Take Your Chances: Try the Monte Carlo Technique for Numerical Integration in Three Dimensions

Introduction

OBJECTIVE: Learn the Monte Carlo method for approximating a multiple integral that cannot be integrated symbolically.

How can you use a game of chance to evaluate a multiple integral? That is just what you will do with this project. You will generate random points within a fixed region and then estimate the volume of the desired portion by considering the percentage of random points that fall within the boundaries of the desired portion. Since this will only estimate the exact volume, you will also explore the accuracy of this method.

Technology Guidelines

NOTE: If you have just finished a module, restart *Mathematica* or close the *Kernel* before executing a new module.
TO OPEN CELLS, put your cursor on the right cell bracket and double click.
TO STOP AN EXECUTION
Select the *Kernel* pull-down menu and click on *Abort Evaluation.*
ORDER OF EXECUTION
Execute cells in the order given. Do not skip any Input cells within a given notebook.
SAVING NOTEBOOKS
You can save anytime to any directory you choose, and it is wise to save often.
However, before you do your final save, it is a good idea to delete all your output by selecting the
Delete All Output selection under the *Kernel* pull-down menu.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, then shut down *Mathematica* and start it up again.

Part I: The Monte Carlo Method

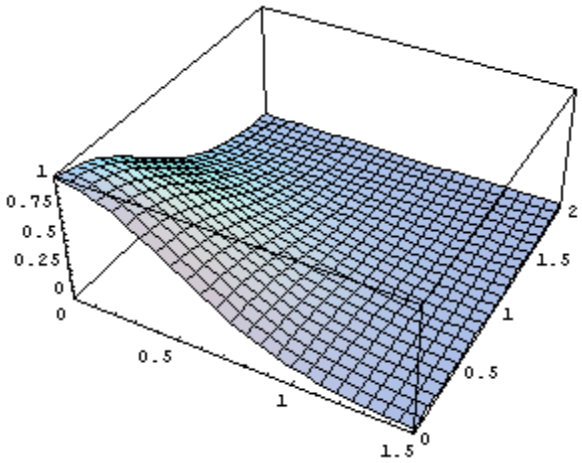
The Monte Carlo method is an example of a numerical integration technique. In this method, the volume under a surface is completely enclosed in a rectangular box and points within that box are randomly selected. The number of points that land under the surface is determined and the volume under the surface to the bottom of the box is approximated as the percentage of the generated points lying under the surface times the volume of the entire box. In the example that follows, 1000 points are generated randomly to estimate the volume. Note that the sample function used here cannot be integrated symbolically, so we are using *Mathematica*'s numerical integrator, which uses a different approximation technique, to come up with an answer that we would expect to be close to the actual one.

In utilizing the **Random** command in *Mathematica*, the default **Random[]** selects numbers between 0 and 1; other selections are specified. The following commands will make the dimesions of the box selected 2 by 3 by 1, giving a volume of 6. When applying a function of your own to this procedure, be certain to carefully construct the box in which you enclose your desired volume. In choosing a function, be careful to choose one that is not negative in the specifed domain, and note that you are finding the volume between the surface and the *x*-*y* plane. We begin by defining the function, specifying the region over which we will generate points, and then looking at the graph of the function and the region

In[1]:=

```
Off[General::spell]

Off[General::spell1]

Clear[x, y, z, f]

f[x_, y_] := e^(-2 x^2 - y^2)

xminf = 0;

xmaxf = 1.5;

yminf = 0;

ymaxf = 2;

zminf = 0;
```

```
zmaxf = 1;

pf = Plot3D[f[x, y], {x, xminf, xmaxf}, {y, ymi
  PlotRange -> All];

actualf = NIntegrate[f[x, y], {x, xminf, xmaxf
  {y, yminf, ymaxf}];

Print["actual volume = ", actualf]
```




```
actual volume = 0.55127
```

Now we will generate random points in our specified region and check to see whether or not they are under our surface. The points in **listin** are under the surface in the volume we are computing and they will be plotted in red. The points in **listout** are not under the surface and they will be plotted in black. The scatter plot command requires the following package which must be read in first.

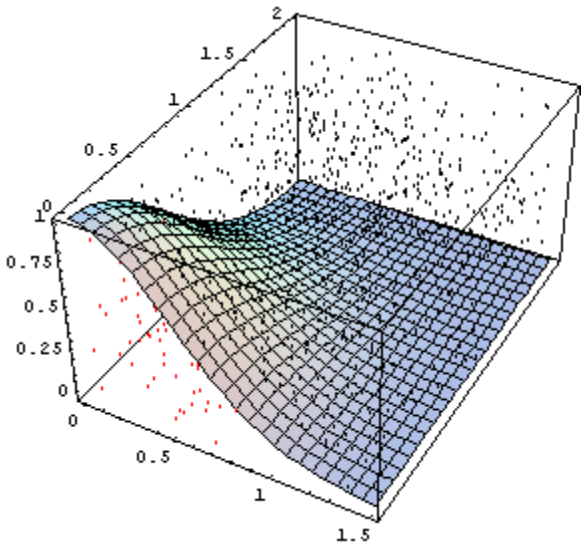
In[14]:=

```
<<Graphics`Graphics3D`
```

In[15]:=

```
numberofpoints = 1000;

volumeofboxf = (xmaxf - xminf) (ymaxf - yminf) (

count = 0;
```

```
listout = {};

listin = {};

Do[{Clear[x, y, z], SeedRandom[],
  x = Random[Real, {xminf, xmaxf}],
  y = Random[Real, {yminf, ymaxf}],
  z = Random[Real, {zminf, zmaxf}],
          If[z ≤ f[x, y],
   {count = count + 1, AppendTo[listin, {x, y, 
   AppendTo[listout, {x, y, z}]]}, {numberofp

pout = ScatterPlot3D[listout, DisplayFunctio

pin = ScatterPlot3D[listin, PlotStyle -> RGBCo
  DisplayFunction -> Identity];

Show[pin, pout, pf, DisplayFunction -> $Displ

Print["actual = ", actualf]

estimate = N[volumeofboxf count / numberofpoin

Print["estimate = ", estimate]

error = estimate - actualf;

Print["error = ", error]

relativeerror = error / actualf;

Print["relative error = ", relativeerror]
```



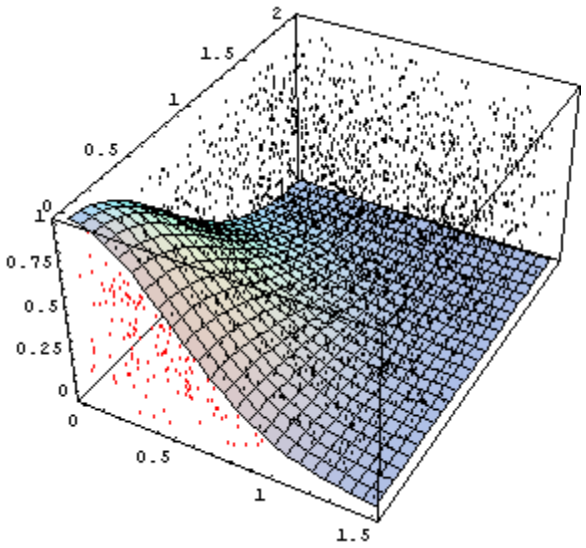

```
actual = 0.55127

estimate = 0.537

error = -0.0142702

relative error = -0.025886
```

How does your error compare to that of one of your classmates or to a second calculation you produce? Note what happens when you increase the number of points you use for the calculation.

In[31]:=

```
Clear[x, y, z]

numberofpoints = 3000;

count = 0;

listout = {};

listin = {};
```

```
Do[{Clear[x, y, z], x = Random[Real, {xminf, x
  y = Random[Real, {yminf, ymaxf}],
  z = Random[Real, {zminf, zmaxf}],
          If[z ≤ f[x, y],
   {count = count + 1, AppendTo[listin, {x, y, 
   AppendTo[listout, {x, y, z}]]}, {numberofp

pout = ScatterPlot3D[listout, DisplayFunctio

pin = ScatterPlot3D[listin, PlotStyle -> RGBCo
  DisplayFunction -> Identity];

Show[pin, pout, pf, DisplayFunction -> $Displ

Print["actual = ", actualf]

estimate = N[volumeofboxf count / numberofpoin

Print["estimate = ", estimate]

error = estimate - actualf;

Print["error = ", error]

relativeerror = error / actualf;

Print["relative error = ", relativeerror]
```




```
actual = 0.55127

estimate = 0.551

error = -0.000270193

relative error = -0.000490128
```

Does this method seem very accurate? We will analyze the error involved in such an approximation.

You Try It: Part I

Choose your own function; just remember to make it nonnegative over the region of integration. The only commands you need to enter in the following template are your function and minimum and maximum values for each of your variables. These are all in red (leave your **zmin** at 0) . Then, simply execute the rest of the commands. Note that we are calling the function in the *You Try It* sections *g(x,y)* instead of *f(x,y)*.

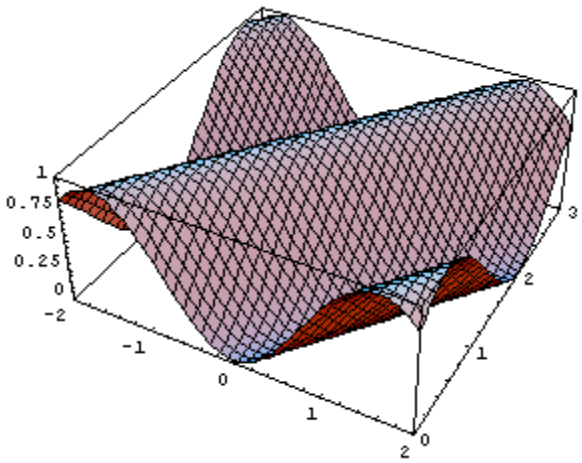
You should realize that the bigger your rectangular region, the more points you will need to generate to get an answer that is reasonably accurate. Why?

We will begin by plotting the function and letting *Mathematica* estimate the volume between the surface and the *xy*-plane over the domain specified. **Be sure to select a function that**

stays above the *xy*-plane.

In[47]:=

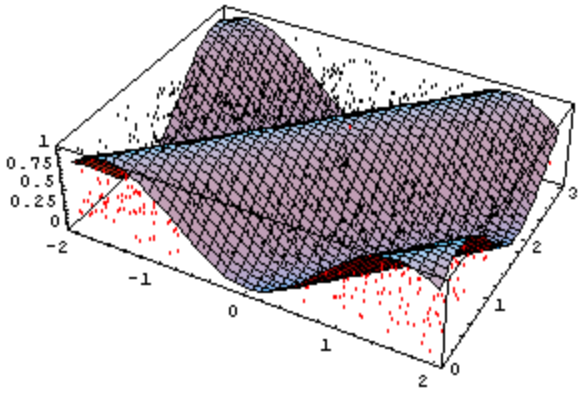
```
Clear[x, y, z, g]

g[x_, y_] = Sin[x - y]^2;

xming = -2;

xmaxg = 2;

yming = 0;

ymaxg = 3;

zming = 0;

zmaxg = 1;

pfg = Plot3D[g[x, y], {x, xming, xmaxg}, {y, ym
  PlotRange -> All, PlotPoints → 40];

actualg = NIntegrate[g[x, y], {x, xming, xmaxg
  {y, yming, ymaxg}];

Print["actual volume = ", actualg]
```



```
actual volume = 5.94713
```

In[58]:=

```
numberofpoints = 2000;

volumeofboxg = (xmaxg - xming) (ymaxg - yming) (

count = 0;

listout = {};

listin = {};

Do[{Clear[x, y, z], x = Random[Real, {xming, x
  y = Random[Real, {yming, ymaxg}],
  z = Random[Real, {zming, zmaxg}],
          If[z ≤ g[x, y],
   {count = count + 1, AppendTo[listin, {x, y, 
   AppendTo[listout, {x, y, z}]]}, {numberofp

pout = ScatterPlot3D[listout, DisplayFunctio

pin = ScatterPlot3D[listin, PlotStyle -> RGBCo
  DisplayFunction -> Identity];

Show[pin, pout, pfg, DisplayFunction -> $Disp

Print["actual = ", actualg]

estimate = N[volumeofboxg count / numberofpoin

Print["estimate = ", estimate]

error = estimate - actualg;

Print["error = ", error]

relativeerror = error / actualg;

Print["relative error = ", relativeerror]
```




```
actual = 5.94713

estimate = 6.078

error = 0.130866

relative error = 0.0220048
```

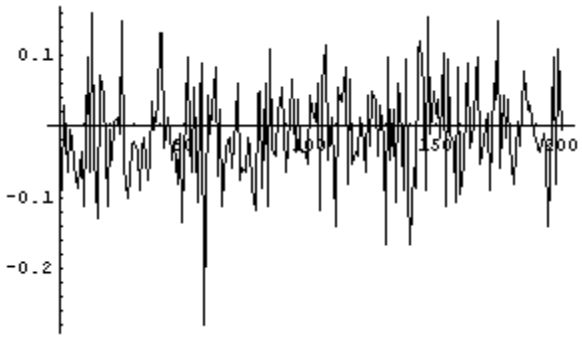
Part II: Error Analysis

In this part, we follow the procedure above with 1000 points; then, we repeat that procedure 200 times and look at the pattern of the errors. Because of the many steps involved, this computation may take several minutes.

In[74]:=

```
Clear[x, y, z]

numberofpoints = 1000;

errorlist = {};

repeat = 200;

Print["actual = ", actualf]

count = 0;
```

```
Do[
 {count = 0,
  Do[{Clear[x, y, z], x = Random[Real, {xminf,
    y = Random[Real, {yminf, ymaxf}],
    z = Random[Real, {zminf, zmaxf}],
          If[z ≤ f[x, y], count = count + 1]},
          estimate = N[volumeofboxf count / n
  error = estimate - actualf, AppendTo[errorli
    {repeat}]

Clear[relativeerrorlist]

relativeerrorlist = errorlist / actualf;

ListPlot[relativeerrorlist, PlotJoined → Tru

actual = 0.55127
```



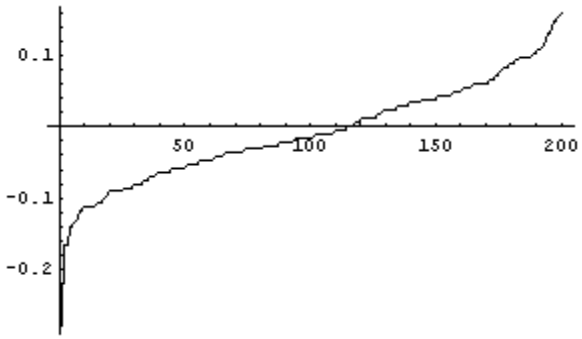
Look at the pattern of your errors. These are referred to as random errors, and they will usually be centered around 0 and should show no particular pattern. We can order them and look at that picture as well.

In[84]:=

```
sortlist = Sort[relativeerrorlist]

ListPlot[sortlist, PlotJoined → True];
```

Out[84]=

```
{-0.281659, -0.167377, -0.167377,
 -0.140168, -0.140168, -0.134726,
 -0.129284, -0.1184, -0.1184, -0.112958,
 -0.112958, -0.112958, -0.112958,
 -0.112958, -0.107516, -0.107516,
 -0.107516, -0.102074, -0.0966317,
 -0.0911898, -0.0911898, -0.0911898,
 -0.0911898, -0.0911898, -0.0911898,
 -0.0857478, -0.0857478, -0.0857478,
 -0.0857478, -0.0803058, -0.0803058,
 -0.0803058, -0.0803058, -0.0748638,
 -0.0748638, -0.0694218, -0.0694218,
 -0.0694218, -0.0639799, -0.0639799,
 -0.0639799, -0.0639799, -0.0639799,
 -0.0639799, -0.0585379, -0.0585379,
 -0.0585379, -0.0585379, -0.0585379,
 -0.0585379, -0.0530959, -0.0530959,
 -0.0530959, -0.0530959, -0.0530959,
 -0.0476539, -0.0476539, -0.0476539,
 -0.0476539, -0.0476539, -0.0476539,
 -0.042212, -0.042212, -0.042212,
 -0.042212, -0.03677, -0.03677,
 -0.03677, -0.03677, -0.03677, -0.03677,
 -0.03677, -0.03677, -0.031328,
 -0.031328, -0.031328, -0.031328,
 -0.031328, -0.031328, -0.031328,
 -0.031328, -0.025886, -0.025886,
 -0.025886, -0.025886, -0.025886,
 -0.025886, -0.020444, -0.020444,
 -0.020444, -0.020444, -0.020444,
 -0.020444, -0.0150021, -0.0150021,
 -0.0150021, -0.0150021, -0.0150021,
 -0.0150021, -0.0150021, -0.0150021,
 -0.00956009, -0.00956009, -0.00956009,
 -0.00956009, -0.00956009, -0.00956009,
 -0.00956009, -0.00956009, -0.00411811,
 -0.00411811, -0.00411811, -0.00411811,
 -0.00411811, 0.00132387, 0.00132387,
 0.00132387, 0.00676584, 0.00676584,
 0.00676584, 0.0122078, 0.0122078,
 0.0122078, 0.0122078, 0.0122078,
 0.0122078, 0.0176498, 0.0176498,
 0.0230918, 0.0230918, 0.0230918,
 0.0230918, 0.0230918, 0.0230918,
 0.0285338, 0.0285338, 0.0285338,
 0.0285338, 0.0285338, 0.0339757,
 0.0339757, 0.0339757, 0.0339757,
```



You should run this again, increasing the number of repeats and/or the number of points to see what happens to the errors. Which will decrease your error size?

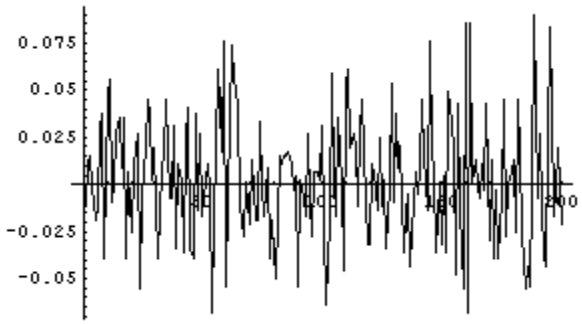
You Try It: Part II

You can extend the problem you began in the previous You Try It section by analyzing your **errorlist** when you repeat the Monte Carlo method with 1000 points being generated each of 200 times.

In[86]:=

```
Clear[x, y, z, errorlist, percenterrorlist]

numberofpoints = 1000;

errorlist = {};

repeat = 200;

Print["actual = ", actualg]

count = 0;
```

```
Do[
 {count = 0,
  Do[{Clear[x, y, z], x = Random[Real, {xming,
    y = Random[Real, {yming, ymaxg}],
    z = Random[Real, {zming, zmaxg}],
          If[z ≤ g[x, y], count = count + 1]},
          estimate = N[volumeofboxg count / nu
  error = estimate - actualg, AppendTo[errorli
    {repeat}]

Clear[relativeerrorlist]

relativeerrorlist = errorlist / actualg;

ListPlot[relativeerrorlist, PlotJoined → Tru

actual = 5.94713
```



Are your errors centered around 0? If not, there might be a mistake here.
See what happens to the range of your errors when you increase the **numberofpoints** command above.

Created by *Mathematica* (April 3, 2005)