d_1 - \frac{4}{27} d_2 - \frac{1}{3} d_3, \frac{1}{4} - \frac{1}{81} d_1 - \frac{8}{81} d_2 - \frac{1}{3} d_3, 1 - \frac{1}{3} d_0 - \frac{1}{3} d_1 - \frac{1}{3} d_2 - \frac{1}{3} d_3 \right\}$$

```
> solve(%);
```

$$\left\{ d_0 = \frac{3}{8}, d_1 = \frac{9}{8}, d_2 = \frac{9}{8}, d_3 = \frac{3}{8} \right\}$$

```
> NC3 := subs(%, rule(3));
```

$$NC3 := \frac{1}{8} f(0) + \frac{3}{8} f\left(\frac{1}{3}\right) + \frac{3}{8} f\left(\frac{2}{3}\right) + \frac{1}{8} f(1)$$

This is sometimes called the "three-eighths rule". Like Simpson's rule, it is exact for polynomials of degree 3 but not of degree 4.

```
> eval(NC3-int(f(x),x=0..1), f = unapply(add(c[j]*x^j,j=0..4),
  x));
```

$$\frac{1}{270}\,c_4$$

What about the 4th order Newton-Cotes rule?

```
> NC4 := subs(solve(eqns(4),{seq(d[j],j=0..4)}),rule(4));
```

$$NC4 := \frac{7}{90}\,f(0) + \frac{16}{45}\,f\left(\frac{1}{4}\right) + \frac{2}{15}\,f\left(\frac{1}{2}\right) + \frac{16}{45}\,f\left(\frac{3}{4}\right) + \frac{7}{90}\,f(1)$$

```
> eval(NC4-int(f(x),x=0..1), f = unapply(add(c[j]*x^j,j=0..6),
  x));
```

$$\frac{1}{2688}\,c_6$$

In general, when $k$ is odd the Newton-Cotes rules of orders $k-1$ and $k$ are both exact for polynomials of degree $k$ but not degree $k+1$.

These are the "simple" versions of the rules. For the "compound" version of the order $k$ rule, you divide the interval into a number of subintervals that is a multiple of $k$, and use the simple rule on the first $k$ intervals, the next $k$, etc.

It wouldn't be hard to write a function to generate Newton-Cotes rules, but the Student[Calculus1] package already has one, called ApproximateInt.

```
> with(Student[Calculus1]):
```

```
> ApproximateInt(f(x), x=a..b, partition=1,
  method=newtoncotes[4]);
```

$$\frac{7}{90}\,f(0) + \frac{16}{45}\,f\left(\frac{1}{4}\right) + \frac{2}{15}\,f\left(\frac{1}{2}\right) + \frac{16}{45}\,f\left(\frac{3}{4}\right) + \frac{7}{90}\,f(1)$$

```
> ApproximateInt(f(x), x=a..b, partition=2,
  method=newtoncotes[4]);
```

$$\frac{7}{180}\,f(0) + \frac{8}{45}\,f\left(\frac{1}{8}\right) + \frac{1}{15}\,f\left(\frac{1}{4}\right) + \frac{8}{45}\,f\left(\frac{3}{8}\right) + \frac{7}{90}\,f\left(\frac{1}{2}\right) + \frac{8}{45}\,f\left(\frac{5}{8}\right) + \frac{1}{15}\,f\left(\frac{3}{4}\right) + \frac{8}{45}\,f\left(\frac{7}{8}\right) + \frac{7}{180}\,f(1) \qquad \textbf{(4.1)}$$

For the Newton-Cotes rule of order $k$ with $n$ intervals (where $n$ is divisible by $k$), you use **partition = n/k** and **method = newtoncotes[k]**.

The error in the Newton-Cotes rule with order $k$ and $n$ intervals is $\mathrm{O}\left(\frac{1}{n^{k+1}}\right)$ if $k$ is odd, or $\mathrm{O}\left(\frac{1}{n^{k+2}}\right)$ if $k$ is even, i.e. it's $\mathrm{O}\left(\frac{1}{n^{p+1}}\right)$ where the rule is exact for polynomials of degree up to $p$. Let's look at this for $k=8$ with $f(x) = \frac{1}{1+x}$. The higher-order rules are so accurate that I need to increase Digits (otherwise roundoff error will overwhelm the real error).

```
> Digits := 30:
  f := x -> 1/(1+x): J := int(f(x),x=a..b):
  NC8Errors := [seq([8*n, evalf(J - ApproximateInt(f(x),x=a..b,
  method=newtoncotes[8],partition=n))], n = 1 .. 15)];
```

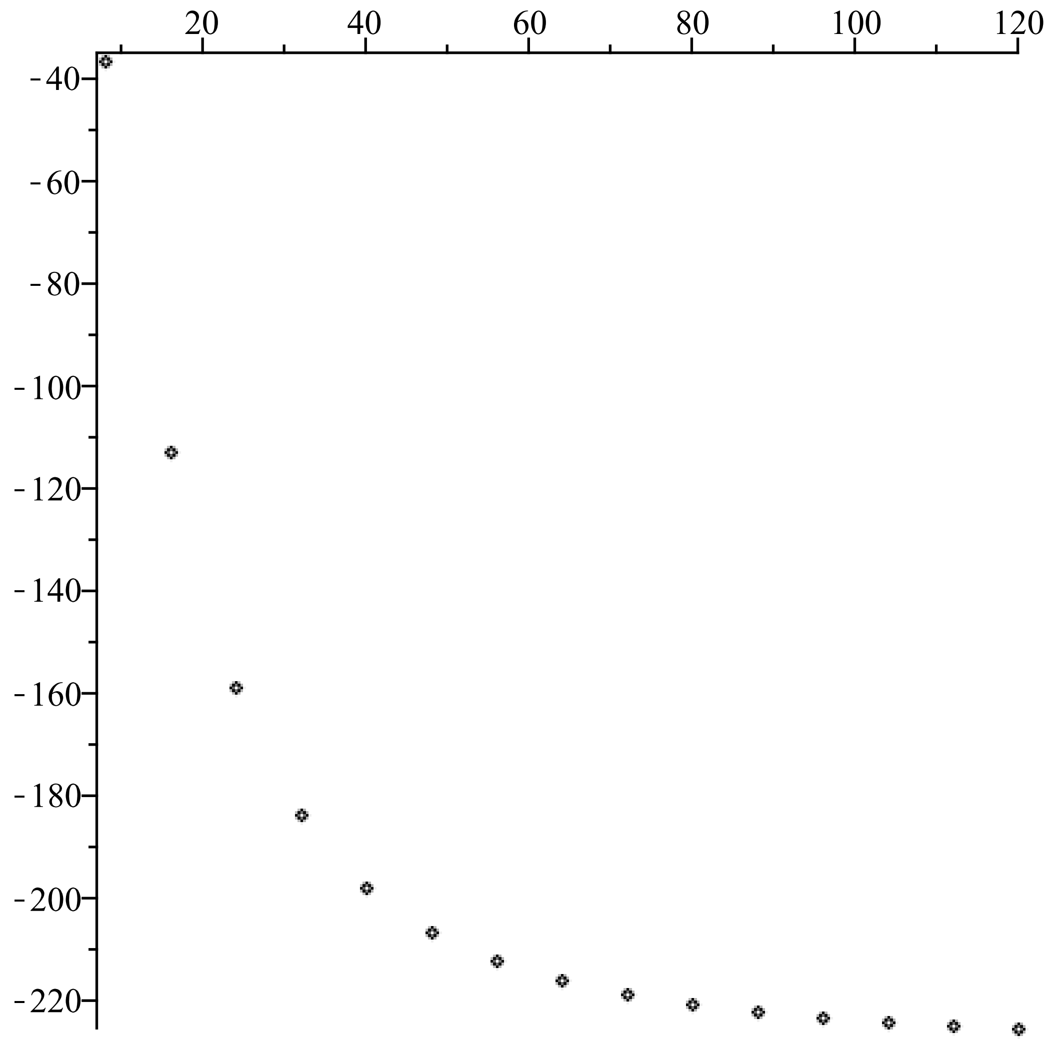
$NC8Errors := [[8, -3.39735126102840738232 34\,10^{-8}], [16,$ **(4.2)**

$-1.02583144702885063155\ 10^{-10}$], [24, $-2.50325482423172262\ 10^{-12}$], [32, $-1.63081755766660319\ 10^{-13}$], [40, $-1.8866008564797685\ 10^{-14}$], [48, $-3.180435378235352\ 10^{-15}$], [56, $-6.99277359532996\ 10^{-16}$], [64, $-1.87265094275717\ 10^{-16}$], [72, $-5.8386865735369\ 10^{-17}$], [80, $-2.0541678317188\ 10^{-17}$], [88, $-7.972899159759\ 10^{-18}$], [96, $-3.357038443104\ 10^{-18}$], [104, $-1.513826681475\ 10^{-18}$], [112, $-7.23796118880\ 10^{-19}$], [120, $-3.64004335058\ 10^{-19}$]]

```
> NC8ScaledErrors:= [seq([t[1],t[2]*t[1]^10],t=NC8Errors)];
```

*NC8ScaledErrors* := [[8, -36.4787813978534225851099287388], [16, -112.791360414650107672990612193], [24, -158.714819274179246654785817936], [32, -183.613733625414412467156398637], [40, -197.824437968412973465600000000], [48, -206.489964464144113652829360488], [56, -212.094661546994630622013655351], [64, -215.901954252702690489941771682], [72, -218.594951113925782651812452499], [80, -220.564591443186936709120000000], [88, -222.046019761742359729124740694], [96, -223.186871725363702603413826044], [104, -224.083329361027097648598456730], [112, -224.800087902994022275224824709], [120, -225.381889929011181649920000000]] **(4.3)**

```
> with(plots): pointplot(NC8ScaledErrors);
```




To some extent, the higher the order, the better. That is, if one method has error estimate $\frac{A}{n^p}$ and a second has $\frac{B}{n^q}$ with $p < q$, then the second will be more accurate when $n$ is sufficiently large.
This isn't necessarily true for a fixed $n$, however, because B might be much bigger than A. In the case of the error estimates for Newton-Cotes rules, the coefficient of $n^{-p}$ depends on the p'th derivative of the function f. So for a function whose higher-order derivatives might grow rapidly, higher order might not be better.

# Errors for Newton-Cotes rules with fixed n.

I want to look at the errors in Newton-Cotes rules with different orders, all using the same $n$, for some functions on the interval 0 .. 1.
I'll take $n$ to be 36, so the order $k$ can be any factor of 36. These are the possibilities.

```
> K := [1,2,3,4,6,9,12,18,36];
```

$$K := [1, 2, 3, 4, 6, 9, 12, 18, 36]$$

First I'll use our function $f(x) = \frac{1}{1+x}$.

```
> seq(evalf(J - ApproximateInt(f(x),x=0..1,method=newtoncotes[K
  [j]],
  partition=36/K[j])), j=1..9);
```

$-0.000048220659074225432544874437$, $-1.8569675901583997777594\ 10^{-8}$, $-4.1701931418315024957474\ 10^{-8}$, $-1.1291685508126363460\ 10^{-10}$, $-1.782655156859319104\ 10^{-12}$, $-1.04704005629163311\ 10^{-13}$, $-1.47509332767563\ 10^{-16}$, $-1.24862983667\ 10^{-19}$, $-3.3836\ 10^{-26}$

For this function, the higher-order rules turned out to be better. But if we take an f whose higher derivatives grow faster, that might not be true.

```
> f:= x -> 1/(x^2 + 1/100):
  J:= int(f(x),x=a..b):
  seq(evalf(J - ApproximateInt(f(x),x=a..b,method=newtoncotes[K
  [j]],
  partition=36/K[j])), j=1..9);
```

$0.0001260432912976827321474639$, $0.0001283845192766359990055555$, $0.0020884379267844848008648205$, $-0.0022812707914423330774552119$, $-0.0010850923254841156779240360$, $-0.0004097357589139511752654873$, $0.0010532909662917247600065260$, $0.0004493644413640830138650209$, $-0.0005849528645731961183032399$ **(5.1)**

Here the best answer was obtained with k = 1, i.e. the Trapezoid rule.