Riemann, Trapezoids, and Simpson

Note: You may notice differences between this Maple worksheet and the equivalent Mathematica notebook. These differences were introduced to preserve the content of these modules and were necessary because of major functional differences between Maple and Mathematica.

Introduction

OBJECTIVE: To visualize the process of using Riemann sums, the trapezoid rule, and Simpson's rule for approximating definite integrals and to understand the error associated with each method.

To evaluate a definite integral, it is often necessary to use a numerical estimate of the integral in place of an exact value. This occurs in three instances: 1) when there is no simple formula for an antiderivative of the integrand; 2) when the antiderivative of the integrand is difficult to determine and/or evaluate; and, 3) when the integrand function is represented by a table of numeric values rather than by a formula. In this module, we investigate several numerical methods for estimating definite integrals.

Technology Guidelines

NOTE: If you have just finished a worksheet, **restart** *Maple* before executing a new worksheet.
TO OPEN SECTIONS,
Click on the **PLUS** sign at the left hand side of the screen *or* select **Expand All Sections** from the **View** drop down menu.

TO STOP AN EXECUTION
Click on **STOP** button from the toolbar.

ORDER OF EXECUTION
Execute commands in the order given. Do not skip any *Maple* Input lines within a given worksheet

Alternatively, you can execute the entire worksheet by selecting the **Execute Worksheet** command from the **Edit** drop down menu.

SAVING WORKSHEETS.

You can save anytime to any directory you choose, and it is wise to save often.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, and then shut down *Maple* and start it up again.

Part I: Riemann Sums and Errors

The most direct method for estimating a definite integral is to replace it with a Riemann sum. That

is, we estimate the integral as follows: $\int_a^b f(x)\,dx = \sum_{i=1}^{n} f(c_i)\,h$, where the interval *[a, b]* is divided into *n* subintervals, each of length $h = \frac{b-a}{n}$, and where c_i is any value of *x* taken from the i^{th} subinterval. To assess the error in the approximation, we will perform a numerical experiment wherein we use a Riemann sum to estimate an integral for which we know the exact value. (You should keep in mind, however, that numerical integration is primarily used for estimating integrals in situations where we can't determine a decimal or fraction representation for the exact value of the integral.) Our first goal is to determine what factors affect the error when we use left- or right-hand Riemann sums to estimate the exact value of an integral.

The integral we choose for our experiment is $\int_0^{\frac{\pi}{2}} \cos(x)\,dx = 1$. First, we look at a graphical depiction of the situation. For this, the **riemannsum( )** command is created using the **leftbox( )** and **rightbox( )** commands from the **student** package. The command is **riemannsum(f, xrange, n, box, incdec)**. The arguments are the function, **f**, the independent variable, **x**, the range of the integral **xrange**, the number of rectangles, **n**, the "right" or "left" sum indicator, **box**, and an "increasing/decreasing" indicator, **incdec**. If the integrand function increases over the interval of the integral, then the argument for **incdec** is the word "increasing," and if it decreases then the argument is the word "decreasing." The error is taken as the Riemann estimate minus the exact value of the integral.

The special command **riemannsum( )** is only available in this module and is not a built-in *Maple* command. Here's how it works.

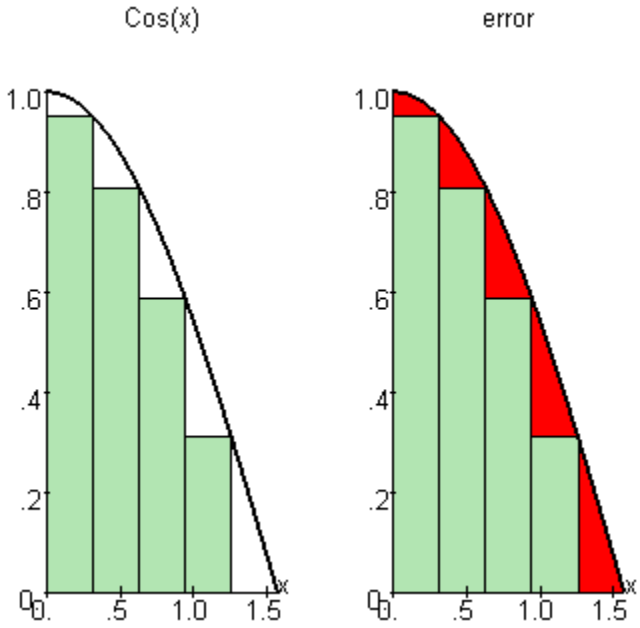
```
> restart;
  with(student):
  with(plots):
```

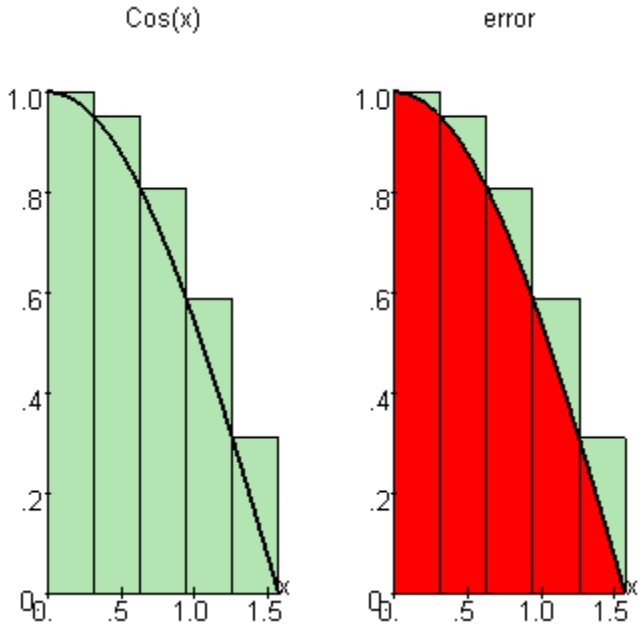
Warning, the name changecoords has been redefined

```
> riemannsum:=proc(f, xrange, n, box, incdec)
    local a, p1, p2;
    a:=array(1..2):
    if box=right then
      a[1]:=rightbox(f, xrange, n, color=black, title=`Cos(x)`,tickmarks=[4,4]):
      p1:=rightbox(f, xrange, n, color=black, title=`error`,tickmarks=[4,4]):
      p2:=plot(f, xrange, filled=true, tickmarks=[4,4]):
      if incdec=increasing then a[2]:=display(p2,p1) else a[2]:=display(p1,p2) fi:
      display(a);
```

```
  else
    if box=left then
    a[1]:=leftbox(f, xrange, n, color=black, title=`Cos(x)`,tickmarks=[4,4]):
    p1:=leftbox(f, xrange, n, color=black, title=`error`,tickmarks=[4,4]):
    p2:=plot(f, xrange, filled=true,tickmarks=[4,4]):
    if incdec=increasing then a[2]:=display(p1,p2) else a[2]:=display(p2,p1) fi:
    display(a);
    fi:
  fi:
end:
```

```
> riemannsum(cos(x), x=0..Pi/2, 5, right,decreasing);
  riemannsum(cos(x), x=0..Pi/2, 5, left,decreasing);
```





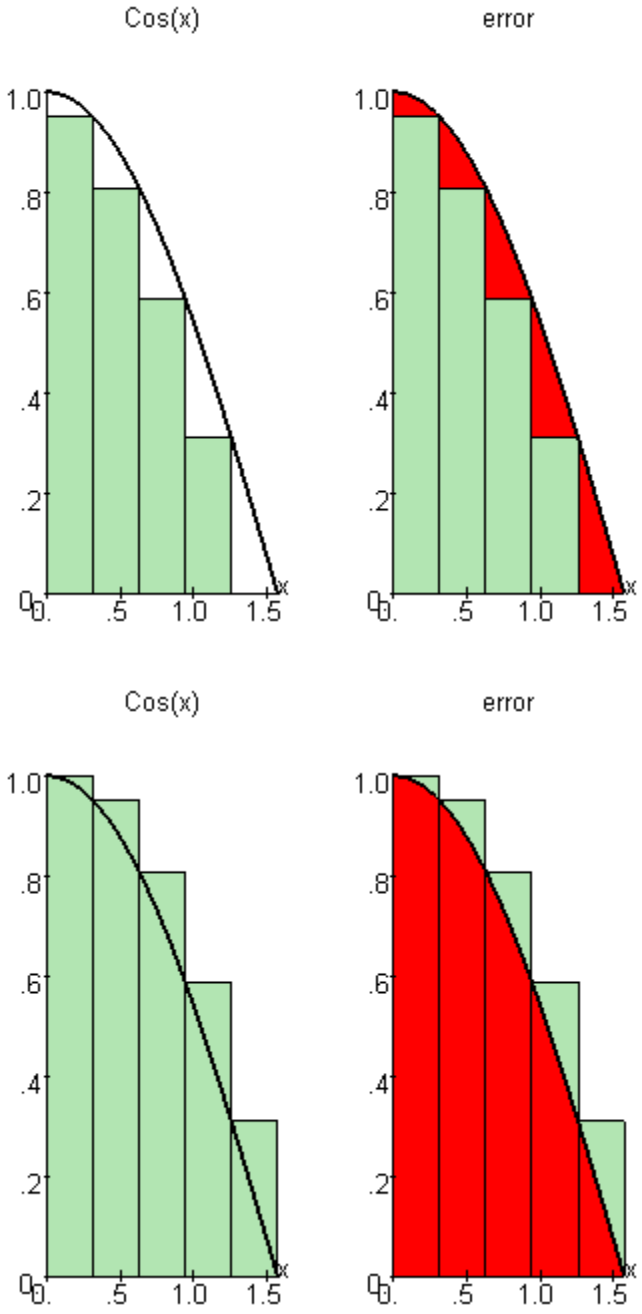
The area of each small triangular-shaped region on the error graph is the local error associated with each rectangle in the Riemann sum. The sum of the areas of these triangular-shaped regions is the global or total error in the estimate of the integral.

In general, the areas of the rectangles can be positive, negative, or 0. Because these areas are signed, we will refer to them hereafter as "signed areas."

You Try It: Visualizing the Errors

Use the **riemansumm( )** command in the preceding section (copied below) to respond to the items that follow.

```
> n:=5:
  f:=x->cos(x):
  a:=0:
  b:=Pi/2:
  incdec:=decreasing:
  riemannsum(f(x), x=a..b, n, right,incdec);
  riemannsum(f(x), x=a..b, n, left,incdec);
```



a) Increase the number of rectangles in the preceding input cell, and describe qualitatively what happens to the error in the estimate of the integral each time you double the number of rectangles.

b) What effect does the slope of the function have on the error?

c) The function $\cos(x)$ decreases on the interval from 0 to $\frac{\pi}{2}$, and its first derivative is negative. In this case, the right-hand Riemann sum underestimates the integral (i.e., the error is negative), and the left-hand sum overestimates the integral (i.e., the error is positive). Describe what happens to the error when we use left- and right-hand Riemann sums to estimate

$$\int_0^{\frac{\pi}{2}} \sin(x)\,dx \quad .$$

d) While the error for each rectangle (i.e., the local error) on the left-hand sum appears to be nearly equal in magnitude to the local error for each corresponding rectangle on the right-hand sum, they aren't exactly the same because the graph of the function curves. Where does the difference in the local errors appear to be largest, and how is the difference in the local errors related to the second derivative of the function? (To see this effect more dramatically, try using two and/or three rectangles in the **riemannsum( )** command.)

e) Based on your observations in part (d), specify two different ways in which you could use Riemann sums to improve the estimate of the integral by reducing the errors. The trick is to look for ways to estimate the integrals so that the local errors nearly cancel. For your improved methods, which feature of the integrand function would you expect to have the most significant effect on the error?

f) Try the **riemannsum( )** command on a function that we can't integrate exactly. Two examples are $\int_1^2 \frac{1}{x}\,dx$ and $\int_0^3 \mathbf{e}^{(-x^2)}\,dx$.

Part II: Analyzing the Errors

Now that we have made some qualitative observations about Riemann estimates of an integral and the associated errors, we are ready to be more quantitative in our analysis. For this we use the following commands from the **student** package, **leftsum(f, range, n)** and **rightsum(f, range, n),** to calculate left-hand and right-hand Riemann sums. The arguments are **f**, the integrand function, **range,** the range of the function to be integrated and **n**, the number of subintervals.

We use these commands, together with the fact that the exact value of the integral $\int_0^{\frac{\pi}{2}} \cos(x)\,dx$ is 1, to build a table of values that includes the step size **h**, the numerical

estimates of the integral, **RleftI** and **RrightI**, and the errors in the estimates, *Eleft = RelftI-1 and Eright = RrightI-1*. We start with two rectangles and double the number of rectangles ten times.

```
> f:=cos(x):
  a:=0:
```

```
b:=Pi/2.:
exactvalue:=int(f, x=a..b):
print(`The exact value of the integral`, exactvalue);
```

The exact value of the integral, 1.

```
> matrix([[n, h, left_est, right_est, left_error, right_error],
  seq( [2^i, Pi/2/2^i, evalf(leftsum(f, x=a..b, 2^i)), evalf(rightsum(f, x=a..b,2^i)),  evalf(leftsu
x=a..b, 2^i))-exactvalue, evalf(rightsum(f, x=a..b, 2^i))-exactvalue], i=1..11 )]);
```

5	h	*left_est*	*right_est*	*left_error*	*right_error*
2	$\frac{\pi}{4}$	1.340758531	0.5553603672	0.340758531	-0.4446396328
4	$\frac{\pi}{8}$	1.183465342	0.7907662602	0.183465342	-0.2092337398
8	$\frac{\pi}{16}$	1.094959942	0.8986104019	0.094959942	-0.1013895981
16	$\frac{\pi}{32}$	1.048284066	0.9501092953	0.048284066	-0.0498907047
32	$\frac{\pi}{64}$	1.024342887	0.9752555020	0.024342887	-0.0247444980
64	$\frac{\pi}{128}$	1.012221646	0.9876779539	0.012221646	-0.0123220461
128	$\frac{\pi}{256}$	1.006123373	0.9938515270	0.006123373	-0.0061484730
256	$\frac{\pi}{512}$	1.003064824	0.9969289012	0.003064824	-0.0030710988
512	$\frac{\pi}{1024}$	1.001533196	0.9984652354	0.001533196	-0.0015347646
1024	$\frac{\pi}{2048}$	1.000766794	0.9992328135	0.000766794	-0.0007671865
2048	$\frac{\pi}{4096}$	1.000383446	0.9996164561	0.000383446	-0.0003835439

The data in the table should confirm your qualitative observations from the "You Try It: Visualizing the Errors" above, but now we can be more specific. We make the following quantitative observations.

1) As the number of rectangles increases, the error decreases, and, in fact, each time we double the number of rectangles or cut h in half, we cut the error roughly in half. A numerical estimate that exhibits this characteristic is said to have an error of order h, and we designate this as $O(h)$.

2) The left-hand sum overestimates the integral, whereas the right-hand some underestimates it, and, for the same number of rectangles, the errors are roughly equal in magnitude.

3) When the number of rectangles is small, the difference between the errors for the left- and right-hand estimates is larger, showing the effect of the second derivative and the concavity of the graph on the difference in errors.

The one effect that is not evident from the numeric data is that the local error is larger when the magnitude of the first derivative, that is, the magnitude of the slope of the graph, is larger. We explore this effect further in Part III below.

You Try It: On Another Function

Perform a numerical experiment like the one in Part II on the integral $\int_0^{\frac{\pi}{2}} \sin(x)^2\,dx$, and then

answer the questions that follow. To help, we include the commands to generate the table of values like the one in Part II.

```
> f:=sin(x)^2:
  a:=0:
  b:=Pi/2.:
  exactvalue:=int(f, x=a..b):
  print(`The exact value of the integral`,exactvalue);
```

The exact value of the integral, 0.7853981634

```
> matrix([[n, h, left_est, right_est, left_err, right_err],
    seq( [2^i, Pi/2/2^i,   evalf(leftsum(f, x=a..b, 2^i)), evalf(rightsum(f, x=a..b,2^i)),   evalf(left
  (f, x=a..b,   2^i))-exactvalue, evalf(rightsum(f, x=a..b, 2^i))-exactvalue], i=1..11 )]);
```

5	h	*left_est*	*right_est*	*left_err*	*right_err*
2	$\frac{\pi}{4}$	0.3926990818	1.178097245	-0.3926990816	0.3926990816
4	$\frac{\pi}{8}$	0.5890486226	0.9817477044	-0.1963495408	0.1963495410
8	$\frac{\pi}{16}$	0.6872233931	0.8835729338	-0.0981747703	0.0981747704
16	$\frac{\pi}{32}$	0.7363107781	0.8344855488	-0.0490873853	0.0490873854
32	$\frac{\pi}{64}$	0.7608544709	0.8099418561	-0.0245436925	0.0245436927
64	$\frac{\pi}{128}$	0.7731263172	0.7976700102	-0.0122718462	0.0122718468
128	$\frac{\pi}{256}$	0.7792622402	0.7915340867	-0.0061359232	0.0061359233
256	$\frac{\pi}{512}$	0.7823302020	0.7884661250	-0.0030679614	0.0030679616
512	$\frac{\pi}{1024}$	0.7838641827	0.7869321443	-0.0015339807	0.0015339809
1024	$\frac{\pi}{2048}$	0.7846311733	0.7861651538	-0.0007669901	0.0007669904
2048	$\frac{\pi}{4096}$	0.7850146682	0.7857816587	-0.0003834952	0.0003834953

1. What happens to the error each time the number of rectangles is doubled? What is the order of the errors for each Riemann sum estimate of the integral?

2. Does the left-hand Riemann sum overestimate or underestimate the integral? What about the right-hand Riemann sum?

3. What effect does the second derivative of the integrand have on the difference of the errors for the left and right Riemann sums?

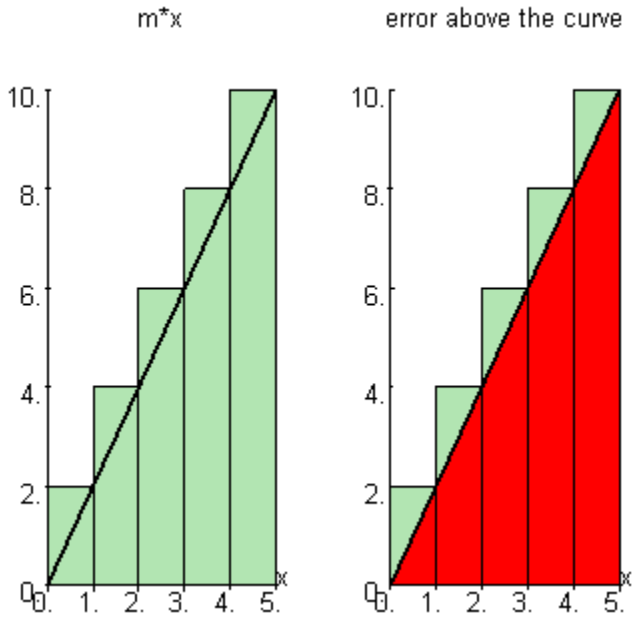
Part III: The Effect of f' on the Error

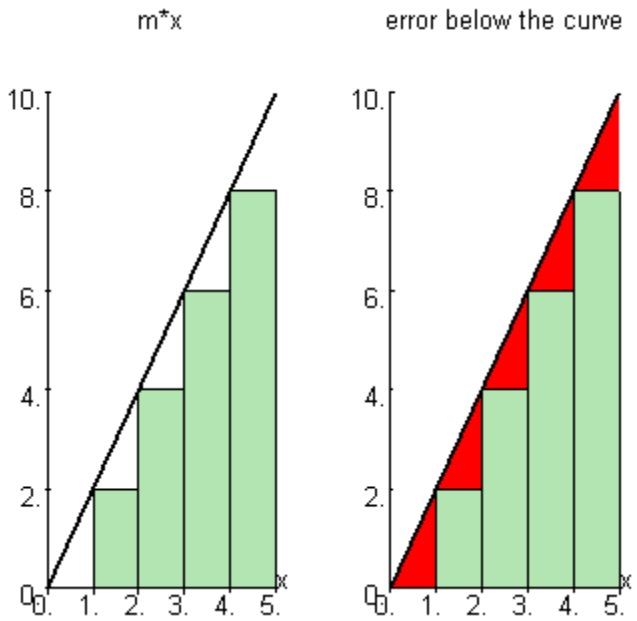
To quantify the effect of the slope on the error, we investigate straight-line functions,

$f(x) = m\,x$, with varying slopes. First, let's look at a graphical depiction.

```
> riemannsum:=proc(f, xrange, n, box)
   local a, p1, p2:
   a:=array(1..2):
    if box=right then
      a[1]:=rightbox(f, xrange, n, color=black, title=`m*x`):
      p1:=rightbox(f, xrange, n, color=black, title=`error above the curve`):
      p2:=plot(f, xrange, filled=true):
      a[2]:=display(p2,p1):
      display(a);
    else
      a[1]:=leftbox(f, xrange, n, color=black, title=`m*x`):
      p1:=leftbox(f, xrange, n, color=black, title=`error below the curve`):
      p2:=plot(f, xrange, filled=true):
      a[2]:=display(p1,p2):
      display(a);
    fi:
  end:

> n:=5:
  m:=2:
  riemannsum(m*x, x=0..5, n, right);
  riemannsum(m*x, x=0..5, n, left);
```





Now we set up a table of numeric values, but in this case we keep the number of rectangles fixed at 100 and change *m*, the slope of the straight-line graph, starting with $m = .1e\text{-}1$ and doubling ten times. The exact values of the integrals are $\frac{25\,m}{2}$.

```
> unassign('m,n,f,x'):
  m:=0.1*2^i:
  f:=m*x:
  a:=0:
  b:=5:
```

```
> exactvalue:=int(f,x=a..b):
  print(`The exact value of the integral is`, exactvalue);
  matrix([[slope, exact, left_est, right_est, left_err, right_err],
    seq( [m(i), exactvalue(i), evalf(leftsum(f, x=a..b, 100)), evalf(rightsum(f, x=a..b,100)),  eva
  (leftsum(f, x=a..b, 100))-exactvalue, evalf(rightsum(f, x=a..b, 100))-exactvalue], i=0..10 )]);
```

$$\textit{The exact value of the integral is,}\ 1.250000000\ 2.^{i}$$

slope	exact	left_est	right_est	left_err	right_err
0.1	1.250000000	1.237500000	1.262500000	-0.012500000	0.012500000
0.2	2.500000000	2.475000000	2.525000000	-0.025000000	0.025000000
0.4	5.000000000	4.950000000	5.050000000	-0.050000000	0.050000000
0.8	10.00000000	9.900000000	10.10000000	-0.100000000	0.10000000
1.6	20.00000000	19.80000000	20.20000000	-0.20000000	0.20000000
3.2	40.00000000	39.60000000	40.40000000	-0.40000000	0.40000000
6.4	80.00000000	79.20000000	80.80000000	-0.80000000	0.80000000
12.8	160.0000000	158.4000000	161.6000000	-1.6000000	1.6000000
25.6	320.0000000	316.8000000	323.2000000	-3.2000000	3.2000000
51.2	640.0000000	633.6000000	646.4000000	-6.4000000	6.4000000
102.4	1280.000000	1267.200000	1292.800000	-12.800000	12.800000

What happens to the error each time you double the slope of the line?

Note that for nonlinear functions, the effect of the slope on the error is local in that it varies from point to point on the graph. But if we double all of the slopes, we would also double the error in using a left- or right-hand Riemann sum to estimate the integral.

Note also that the differences in the magnitudes of the errors are all 0. This supports the observation that the difference in the errors is dependent upon the second derivative or the concavity of the graph. One method for improving our numerical estimate of the integral is to add the left- and right-hand estimates and then divide the result by two. The idea that motivates this is that the errors for the two estimates will nearly cancel each other out when we add the left- and right-hand sums together. For linear functions, the second derivative is 0, and the graph has no concavity; therefore, this improvement will give the exact value of the integral.

You Try It: The Effect of f'' on the Error Difference

Design a numerical experiment like the one in Part III to demonstrate the effect of the value of the second derivative on the difference between the errors in the estimate of the integral that are obtained using right- and left-hand Riemann sums. To do this, select a simple function that has a constant second derivative, is easy to integrate, and allows you to change the value of the second derivative. Generate a table of numerical values that shows the values of the second derivative and the difference in the errors for the left and right Riemann sums. Take 1 as the starting value of the second derivative, and double its value ten times. Try functions that are concave down as well as ones that are concave up, and summarize the results of your investigation. (We have put the solution for this problem in the next input section with the code hidden. Try to do it on your own, and then look at our solution.)

Solution

```
> unassign('m,n,f,x'):
  m:=2^i:
  f:=1/2*m*x^2:
  a:=0:
  b:=5:exactvalue:=int(f,x=a..b):
  print(`The exact value of the integral is`, exactvalue);
  matrix([[`f''(x)`, exact, left_est, right_est, left_err, right_err], seq( [m(i), evalf(exactva
  evalf(leftsum(f, x=a..b, 100)), evalf(rightsum(f, x=a..b,100)), evalf(leftsum(f, x=a..b, 10
  exactvalue, evalf(rightsum(f, x=a..b, 100))-exactvalue], i=0..10 )]);
```

$$\textit{The exact value of the integral is, } \frac{125\,2^{i}}{6}$$

f''(x)	exact	left_est	right_est	left_err	right_err
1	20.83333333	20.52187500	21.14687500	-0.31145833	0.31354167
2	41.66666667	41.04375000	42.29375000	-0.62291667	0.62708333
4	83.33333333	82.08750000	84.58750000	-1.24583333	1.25416667
8	166.6666667	164.1750000	169.1750000	-2.4916667	2.5083333
16	333.3333333	328.3500000	338.3500000	-4.9833333	5.0166667
32	666.6666667	656.7000000	676.7000000	-9.9666667	10.0333333
64	1333.333333	1313.400000	1353.400000	-19.933333	20.066667
128	2666.666667	2626.800000	2706.800000	-39.866667	40.133333
256	5333.333333	5253.600000	5413.600000	-79.733333	80.266667
512	10666.66667	10507.20000	10827.20000	-159.46667	160.53333
1024	21333.33333	21014.40000	21654.40000	-318.93333	321.06667

Part IV: Trapezoids and Midpoint Riemann Sums

In this part, we investigate two ways to improve the efficiency of numerical estimates of definite integrals.

Trapezoids

We achieve the first improvement by adding together the left and right Riemann sum estimates of the integral and dividing by two. The motivation for this approach is that the errors in the left and right sums should nearly cancel each other out. In the next section, we form the new function **trapezoid( )** that does this.

```
> trapezoid:=proc(f, a, b, n):
  (evalf(leftsum(f, x=a..b, n)+rightsum(f, x=a..b, n)))/2:
  end:
```

The trapezoid method gets its name from the signed area of a trapezoid in each subinterval that results from adding the signed areas of the left and right rectangles in each subinterval and dividing by 2. That is, $\frac{f(c_{left})\,h + f(c_{right})\,h}{2} = \frac{(f(c_{left}) + f(c_{right}))\,h}{2}$, the area of a trapezoid, where $f(c_{left})$ is the length of one parallel side, $f(c_{right})$ is the length of the other parallel side, and h is the distance between the two sides.

Now we use **trapezoid( )** to estimate $\int_0^{\frac{\pi}{2}} \cos(x)\,dx$ with ten subintervals and compare the result with the left- and right-hand Riemann sums. First, we calculate the exact value of the integral.

```
> unassign('f, a, b, n, exactvalue'):
  f:=cos(x):
  a:=0:
  b:=Pi/2:
  exactvalue:=int(f,x=a..b):
  print(`The exact value of the integral is `,exactvalue);
```

The exact value of the integral is , 1

Now we calculate the estimates of the integral .

```
> trapezoid(f, a, b, 10);
  evalf(leftsum(f, x=a..b, 10));
  evalf(rightsum(f, x=a..b, 10));
```

0.9979429865

1.076482803

0.9194031700

And we compute the errors.

```
> trapezoid(f, a, b, 10)-exactvalue;
  evalf(leftsum(f, x=a..b, 10))-exactvalue;
  evalf(rightsum(f, x=a..b, 10))-exactvalue;
```

$$-0.0020570135$$

$$0.076482803$$

$$-0.0805968300$$

For the same number of subintervals, we get a much more precise estimate of the integral from the trapezoid method.

Midpoint Riemann Sums

The second improvement we consider is to use the midpoint of each subinterval in a Riemann sum. The effect of doing this is to nearly cancel the error on either side of each midpoint in each subinterval. If the function we integrate is increasing over a subinterval, then the midpoint rectangle will overestimate the actual area on the left side of the midpoint and will underestimate, by nearly the same amount, the actual area on the right side of the midpoint. Again we expect that these errors will nearly cancel each other out. Here is the **riemSumMid( )** command.

```
> riemSumMid:=proc(f,a,b,n):
  evalf(sum('subs(x=(a+(i-1/2)*(b-a)/n),f)*(b-a)/n', i=1..n)):
  end:
```

Let's try it on our old friend and compare the result with left and right Riemann sums and with the trapezoid estimate. First, we calculate the exact value of the integral.

```
> unassign('f, a, b, n, exactvalue'):
  f:=cos(x):
  a:=0:
  b:=Pi/2:
  exactvalue:=evalf(int(f,x=a..b)):
  print(`The exact value of the integral is `,exactvalue);
```

$$\textit{The exact value of the integral is }, 1.$$

Now we calculate the estimates of the integral.

```
> riemSumMid(cos(x), 0, Pi/2, 10);
  trapezoid(f, a, b, 10);
  evalf(leftsum(f, x=a..b, 10));
  evalf(rightsum(f, x=a..b, 10));
```

1.001028824

0.9979429865

1.076482803

0.9194031700

Again, we compute the errors.

```
> riemSumMid(cos(x), 0, Pi/2, 10)-exactvalue;
  trapezoid(f, a, b, 10)-exactvalue;
  evalf(leftsum(f, x=a..b, 10))-exactvalue;
  evalf(rightsum(f, x=a..b, 10))-exactvalue;
```

0.001028824

-0.0020570135

0.076482803

-0.0805968300

For the same number of intervals, we get a much more precise estimate of the integral using the midpoint rule; it is even more precise than the estimate given by the trapezoid rule. What relationship does the error for the midpoint estimate seem to have with that for the trapezoid estimate?

You Try It: On Some Other Functions

Compare the Riemann left, Riemann right, trapezoid, and midpoint estimates of some integrals that you pick. How does the error for the midpoint rule compare with the error for the trapezoid rule with the same number of subdivisions? Also, try increasing the number of subintervals. What happens to the errors when you double the number of subintervals?

Here are some integrals that you might want to try:

$$\int_0^{\frac{\pi}{2}} \sin(x)\,dx \quad , \int_{-1}^{1} e^x\,dx \quad , \int_0^1 \sqrt{1-x}\,dx$$

To help you out, we provide some commands in the two cells that follow.

```
> unassign('f, a, b, n, exactvalue'):
  f:=cos(x):
  a:=0:
  b:=Pi/2:
  exactvalue:=evalf(int(f,x=a..b)):
  n:=10:
  print(`The exact value of the integral is `, exactvalue);
  print(`The midpoint estimate is`, evalf(riemSumMid(f, a, b, n)));
  print(`The trapezoid estimate is`, trapezoid(f, a, b, n));
  print(`The left Riemann estimate is`, evalf(leftsum(f, x=a..b,n)));
  print(`The right Riemann estimate is`, evalf(rightsum(f, x=a..b,n)));
```

The exact value of the integral is , 1.

The midpoint estimate is, 1.001028824

The trapezoid estimate is, 0.9979429865

The left Riemann estimate is, 1.076482803

The right Riemann estimate is, 0.9194031700

```
> print(`The midpoint error is`, riemSumMid(f, a, b, n)-exactvalue);
  print(`The trapezoid error is`, trapezoid(f, a, b, n)-exactvalue);
  print(`The left Riemann error is`, evalf(leftsum(f, x=a..b,n))-exactvalue);
  print(`The right Riemann estimate is`, evalf(rightsum(f, x=a..b,n))-exactvalue);
```

The midpoint error is, 0.001028824

The trapezoid error is, -0.0020570135

The left Riemann error is, 0.076482803

The right Riemann estimate is, -0.0805968300

Part V: Experiment with Trapezoids and Midpoints

In this part, we do some numerical experiments to determine what factors affect the errors when we use trapezoids and midpoint Riemann sums to estimate a definite integral.

The Effect of the Step Size on the Error

First, we would like to determine the effect of the step size *h* on the error. We build a table similar to the one in Part II.

```
> unassign('f, a, b, n,i,j, exactvalue'):
  f:=cos(x):
  a:=0:
  b:=Pi/2:
  exactvalue:=int(f,x=a..b):
  print(`The exact value of the integral is `, exactvalue);
  matrix([[n, h, trap_est, mid_est, trap_err, mid_err],
    seq( [2^j, Pi/2/2^j, trapezoid(f, a, b, 2^j), riemSumMid(f, a, b, 2^j),
    trapezoid(f, a, b, 2^j)-exactvalue, riemSumMid(f, a, b, 2^j)-exactvalue],
  j=1..11 )]);
```

The exact value of the integral is , 1

n	h	*trap_est*	*mid_est*	*trap_err*	*mid_err*
2	$\frac{\pi}{4}$	0.9480594490	1.026172153	-0.0519405510	0.026172153
4	$\frac{\pi}{8}$	0.9871158010	1.006454543	-0.0128841990	0.006454543
8	$\frac{\pi}{16}$	0.9967851720	1.001608189	-0.0032148280	0.001608189
16	$\frac{\pi}{32}$	0.9991966805	1.000401708	-0.0008033195	0.000401708
32	$\frac{\pi}{64}$	0.9997991945	1.000100406	-0.0002008055	0.000100406
64	$\frac{\pi}{128}$	0.9999498000	1.000025100	-0.0000502000	0.000025100
128	$\frac{\pi}{256}$	0.9999874500	1.000006276	-0.0000125500	$0.6276\ 10^{-5}$
256	$\frac{\pi}{512}$	0.9999968625	1.000001569	$-0.31375\ 10^{-5}$	$0.1569\ 10^{-5}$
512	$\frac{\pi}{1024}$	0.9999992155	1.000000394	$-0.7845\ 10^{-6}$	$0.394\ 10^{-6}$
1024	$\frac{\pi}{2048}$	0.9999998040	1.000000099	$-0.1960\ 10^{-6}$	$0.99\ 10^{-7}$
2048	$\frac{\pi}{4096}$	0.9999999510	1.000000024	$-0.490\ 10^{-7}$	$0.24\ 10^{-7}$

The Effect of f'' on the Error

Based on the observations made in Parts II, III, and IV, we conjecture that the error for these methods depend on the magnitude of the second derivative of the integrand function. We test our conjecture by comparing the error in estimating the area under the quadratic functions of the form $f(x) = \frac{a x^2}{2}$ between $x = 0$ and $x = 5$ for various values of a. The exact values of the integrals are $\int_0^5 \frac{a x^2}{2} dx = \frac{125 a}{6}$, and the second derivative of $f(x)$ is equal to a. We set up a table in which a starts at 0.1 and doubles ten times. The number of subintervals is kept constant at 100.

```
> unassign('ac,n,i,j,f,x'):
  ac:=0.1*2^j:
  f:=1/2*ac*x^2:
  a:=0:
  b:=5:
  exactvalue:=int(f,x=a..b):
  Digits:=(7);
  print(`The exact value of the integral is`, exactvalue);
  matrix([[f_2(x), exact, trap_est, mid_est, trap_err, mid_err],
    seq( [ac(j), evalf(exactvalue(j)), trapezoid(f,a,b, 100), riemSumMid(f, a,b,100),     tra
  a,b, 100)-exactvalue, riemSumMid(f,a,b,100)-exactvalue],
  j=0..10 )]);
```

$$Digits := 7$$

$$\text{The exact value of the integral is, } 2.083333333\ 2.^{j}$$

f_2(x)	exact	trap_est	mid_est	trap_err	mid_err
0.1	2.083333	2.083438	2.083281	0.000105	-0.000052
0.2	4.166666	4.166875	4.166562	0.000209	-0.000104
0.4	8.333332	8.333750	8.333125	0.000418	-0.000207
0.8	16.66666	16.66750	16.66625	0.00084	-0.00041
1.6	33.33333	33.33500	33.33250	0.00167	-0.00083
3.2	66.66666	66.67000	66.66500	0.00334	-0.00166
6.4	133.3333	133.3400	133.3300	0.0067	-0.0033
12.8	266.6666	266.6800	266.6600	0.0134	-0.0066
25.6	533.3332	533.3600	533.3200	0.0268	-0.0132
51.2	1066.666	1066.720	1066.640	0.054	-0.026
102.4	2133.333	2133.440	2133.280	0.107	-0.053

In "You Try It" below, we ask you to summarize the results of these experiments.

You Try It: Summarize the Experiment's Results

Summarize the results of the numerical experiments in Part V. In your summary, address the following items.

a) How is the error for each method related to the number of subintervals n and the interval width h? What is the order of the error for each method? (Note: If cutting the interval width in half reduces the error by a factor of $\left(\frac{1}{2}\right)^n$, then the error is of order h^n, and we designate this by $O(h^n)$.)

b) How is the error for the trapezoid method related to the error for the midpoint Riemann sum?

c) How is the error related to the second derivative of the integrand function? (To answer this completely, repeat the second experiment in Part V for negative values of a.)

d) Are your conclusions consistent with the trapezoid error formula that is found in the text?

e) From the pattern of errors for the trapezoid method and the midpoint Riemann sum method, specify how you might combine these two estimates of the integral so that the errors will nearly cancel each other. Compare your idea for combining the estimates with those of other students.

Part VI: Simpson's Method

Noting that the magnitude of the error for the trapezoid method is approximately two times the magnitude of the error for a midpoint Riemann sum with the same number of subintervals, we can make yet another improvement in the efficiency of our numerical estimations of the integrals. If we add two midpoint estimates and one trapezoid estimate and divide the result by three, the errors for the two methods should nearly cancel. The result of this combination is called Simpson's method.

To illustrate this, consider the following estimates of the integral $\int_0^{\frac{\pi}{2}} \cos(x)\,dx$. We compare

the midpoint and trapezoid estimates and their errors, and then we combine them as indicated above and calculate the error for the new estimate.

```
> unassign('f, a, b, n, i,j, exactvalue'):
  f:=cos(x):
  a:=0:
  b:=Pi/2:
  n:=10:
  exactvalue:=evalf(int(f,x=a..b)):

  trap:=trapezoid(f,a,b,n):
  mid:=riemSumMid(f,a,b,n):

  simps:=(trap+2*mid)/3:

  print(`The exact value of the integral is `, exactvalue);
  print(`The trapezoid estimate is`, trap);
  print(`The trapezoid error is`, trap-exactvalue);
  print(`The midpoint estimate is`, mid);
  print(`The midpoint error is`, mid-exactvalue);
  print(`The Simpson estimate is`, simps);
  print(`The error for Simpson's estimate is`, simps-exactvalue);
```

The exact value of the integral is , 1.

The trapezoid estimate is, 0.9979430

The trapezoid error is, -0.0020570

The midpoint estimate is, 1.001029

The midpoint error is, 0.001029

The Simpson estimate is, 1.000000

The error for Simpson's estimate is, 0.

Since Simpson's method requires three function evaluations in each subinterval, one at each end point for the trapezoid plus one at the mid point, we count the number of subintervals differently. If we take the dividing points for the subintervals to be all of the places where the integrand function is evaluated, then there are actually twice as many subintervals for Simpson's method than there would be for the trapezoid method or the midpoint Riemann sum. One consequence of this is that we need an even number of subintervals for Simpson's method to work.

We use *Maple's* built-in **simpson( )** function to estimate the definite integral.

Let's use it to estimate $\int_0^{\frac{\pi}{2}} cos(x)\,dx$ and then calculate the errors, comparing the results with

those obtained using trapezoids and midpoint Riemann sums.

```
> unassign('f, a, b, n, i,j, exactvalue'):
  f:=cos(x):
  a:=0:
  b:=Pi/2:
  n:=6:
  exactvalue:=evalf(int(f,x=a..b)):
  print(`The exact value of the integral is `, exactvalue);
  print(`The Simpson estimate of the integral is`, evalf(simpson(f,x=a..b,n)));
  print(`The trapezoid estimate of the integral is`, trap);
  print(`The midpoint estimate is`, riemSumMid(f,a,b,n));
```

The exact value of the integral is , 1.

The Simpson estimate of the integral is, 1.000027

The trapezoid estimate of the integral is, 0.9979430

The midpoint estimate is, 1.002861

Now we compare the errors.

```
> print(`The error in the Simpson estimate is`, evalf(simpson(f,x=a..b,n))-exactvalue);
  print(`The error in the trapezoid estimate is`, trapezoid(f,a,b,n)-exactvalue);
  print(`The error in the midpoint is`, riemSumMid(f,a,b,n)-exactvalue);
```

The error in the Simpson estimate is, 0.000027

The error in the trapezoid estimate is, -0.0057180

The error in the midpoint is, 0.002861

You Try It: Experiment with Simpson

Perform some numerical experiments like those in the preceding parts to show that the order of the error for Simpson's method is fourth. Also show that the local error is proportional to the fourth derivative of the integrand function. (We have put the solutions for this problem in the next two sections. Try to do it on your own, and then look at our solutions.)

Solution 1

```
> unassign('f, a, b, n, i,j, exactvalue'):
  f:=cos(x):
  a:=0:
  b:=Pi/2:
  exactvalue:=evalf(int(f,x=a..b)):
  Digits:=(20):
  print(`The exact value of the integral is `, exactvalue);
  matrix([[n, h, Simpson_estimate, Simpson_error],
    seq( [2^j, Pi/2/2^j, evalf(simpson(f,x=a..b, 2^j)),     evalf(simpson(f,x=a..b,2^j))-exac
  j=1..11 )]);
```

The exact value of the integral is , 1.

n	h	*Simpson_estimate*	*Simpson_error*
2	$\frac{\pi}{4}$	1.0022798774922104777	0.0022798774922104777
4	$\frac{\pi}{8}$	1.0001345849741939045	0.0001345849741939045
8	$\frac{\pi}{16}$	1.0000082955239677588	$0.82955239677588\ 10^{-5}$
16	$\frac{\pi}{32}$	1.0000005166847065004	$0.5166847065004\ 10^{-6}$
32	$\frac{\pi}{64}$	1.0000000322650009616	$0.322650009616\ 10^{-7}$
64	$\frac{\pi}{128}$	1.0000000020161287035	$0.20161287035\ 10^{-8}$
128	$\frac{\pi}{256}$	1.0000000001260012666	$0.1260012666\ 10^{-9}$
256	$\frac{\pi}{512}$	1.0000000000078749733	$0.78749733\ 10^{-11}$
512	$\frac{\pi}{1024}$	1.0000000000004921842	$0.4921842\ 10^{-12}$
1024	$\frac{\pi}{2048}$	1.0000000000000307615	$0.307615\ 10^{-13}$
2048	$\frac{\pi}{4096}$	1.0000000000000019226	$0.19226\ 10^{-14}$

Solution 2

```
> unassign('f, a, b, n, i,j, exactvalue'):
  m:=0.1*2^j:
  f:=m/24*x^4:
  a:=0:
  b:=5:
  exactvalue:=int(f,x=a..b):
  Digits:=(12):
  print(`The exact value of the integral is `, exactvalue);
  matrix([[f_4, h, exact, Simpson_estimate, Simpson_error],
    seq( [m, Pi/2/2^j, exactvalue, evalf(simpson(f,x=a..b, 100)),    evalf(simpson(f,x=a..b
  exactvalue)],
  j=0..10 )]);
```

The exact value of the integral is , $2.6041666666666667\ 2.^{j}$

f_4	h	*exact*	*Simpson_estimate*	*Simpson_error*
0.1	$\frac{\pi}{2}$	2.60416666667	2.60416668403	$0.17363\ 10^{-7}$
0.2	$\frac{\pi}{4}$	5.20833333334	5.20833336807	$0.3473\ 10^{-7}$
0.4	$\frac{\pi}{8}$	10.4166666667	10.4166667361	$0.6941\ 10^{-7}$
0.8	$\frac{\pi}{16}$	20.8333333334	20.8333334722	$0.13876\ 10^{-6}$
1.6	$\frac{\pi}{32}$	41.6666666667	41.6666669444	$0.2777\ 10^{-6}$
3.2	$\frac{\pi}{64}$	83.3333333334	83.3333338887	$0.5553\ 10^{-6}$
6.4	$\frac{\pi}{128}$	166.666666667	166.666667778	$0.11113\ 10^{-5}$
12.8	$\frac{\pi}{256}$	333.333333334	333.333335556	$0.2222\ 10^{-5}$
25.6	$\frac{\pi}{512}$	666.666666668	666.666671113	$0.4445\ 10^{-5}$
51.2	$\frac{\pi}{1024}$	1333.33333334	1333.33334222	$0.8880\ 10^{-5}$
102.4	$\frac{\pi}{2048}$	2666.66666667	2666.66668444	0.000017772

>