Double Integrals using Riemann Sums

Introduction and Goals:

The goal of this lab is to become more familiar with Riemann sums, both as a definition for the double integral and as an approximation method for the double integral. This lab also explores Maple more deeply by creating several numeric and graphical commands that calculate and display Riemann sums.

Before You Start:

Make sure that you read and understand the mathematics from the corresponding sections in your textbook.

Textbook Correspondence:


Maple Commands and Packages Used:

Packages: plots and plottools.
Commands: plot3d, evalf, display, proc, sum, limit, lhs, rhs and subs.
User-Defined Commands: Riemann3dLL, plotRS3dLL, LRiemann3d and plotLRS3d.

History & Biographies:

Maple Commands:

As you know from Calculus I and II the definite integral of a function is defined as the limit of Riemann sums as the width of the rectangles (or the mesh) goes to zero. Equivalently, as the number of rectangles goes to infinity, assuming that the width of the base of the rectangles are all the same. You also know from your reading that we define the double integral over a rectangle $R$ by

$$\int\int_R f(x, y)dA = \lim_{m,n\to\infty} \sum_{i=1}^m \sum_{j=1}^n f(x^*_{ij}, y^*_{ij})\Delta A$$

where $\left(x^*_{ij}, y^*_{ij}\right)$ is a sample point from the $ij^{th}$ subrectangle of the region $R$. In other words, it is the two variable version of the Riemann sum. For this lab we are not
interested in using this definition to calculate the double integral but instead to use the Riemann sum to approximate the double integral. We are also interested in producing graphical representations of these Riemann sum approximations. This gives us an opportunity to delve a little more deeply into the Maple functions.

We will start with calculating Riemann sums. We will take a specific example and then generalize it into the creation of a new Maple command that will take in a few parameters and output a numeric value for the Riemann sum approximation. Consider the function, \( f(x, y) = x^2 + y^2 + 1 \). We will approximate the double integral of this function over the rectangle \([0, 1] \times [0, 1]\). The actual value of the double integral is \( \frac{\pi}{3} \) and is easy to calculate using methods we will see shortly, but not here. To approximate this double integral we need to calculate

\[
\sum_{i=1}^{m} \sum_{j=1}^{n} f(x_{ij}^*, y_{ij}^*) \Delta A
\]

where \( m \) and \( n \) are positive integers, the region \([0, 1] \times [0, 1]\) is divided into \( m \) equal segments in the \( x \) direction and \( n \) equal segments in the \( y \) direction, and the point \((x_{ij}^*, y_{ij}^*)\) is some sample point in the \( ij^{th} \) subrectangle of the region. For this example we will choose \( n = 5, m = 7 \) and the point \((x_{ij}^*, y_{ij}^*)\) will be in the lower left corner of the subrectangle. It will help if you draw a detailed graph of the region, its subrectangles and the values of each of the divisions. Since the \( x \) and \( y \) ranges are both \([0, 1]\) the divisions for the \( x \) direction are \([0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, \frac{5}{5}]\) and the divisions for the \( y \) direction are \([0, \frac{1}{7}, \frac{2}{7}, \frac{3}{7}, \frac{4}{7}, \frac{5}{7}, \frac{6}{7}, \frac{7}{7}]\). Also, \( \Delta A = \frac{1}{35} \cdot \frac{1}{35} = \frac{1}{1225} \) and since the point \((x_{ij}^*, y_{ij}^*)\) will be in the lower left corner of the subrectangle we are interested in the \( x \) direction values of \([0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, \frac{5}{5}]\) and the \( y \) direction values of \([0, \frac{1}{7}, \frac{2}{7}, \frac{3}{7}, \frac{4}{7}, \frac{5}{7}, \frac{6}{7}, \frac{7}{7}]\). Mathematically, the Riemann sum is

\[
\frac{1}{35} \sum_{i=0}^{4} \sum_{j=0}^{6} f\left(\frac{i}{5}, \frac{j}{7}\right)
\]

Let’s set this up in Maple. First define your function,

\[
f := (x, y) \rightarrow x^2 + y^2 + 1;
\]

Now define and execute the double sum.

\[
\frac{1}{35} \sum_{i=0}^{4} \sum_{j=0}^{6} f\left(\frac{i}{5}, \frac{j}{7}\right);
\]

\[
\frac{1}{35} \sum_{i=0}^{4} \sum_{j=0}^{6} f\left(\frac{i}{5}, \frac{j}{7}\right); \quad 184441825
\]

\[
\text{evalf}\left(\frac{1}{35} \sum_{i=0}^{4} \sum_{j=0}^{6} f\left(\frac{i}{5}, \frac{j}{7}\right)\right); \quad 1.505306122
\]
so our approximation for the double integral is 1.505306122. We can, of course, get better approximations of the integral by increasing the number of subrectangles over the region. For example, if we used 100 subintervals in each direction (10000 rectangles in all) we would obtain,

\[
\frac{1}{10000} \sum \left( \sum f\left(\frac{i}{100}, \frac{j}{100}\right), i=0..99, j=0..99 \right);
\]

\[
\frac{16567}{10000}
\]

\[
\text{evalf}\left(\frac{1}{10000} \sum \left( \sum f\left(\frac{i}{100}, \frac{j}{100}\right), i=0..99, j=0..99 \right)\right);
\]

\[
1.656700000
\]

If we used 1000 subintervals in each direction (1,000,000 rectangles in all) we would obtain,

\[
\frac{1}{1000000} \sum \left( \sum f\left(\frac{i}{1000}, \frac{j}{1000}\right), i=0..999, j=0..999 \right);
\]

\[
\frac{1665667}{1000000}
\]

\[
\text{evalf}\left(\frac{1}{1000000} \sum \left( \sum f\left(\frac{i}{1000}, \frac{j}{1000}\right), i=0..999, j=0..999 \right)\right);
\]

\[
1.665667000
\]

As you can see, these approximations are getting closer and closer to the actual value of \(\frac{5}{3}\). This example is easy enough to calculate the exact value of the double integral using the limit of the Riemann sums. We will not do this with all of the examples in the lab since Maple, and us for that matter, could not handle them. To calculate the actual value of this double integral we need to make the number of \(x\) and \(y\) divisions arbitrary and then take two limits, as the number of divisions increases without bound. Making the number of \(x\) and \(y\) divisions arbitrary is fairly easy we simply replace the numbers with an \(m\) and \(n\), as below,

\[
\frac{1}{mn} \sum \left( \sum f\left(\frac{i}{m}, \frac{j}{n}\right), i=0..m-1, j=0..n-1 \right);
\]

\[
\frac{5nm}{3} - \frac{n}{2} + \frac{n}{6} - \frac{m}{2} + \frac{m}{6n}
\]

\[
\frac{5}{3}
\]

Notice that we actually get a closed form for the Riemann sum, which seldom happens. Now all that remains is to take the limit of this expression as both \(m\) and \(n\) increase without bound.

\[
\text{limit}\left(\text{limit}\left(\frac{1}{mn} \sum \left( \sum f\left(\frac{i}{m}, \frac{j}{n}\right), i=0..m-1, j=0..n-1 \right), n=\text{infinity}\right), m=\text{infinity}\right);
\]

\[
\frac{5}{3}
\]
Walla, we have the exact value of the double integral. Before we generalize this to a new command let’s look at another example. We will consider the same function but this time on the region \( c \). This will clearly be similar to the last Riemann sum we set up but there will be some differences. The question is, what are the differences? Since we are changing the region that we are integrating over the differences will be in any part of the sum that involves the region. This would include the way we select our sample points and the calculation of \( \Delta A \). The rest of the sum will be unaltered. For \( \Delta A \) all we need to do is multiply by \( \frac{m\Delta x}{n} \) and \( \frac{m\Delta y}{n} \) where \( R_x \) and \( R_y \) are the lengths of the \( x \) and \( y \) ranges respectively. As for \( \left( x^*_i, y^*_j \right) \) we need to divide the ranges \([2,5]\) and \([-3,1]\) into \( m \) and \( n \) equal segments. One easy way to do this is to start at the left endpoint of each range and add a multiple of the range. That is, for the \( i^{th} \) division in the \( x \) direction we can use \( 2 + 3\frac{i}{m} \) and for the \( j^{th} \) division in the \( y \) direction we can use \( -3 + 4\frac{j}{n} \). Again, we will choose \( n = 5 \), \( m = 7 \) and the point \( \left( x^*_i, y^*_j \right) \) will still be in the lower left corner of the subrectangle. The Maple command that finds the Riemann sum is

\[
\sum\sum f(2+3i/5,-3+4j/7) i=0..4, j=0..6;
\]

\[
\frac{219312}{1225}
\]

\[
\text{evalf}(3/5*4/7*\sum\sum f(2+3i/5,-3+4j/7), i=0..4, j=0..6));
\]

\[
179.0302041
\]

As before we can easily change the number of divisions to obtain better and better approximations to the double integral.

\[
\text{evalf}(3/100*4/100*\sum\sum f(2+3i/100,-3+4j/100), i=0..99, j=0..99));
\]

\[
195.2250000
\]

\[
\text{evalf}(3/10000*4/10000*\sum\sum f(2+3i/10000,-3+4j/10000), i=0..9999, j=0..9999));
\]

\[
195.9922005
\]

\[
\text{evalf}(3/100000000*4/100000000*\sum\sum f(2+3i/100000000,-3+4j/100000000), i=0..99999999, j=0..99999999));
\]

\[
195.9999992
\]

These values seem to be approaching the value of 196. By taking the limit as \( m \) and \( n \) increase without bound we see that this is the actual value.

\[
\sum\sum f(2+3i/m,-3+4j/n), i=0..m-1, j=0..n-1);
\]
\[
\frac{12}{m n} \left( \frac{49 n m}{3} + 4 m + \frac{8 m}{3 n} - \frac{21 n}{2} + \frac{3 n}{2 m} \right)
\]

> \text{limit(limit(3/m*4/n*sum(sum(f(2+3*i/m,-3+4*j/n),i=0..m-1),j=0..n-1),n=infinity),m=infinity);} \\
196

You may be wondering why we are spending so much time discussing approximate solutions when we can simply put a couple limits on the general expression and obtain the exact solution? Well, the fact is that these exact solutions are attainable only for relatively nice functions. If we tried a function that is a little more complex the limits are not easy, or possible, to do. For example consider the function \( g(x, y) = e^{x+y/2} \) on the region \([-2,2] \times [-2,2]\]. Using 10,000 subrectangles we get an approximation of 23.23976922.

> \text{evalf(4/100*4/100*sum(sum(evalf(g(-2+4*i/100,-2+4*j/100)),i=0..99),j=0..99));} \\
23.23976922

Replacing the 10,000 subrectangles with \( mn \) subrectangles we see that Maple is no longer able to provide a closed solution to the Riemann sum, it simply spits back what we gave it.

> \text{4/m*4/n*sum(sum(g(-2+4*i/m,-2+4*j/n),i=0..m-1),j=0..n-1);} \\
16 \left( \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} e^{\frac{1}{m}} \left( \frac{4 i}{m} \right)^2 - \frac{4 j}{m} \right)^{1/2} \left( \frac{4 i}{m} \right)^2 + \frac{4 j}{m} + 1) \right) \right) \right) m n

Blindly trying to find the limit of this expression results in a long wait until you get fed up with it and click the Stop button.

> \text{limit(limit(4/m*4/n*sum(sum(g(-2+4*i/m,-2+4*j/n),i=0..m-1),j=0..n-1),n=infinity),m=infinity);} \\

Now we will construct a new command that will calculate the Riemann sums for us given the function, bounds and divisions. Since we have been working with our sample points being in the lower left corner of the subrectangles we will construct the command \text{Riemann3dLL} that will find the Riemann sum using the lower left corner. We would like to give the command the function, ranges for both the variables and the number of divisions of each of the ranges. So the input and output might look like the following.
We also want to be sure that the choice of variables is arbitrary, we don’t want to be stuck always using an $x$ and $y$.

If we look at one of the specific sums we did above we can generalize the numbers to a variable.

We would like the general syntax of the Riemann3dLL command to be

where $f$ is the function, $xrng$ is the range of the first variable, $yrng$ is the range of the second variable, $xdiv$ is the number of divisions to use in the $x$ direction and $ydiv$ is the number of divisions to use in the $y$ direction. From the specific example above we see that we will need the left endpoint to each range, we will need to evaluate the function at a specific point, we need the numeric value of the range in each direction. The rest of the values needed are in the parameters. This brings up the question of how do we extract the left and right bounds from a range that is input as $x = 2..5$? The base data structure for Maple is the list. That is, everything in Maple is stored in some type of list. Ranges are no exception. Since the range is something like an equation Maple provides two commands for selecting portions of the equation, lhs and rhs. The lhs command extracts the left-hand side of the equation and rhs extracts the right-hand side of the equation. For example,

Using lhs and rhs together we can extract the bounds on the range.
> \text{lhs}(\text{rhs}(\text{xrng}))
\>
\text{2}

> \text{rhs}(\text{rhs}(\text{xrng}))
\>
\text{5}

So the

\[ \frac{3}{5} \times \frac{4}{7} \]

calculation of $\Delta A$ will generalize to

\[ \frac{\text{rhs}(\text{rhs}(\text{xrng})) - \text{lhs}(\text{rhs}(\text{xrng}))}{\text{xdiv}} \times \frac{\text{rhs}(\text{rhs}(\text{yrng})) - \text{lhs}(\text{rhs}(\text{yrng}))}{\text{ydiv}} \]

The

\[ 2 + \frac{3i}{5} \]

calculation of the $x$ coordinate of the lower left corner will generalize to

\[ \text{lhs}(\text{rhs}(\text{xrng})) + \frac{\text{rhs}(\text{rhs}(\text{xrng})) - \text{lhs}(\text{rhs}(\text{xrng}))}{\text{xdiv}} \times i \]

and the

\[ -3 + \frac{4j}{7} \]

calculation of the $y$ coordinate of the lower left corner will generalize to

\[ \text{lhs}(\text{rhs}(\text{yrng})) + \frac{\text{rhs}(\text{rhs}(\text{yrng})) - \text{lhs}(\text{rhs}(\text{yrng}))}{\text{ydiv}} \times j \]

to evaluate the function at these values we have several choices of commands but we will use the \text{subs} command. In order to make the \text{Riemann3dLL} command run for any variable we need to place

\[ \text{lhs}(\text{xrng}) = \]

and

\[ \text{lhs}(\text{yrng}) = \]

before the expression for the position. So the entire evaluation expression will be

\text{subs}(\text{lhs}(\text{xrng}) = \text{lhs}(\text{rhs}(\text{xrng})) + \frac{\text{rhs}(\text{rhs}(\text{xrng})) - \text{lhs}(\text{rhs}(\text{xrng}))}{\text{xdiv}} \times i, \text{lhs}(\text{yrng}) = \text{lhs}(\text{rhs}(\text{yrng})) + \frac{\text{rhs}(\text{rhs}(\text{yrng})) - \text{lhs}(\text{rhs}(\text{yrng}))}{\text{ydiv}} \times j, f)
The only thing remaining is the ranges for the $i$ and $j$ in the evaluation expression, and then putting it all together. The ranges are fairly easy they simply go from 0 to $x\text{div}-1$ and $y\text{div}-1$.

\[
\begin{align*}
  i &= 0 .. x\text{div}-1 \\
  j &= 0 .. y\text{div}-1
\end{align*}
\]

Putting it all together with a couple evalf’s our new command looks like the following.

\[
\begin{align*}
  \text{Riemann3dLL:} &= (f, \text{xrng}, \text{yrng}, x\text{div}, y\text{div}) \rightarrow \text{evalf}(
    (\text{rhs}(\text{rhs}(\text{yrng})) - \text{lhs}(\text{rhs}(\text{yrng}))) / y\text{div} * (\text{rhs}(\text{rhs}(\text{xrng})) - \\
    \text{lhs}(\text{rhs}(\text{xrng}))) / x\text{div} * \text{sum} \left( \text{sum}(\text{evalf}(\text{subs}(\text{lhs}(\text{xrng}) = \text{lhs}(\text{rhs}(\text{xrng})) + (\text{rhs}(\text{rhs}(\text{xrng})) - \text{lhs}(\text{rhs}(\text{xrng}))) * i / x\text{div}, \\
    \text{lhs}(\text{yrng}) = \text{lhs}(\text{rhs}(\text{yrng})) + (\text{rhs}(\text{rhs}(\text{yrng})) - \\
    \text{lhs}(\text{rhs}(\text{yrng}))) * j / y\text{div}, f)), i = 0 .. x\text{div}-1, j = 0 .. y\text{div}-1))
  \end{align*}
\]

For the computer scientists in the room this code does look a bit ugly. Although we are not going to get into a discussion on procs (Maple procedures), here are two ways to write the same command in a better way.

\[
\begin{align*}
  \text{Riemann3dLL:} &= \text{proc}(f, \text{xrng}, \text{yrng}, x\text{div}, y\text{div}) \\
  &\quad \text{local} \ i, j, \text{uxb}, \text{lxb}, \text{uyb}, \text{lyb}, \text{xdist}, \text{ydist}, \text{xc}, \text{yc}, \text{hsum}; \\
  &\quad \text{uxb} := \text{rhs}(\text{rhs}(\text{xrng})); \\
  &\quad \text{lxb} := \text{lhs}(\text{rhs}(\text{xrng})); \\
  &\quad \text{uyb} := \text{rhs}(\text{rhs}(\text{yrng})); \\
  &\quad \text{lyb} := \text{lhs}(\text{rhs}(\text{yrng})); \\
  &\quad \text{xdist} := \text{uxb} - \text{lxb}; \\
  &\quad \text{ydist} := \text{uyb} - \text{lyb}; \\
  &\quad \text{xc} := \text{lhs}(\text{xrng}); \\
  &\quad \text{yc} := \text{lhs}(\text{yrng}); \\
  &\quad \text{hsum} := \text{sum}(\text{sum}(\text{evalf}(\text{subs}(\text{xc} = \text{lxb} + \text{xdist} * i / x\text{div}, \\
    \text{yc} = \text{lyb} + \text{ydist} * j / y\text{div}, f)), i = 0 .. x\text{div}-1, j = 0 .. y\text{div}-1)); \\
  &\quad \text{xdist} / x\text{div} * \text{ydist} / y\text{div} * \text{hsum}; \\
  &\quad \text{end proc;}
\end{align*}
\]

\[
\begin{align*}
  \text{Riemann3dLL:} &= \text{proc}(f, \text{xrng}, \text{yrng}, x\text{div}, y\text{div}) \\
  &\quad \text{local} \ i, j, \text{uxb}, \text{lxb}, \text{uyb}, \text{lyb}, \text{xdist}, \text{ydist}, \text{xc}, \text{yc}, \text{hsum}, \\
  &\quad \text{xpos}, \text{ypos}; \\
  &\quad \text{uxb} := \text{rhs}(\text{rhs}(\text{xrng})); \\
  &\quad \text{lxb} := \text{lhs}(\text{rhs}(\text{xrng})); \\
  &\quad \text{uyb} := \text{rhs}(\text{rhs}(\text{yrng})); \\
  &\quad \text{lyb} := \text{lhs}(\text{rhs}(\text{yrng})); \\
  &\quad \text{xdist} := \text{uxb} - \text{lxb}; \\
  &\quad \text{ydist} := \text{uxb} - \text{lxb}; \\
  &\quad \text{xc} := \text{lhs}(\text{xrng});
\end{align*}
\]
\begin{verbatim}
> yc:=lhs(yrng);
> hsum:=0;
> for i from 0 to xdiv-1 do
>   xpos:=lxb+xdist*i/xdiv;
>   for j from 0 to ydiv-1 do
>     ypos:=lyb+ydist*j/ydiv;
>     hsum:=hsum+evalf(subs(xc=xpos,yc=ypos,f));
>   end do;
> end do;
> xdist/xdiv*ydist/ydiv*hsum;
> end proc:

It is interesting to note that the first of these procedures is significantly faster than the second, even though the second one is closer to the way we would write the procedure in a language like C++ or Java. Now that we have a numerical tool for finding Riemann sums let's work on a command for visualizing the Riemann sum. We will create a command plotRS3dLL that will take as input the function, ranges for the x and y directions, the number of subrectangles for each direction and the color of the rectangular solids. Further optional parameters will be allowed and will apply to the surface. For example, we would like to have the following,

\begin{verbatim}
> plotRS3dLL(f(x,y),x=-2..2,y=-2..2,25,25,yellow,
> axes=boxed);
\end{verbatim}

\end{verbatim}

To plot the rectangular solids we need the plottools package and as usual we will probably use something in the plots package, so load both of these into your worksheet.

\begin{verbatim}
> with(plots):
Warning, the name changecoords has been redefined

> with(plottools):
Warning, the name arrow has been redefined
\end{verbatim}
The surface above was produced by the function \( f(x, y) = x \sin(y) + y \sin(x + y) \). It seems like a fine surface to experiment with so define it as a function.

\[
> f := (x, y) \rightarrow x \sin(y) + y \sin(x + y);
\]

\[
> \text{plot3d}(f(x, y), x=-2..2, y=-2..2);
\]

The command that produces a rectangular solid is the cuboid command. The cuboid command takes two three-dimensional points that define diametric corners to the rectangular solid and a color for the solid and produces an image. For example,

\[
> \text{display(cuboid([0,0,0],[1,1,1],color=yellow));}
\]

Note that the cuboid command does not plot on its own. You must use the display command in conjunction with the cuboid command. You can, of course, paste several cubs together using the display command as well.

\[
> p1 := \text{cuboid([0,0,0],[1,1,1],color=yellow)};
> p2 := \text{cuboid([1,0,0],[2,1,2],color=green)};
> p3 := \text{cuboid([0,1,0],[1,2,3],color=blue)};
> p4 := \text{cuboid([1,1,0],[2,2,4],color=red)};
\]
Theoretically, producing the Riemann sum image should be easy. Create the rectangles with the cuboid command and display them along with the surface itself. As with the Riemann sum command we created above, it is an easy process but to generalize it creates a very large command definition. We will begin with a specific example. Take the function defined above and plot it over the region with 250 rectangular solids, 25 in the $x$ direction and 10 in the $y$ direction. To produce the rectangular solids we need to divide the $x$ direction up into 25 equal segments with the expression

$$-2+4i/25$$

and the $y$ direction up into 10 equal segments with the expression

$$-2+4j/10$$

The $z$ coordinate of the bottom of the box must be at zero and the top of the box must have a $z$ coordinate that is on the surface, that is, at

$$evalf(f(-2+4i/25,-2+4j/10))$$

To create the grid of rectangular solids we need to use two seq commands in a similar manner as we did the sum commands in the numeric Riemann sum function we created above. All together the cuboids are produced with the following statement.

$$\text{seq(seq(cuboid([-2+4i/25,-2+4j/10,0],[-2+4*(i+1)/25,-2+4*(j+1)/10, evalf(f(-2+4i/25,-2+4j/10))),color=yellow), i=0..24),j=0..9)}$$

Now all we need to do is place these in a plot with the surface, as below.

$$\text{display(plot3d(f(x,y),x=-2..2,y=-2..2),seq(seq(cuboid([-2+4i/25,-2+4j/10,0],[-2+4*(i+1)/25,-2+4*(j+1)/10,}$$
To generalize this we start with the following declaration.

\[
\text{plotRS3dLL:=(f,xrng,yrng,xdiv,ydiv,boxcolor)->}
\]

The parameters here have exactly the same meaning as in the Riemann3dLL command and the boxcolor is, of course, the color of the boxes. To plot the surface we need only

\[
\text{plot3d}(f,xrng,yrng,\text{seq(args[n],n=7..nargs)})
\]

Note the use of

\[
\text{seq(args[n],n=7..nargs)}
\]

to place all the options, after the six required, as options for the surface plot. In the cuboid commands we need the \(xy\) pairs from

\[
\text{lhs(rhs(xrng))+(rhs(rhs(xrng))-lhs(rhs(xrng)))*i/xdiv,}
\]
\[
\text{lhs(rhs(yrng))+(rhs(rhs(yrng))-lhs(rhs(yrng)))*j/ydiv}
\]

to

\[
\text{lhs(rhs(xrng))+(rhs(rhs(xrng))-lhs(rhs(xrng)))*((i+1)/xdiv,}
\]
\[
\text{lhs(rhs(yrng))+(rhs(rhs(yrng))-lhs(rhs(yrng)))*((j+1)/ydiv}
\]

and we want the \(z\) coordinates to go from 0 to

\[
\text{evalf(subs(lhs(xrng)=lhs(rhs(xrng))+(rhs(rhs(xrng))-lhs(rhs(xrng)))*i/xdiv,}
\]
\[
\text{lhs(yrng)=lhs(rhs(yrng))+(rhs(rhs(yrng))-lhs(rhs(yrng)))*j/ydiv,f))}
\]

So one way to write this monster command is
Again, using a Maple procedure we can make this a bit neater. Here are two possibilities,

> plotRS3dLL:=proc(f,xrng, yrng, xdiv, ydiv, boxcolor)
> local p1,p2,i,j,uxb,lxb,uyb,lyb,xdist,ydist,xc,yc;
> uxb:= rhs(rhs(xrng));
> lxb:= lhs(rhs(xrng));
> uyb:= rhs(rhs(yrng));
> lyb:= lhs(rhs(yrng));
> xdist:=uxb-lxb;
> ydist:=uyb-lyb;
> xc:=lhs(xrng);
> yc:=lhs(yrng);
> p1:=plot3d(f,xrng, yrng, seq(args[n],n=7..nargs));
> p2:=seq(seq(cuboid([lxb+xdist*i/xdiv,lyb+ydist*j/ydiv,0],
> [lxb+xdist*(i+1)/xdiv,lyb+ydist*(j+1)/ydiv,
> evalf(subs(xc=lxb+xdist*i/xdiv,yc=lyb+ydist*j/ydiv,f))],
> color=boxcolor),i=0..xdiv-1),j=0..ydist-1));

> plotRS3dLL(f(x,y),x=-2..2,y=-2..2,25,25,yellow,
axes=boxed);
We will end by creating a couple more commands that are very similar to those we have already produced. Two Riemann sums that are of particular interest are the lower Riemann sum and the upper Riemann sum. Recall that the lower Riemann sum is where the height of each rectangular solid is at the minimum of the surface over the given subrectangle and the upper Riemann sum is where the height of each rectangular solid is at the maximum of the surface over the given subrectangle. The big problem here is that for each of the subregions we need to find the absolute minimum and maximum of the surface. If you recall the work involved in those calculations from the lab on maxima and minima think about doing this 500 or 1,000,000 times. The amount of work would be incredible. Fortunately, Maple has a built-in function that will find the maximum and a function that will find the minimum. The unfortunate thing is that this command can be very slow. Hence we will not be able to generate approximations or images with thousands or millions of rectangles, a few hundred will be the most we can swing. Creating the commands is fairly easy all we need to do is replace the functional
calculation with the maximum or minimum calculation. So for the lower Riemann sum we could use

\[
\text{LRiemann3d} := (f, xrng, yrng, xdiv, ydiv) -> \text{evalf}((\text{rhs}(\text{rhs}(\text{yrng}))) - \text{lhs}(\text{rhs}(\text{yrng}))) / \text{ydiv} * \text{sum}((\text{rhs}(\text{rhs}(\text{xrng}))) - \text{lhs}(\text{rhs}(\text{xrng}))) / \text{xdiv} * \text{sum}((\text{evalf(\text{minimize}(f, \text{lhs}(\text{xrng}) = \text{lhs}(\text{rhs}(\text{xrng}))) + (\text{rhs}(\text{rhs}(\text{xrng}))) - \text{lhs}(\text{rhs}(\text{xrng}))) * (i + 1) / \text{xdiv}, \text{lhs}(\text{yrng}) = \text{lhs}(\text{rhs}(\text{yrng}))) + (\text{rhs}(\text{rhs}(\text{yrng}))) * j / \text{ydiv}) - \text{lhs}(\text{rhs}(\text{yrng}))) * (j + 1) / \text{ydiv})) , i = 0 .. \text{xdiv}-1 , j = 0 .. \text{ydiv}-1));
\]

\[
\text{LRiemann3d}(x^2 + y^2 + 2, x=0..2, y=0..2, 10, 10);
\]

17.12000000

Note that in the above calculation where we used only 100 subregions and a very nice function still took several seconds to complete. If we were to use 10,000 rectangles we would have to wait about 15 minutes for a result. Using a proc, this command would look like,

\[
\text{LRiemann3d} := \text{proc}(f, xrng, yrng, xdiv, ydiv)
\]

\[
\text{local } i, j, \text{uxb}, \text{lxb}, \text{uyb}, \text{lyb}, \text{xdist}, \text{ydist}, \text{xc}, \text{yc}, \text{hsum};
\]

\[
\text{uxb} := \text{rhs}(\text{rhs}(\text{xrng}));
\]

\[
\text{lxb} := \text{lhs}(\text{rhs}(\text{xrng}));
\]

\[
\text{uyb} := \text{rhs}(\text{rhs}(\text{yrng}));
\]

\[
\text{lyb} := \text{lhs}(\text{rhs}(\text{yrng}));
\]

\[
\text{xdist} := \text{uxb} - \text{lxb};
\]

\[
\text{ydist} := \text{uyb} - \text{lyb};
\]

\[
\text{xc} := \text{lhs}(\text{xrng});
\]

\[
\text{yc} := \text{lhs}(\text{yrng});
\]

\[
\text{hsum} := \text{sum}((\text{evalf(\text{minimize}(f, \text{xc} = \text{lxb} + \text{xdist} * i / \text{xdiv} .. \text{lx} + \text{xdist} * (i + 1) / \text{xdiv}, \text{yc} = \text{lyb} + \text{ydist} * j / \text{ydiv} .. \text{y} + \text{ydist} * (j + 1) / \text{ydiv})), i = 0 .. \text{xdiv}-1 , j = 0 .. \text{ydiv}-1));
\]

\[
\text{evalf(\text{xdist} / \text{xdiv} * \text{ydist} / \text{ydiv} * \text{hsum});}
\]

\end proc:

\[
f := (x, y) -> x * \sin(y) + y * \sin(x + y);
\]

\[
f := (x, y) -> x \sin(y) + y \sin(x + y)
\]

\[
\text{LRiemann3d}(f(x, y), x=2..5, y=-3..1, 5, 5);
\]

-37.59793749

Note that the last command took approximately 5 minutes to calculate, with only 25 subregions! As for the image of the lower Riemann sum we could use either
plotLRS3d:=(f,xrng,yrng,xdiv,ydiv,boxcolor)->display(
plot3d(f,xrng,yrng,seq(args[n],n=7..nargs)),seq(seq(cuboid(
[lhs(rhs(xrng))+(rhs(rhs(xrng))-lhs(rhs(xrng)))*i/xdiv,
lhs(rhs(ymrng))+(rhs(rhs(ymrng))-lhs(rhs(ymrng)))*j/ydiv,0],
[lhs(rhs(xrng))+(rhs(rhs(xrng))-lhs(rhs(xrng)))*i/xdiv,
lhs(rhs(ymrng))+(rhs(rhs(ymrng))-lhs(rhs(ymrng)))*j/ydiv,
minimize(f*1.0,lhs(xrng)=lhs(rhs(xrng))+(rhs(rhs(xrng))-
lhs(rhs(xrng)))*i/xdiv..lhs(rhs(xrng))+(rhs(rhs(xrng))-
lhs(rhs(xrng)))*i+1/xdiv,lhs(ymrng)=lhs(rhs(ymrng))+(rhs(rhs(ymrng))-
lhs(rhs(ymrng)))*j/ydiv..lhs(rhs(ymrng))+(rhs(rhs(ymrng))-
lhs(rhs(ymrng)))*j+1/ydiv),
color=boxcolor),i=0..xdiv-1),j=0..ydiv-1)):

or using proc

plotLRS3d:=proc(f,xrng,yrng,xdiv,ydiv,boxcolor)
local p,p1,i,j,uxb,lxb,uyb,lyb,xdist,ydist,xc,yc,hsum,
xpos,ypos,nxpos,nypos;
uxb:= rhs(rhs(xrng));
lxb:= lhs(rhs(xrng));
uyb:= rhs(rhs(yrng));
lyb:= lhs(rhs(yrng));
xdist:=uxb-lxb;
ydist:=uyb-lyb;
xc:=lhs(xrng);
yc:=lhs(yrng);
p1:=plot3d(f,xrng,yrng,seq(args[n],n=7..nargs));
for i from 0 to xdiv-1 do
xpos:=lxb+xdist*i/xdiv;
xnpos:=lxb+xdist*(i+1)/xdiv;
for j from 0 to ydiv-1 do
ypos:=lyb+ydist*j/ydiv;
nynpos:=lyb+ydist*(j+1)/ydiv;
p[i*ydiv+j]:=cuboid([xpos,ypos,0],[xnpos,nypos,evalf(minimize(f,xc=xpos..xnpos,yc=ypos..nypos))],
color=boxcolor)
end do;
end do;
display(p1,seq(seq(p[i*ydiv+j],i=0..xdiv-1),
j=0..ydiv-1));
end proc:

The following command took approximately 90 seconds.
Exercises:

1. Create the command Riemann3dLR that takes the same input as Riemann3dLL but produces the Riemann sum using the lower right point of each subrectangle. Use it to find the Riemann sum approximation for the double integral of 

\[ f(x, y) = x^2 \sin(x - y^2) \]

over the region \([-2, 2] \times [-2, 4]\) using 100, 200, 300, … 1000 divisions in each direction. What are these numbers converging to? If they don’t seem to be converging state why.

2. Create the command Riemann3dUL that takes the same input as Riemann3dLL but produces the Riemann sum using the upper left point of each subrectangle. Use it to find the Riemann sum approximation for the double integral of 

\[ f(x, y) = x^2 \sin(x - y^2) \]

over the region \([-2, 2] \times [-2, 4]\) using 100, 200, 300, … 1000 divisions in each direction. What are these numbers converging to? If they don’t seem to be converging state why.

3. Create the command Riemann3dUR that takes the same input as Riemann3dLL but produces the Riemann sum using the upper right point of each subrectangle. Use it to find the Riemann sum approximation for the double integral of 

\[ f(x, y) = x^2 \sin(x - y^2) \]

over the region \([-2, 2] \times [-2, 4]\) using 100, 200, 300, … 1000 divisions in each direction. What are these numbers converging to? If they don’t seem to be converging state why.

4. Create the command Riemann3dMP that takes the same input as Riemann3dLL but produces the Riemann sum using the midpoint of each subrectangle. Use it to find the Riemann sum approximation for the double integral of 

\[ f(x, y) = x^2 \sin(x - y^2) \]

over the region \([-2, 2] \times [-2, 4]\) using 100, 200, 300, … 1000 divisions in each direction. What are these numbers converging to? If they don’t seem to be converging state why.

5. Create the command URiemann3d that takes the same input as Riemann3dLL but produces the upper Riemann sum. Use it to find the Riemann sum approximation for
the double integral of $f(x, y) = x^2 - 2xy + y^3$ over the region $[0,3] \times [5,7]$ using 5, 10, 15 and 20 divisions in each direction. What are these numbers converging to? If they don’t seem to be converging state why. Do the same approximations with Riemann3dLL, Riemann3dLR, Riemann3dUL, Riemann3dUR, Riemann3dMP and LRiemann3d. How do these approximations compare? Why do you think they are related in this way?

6. Create the command plotRS3dLR that takes the same input as plotRS3dLL but produces the Riemann sum image using the lower right point of each subrectangle. Use it to display the Riemann sum approximation image for the double integral of $f(x, y) = x^2 \sin(x - y^3)$ over the region $[-2,2] \times [-2,4]$ using 10, 20, 30, … 100 divisions in each direction.

7. Create the command plotRS3dUL that takes the same input as plotRS3dLL but produces the Riemann sum image using the upper left point of each subrectangle. Use it to display the Riemann sum approximation image for the double integral of $f(x, y) = x^2 \sin(x - y^3)$ over the region $[-2,2] \times [-2,4]$ using 10, 20, 30, … 100 divisions in each direction.

8. Create the command plotRS3dUR that takes the same input as plotRS3dLL but produces the Riemann sum image using the upper right point of each subrectangle. Use it to display the Riemann sum approximation image for the double integral of $f(x, y) = x^2 \sin(x - y^3)$ over the region $[-2,2] \times [-2,4]$ using 10, 20, 30, … 100 divisions in each direction.

9. Create the command plotRS3dMP that takes the same input as plotRS3dLL but produces the Riemann sum image using the midpoint of each subrectangle. Use it to display the Riemann sum approximation image for the double integral of $f(x, y) = x^2 \sin(x - y^3)$ over the region $[-2,2] \times [-2,4]$ using 10, 20, 30, … 100 divisions in each direction.

10. Create the command plotURS3d that takes the same input as plotRS3dLL but produces the Riemann sum image using the upper Riemann sum. Use it to display the Riemann sum approximation image for the double integral of $f(x, y) = e^{\frac{1}{x+y}}$ over the region $[-1,1] \times [-1,1]$.

11. Use the Riemann sum to approximate the double integral of $f(x, y) = e^{\frac{1}{x+y}}$ over the region $[-1,1] \times [-1,1]$.

12. Use the Riemann sum to approximate the double integral of $f(x, y) = e^{\frac{1}{x+y}}$ over the region $[0,1] \times [0,1]$.

13. We define the triple integral over a rectangular solid region to be

$$
\iiint_E f(x, y, z) dV = \lim_{m,n,p \to \infty} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{p} f(x_{ik}, y_{jk}, z_{jk}) \Delta V
$$

Hence the approximating Reimann sum would be defined as

$$
\sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{p} f(x_{ik}, y_{jk}, z_{jk}) \Delta V
$$

Create a new command called Reimann4dMP that begins as follows,
Riemann4dMP:=(f,xrng,yrng,zrng,xdiv,ydiv,zdiv)->
and calculates the Reimann sum of the function over the rectangular solid defined by
the ranges using the given subdivisions and the midpoint of the subrectangular solid
as the test value.

14. Use the Reimann4dMP command to approximate the triple integral of the function
f(x, y) = \frac{1}{xy} over the region [0,1] \times [0,1] \times [0,1]. It is interesting to note that this
integral is exactly \[ \sum_{n=1}^{\infty} \frac{1}{n}, \] and no one has ever been able to find the exact value of
either of these.