Integration by Stone Throwing

Learning goal: To experience a computational approach to determine the areas of arbitrary shapes by using random numbers to simulate a physical experiment in which stones are thrown randomly. Equivalently, to experience how random numbers can be used to evaluate integrals of arbitrary functions.

Learning objectives

- To experience a new type of mathematics employing chance, in this case via the use of random numbers.
- To use the computer to conduct an "experiment" that can readily be done in the physical world, that is, to simulate the physical activity of random stone throwing.
- To use a random number generator to simulate random events.
- To develop a graphical and concrete understanding of what an integral means.

Activities

In this module we examine a technique for determining areas that uses random numbers. Although it is not the most precise method for integration, it is probably the simplest and is the method of choice for very complicated problems.

Products

At the end of this module students will have learned a technique to determine areas numerically, or equivalently, to perform the definite integration of an arbitrary function.

Problem

Imagine yourself as a water lily farmer who wishes to plant a new crop of lilies in one of your distant ponds. Being a person of action, you grab your sack of Cargill copyrighted water lily seeds and lug it all the way out to that distant pond. You arrive and then read the planting instructions (something you should have done before), only to discover that you need to know the area of the pond to determine how many seeds to scatter. As befits the impulsive type of person that you are, you did not bring any measurement instrument with you and do not have the patience to schlep all the way back to your barn to get them. So what's an impulsive, impatient farmer to do? [Think about this for a while before reading on to see if you figure out how to determine the area of the pond using just your body and what you may find lying about.]

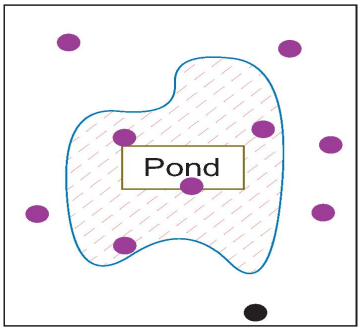


Figure 1 A pond of odd shape whose area we wish to determine. A box of known area is drawn around the pond and the ground within the box cleared of stones. Then handfuls of stones are thrown randomly and uniformly up into the air, and the number of splashes in the pond are counted, as well as the number of stones that have landed on the ground within the box.

Solution: As outlined in Figure 1, this is one way to solve the problem of the compulsive farmer:

1. Grab a stick and use it to trace out a rectangle on the ground that completely encloses the pond.
2. Clear the land area inside the box of all the stones lying on the ground, and fill your pockets with them.
3. Using your boot as an approximate one foot ruler, measure the length and the width of the box. Since the area of the box in square feet A_{box} is just its length times its width, you now know the area of the box.
4. With your eyes closed and while spinning yourself around, throw stone after stone from your pocket up into the air in a rather uniform fashion, all the time counting the number of splashes N_{splash} that you hear.
5. When you are out of stones, open your eyes, regain your equilibrium, and count the number of stones N_{ground} lying on the ground within the box.
6. Common sense, or maybe divine inspiration, tells you that if you have distributed the stones uniformly, then the ratio of the area of the pond to the area of the box is just the ratio of the number of rocks landing in the pond to the total number of rocks landing within the perimeter of the box:

   (1) $$\frac{A_{pond}}{A_{box}} = \frac{N_{splash}}{N_{splash}+ N_{box}}.$$

   Now since we know the approximate area of the box in square feet (or square boots), we can determine the area of the pond by multiplying both sides of (1) by A_{box} :

   (2) $$A_{pond} = A_{box} \times \frac{N_{splash}}{N_{splash}+ N_{box}}.$$

Algorithm

Rather than take the chance of getting our boots all wet and muddy dealing with a real pond, let us image that our pond is a circle of radius 1. We know that the area of a circle $A = \pi r^2$, which for a unit circle (circle with radius 1) is just π. In any case, since we know that $\pi \simeq 3.1415927$, we can try out our new method of integration to calculate the area of a circular pond, and then compare it to the known value of π as a check. (This means that we will calculate π using random numbers and logic.)

1. First we generate pairs of random numbers r_i in the interval 0 to 1:

(3) $$0 \leq r_i \leq 1$$

where *i = 0, 1, …* is an integer that we use to label the numbers in our random sequence.

2. Next, as shown in Figure 2, we imagine a square of side 1 drawn around one quarter of a circle.

3. Then we generate two random numbers and use them as the *x* and *y* coordinates of our stone's landing position:

(4) $$(x_i, y_i) = (r_i, r_{i+1})$$

4. Next, for each landing position, we record a slash whenever the position lies within the circle, that is, whenever

(5) $$x_i^2 + y_i^2 \leq 1$$

5. We repeat the process for a large number of stones in order to obtain “good” statistics.

6. Since our stones (random numbers) always lie between 0 and 1, they must always land within the unit square in Figure 2. Since the area of the square is 1 while the area of the quarter circle within the square is π/4, our algorithm for integration via stone throwing can be thought of as a test of how well

$$\pi \cong 4 \times \frac{N_{splash}}{N_{tot}} \tag{6}$$

where N_{tot} is the total number of random pairs (splashes) generated.

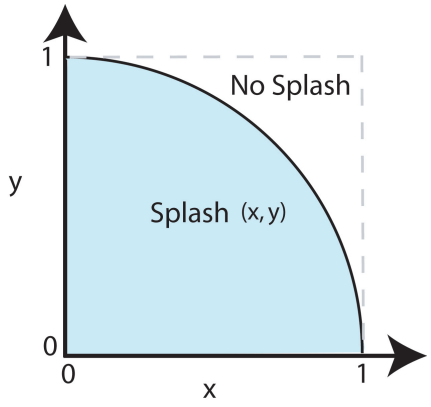


Figure 2 A quarter of a unit circle whose area is to be determined by stone throwing. If the stone lands at *(x, y)* and $x^2 + y^2 < 1$, then a splash occurs. Notice since the box touches the edges of the circle, must stones will land within the circle. As the box is made bigger, the fraction of stones that hit the circle decreases, as does the precision of the answer.

Implementation

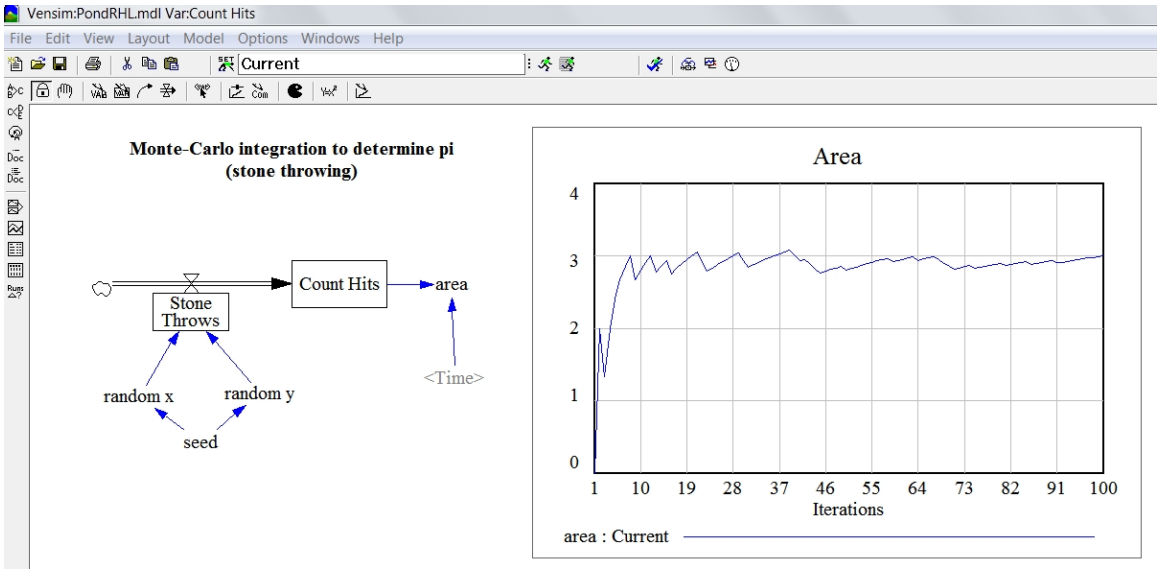
We want you to create a program using your favorite programming tool in order to calculate π using the stone throwing technique. (We give our sample programs below.) We want the program to yield a value of π accurate to *two* decimal places, and then we want to change the program so that it yields a value of π accurate to *three* decimal places.

Implementation: Pond.py

```
#  Pond.py:  Calculate pi via stone throwing

from visual import *
import random
from visual.graph import *
import random
from visual.graph import *

max = 300                              #number of stones to be thrown
hits = 0
#  Pond.py:  Calculate pi via stone throwing

from visual import *
import random
from visual.graph import *

max = 300                              #number of stones to be thrown
hits = 0
graph1 = gdisplay(width = 600, height = 400, title = 'Pond', xtitle =
'iteration', ytitle = 'PiCalc', xmax = 1000., xmin = 0., ymax = 10., ymin = 0.)
pts = gdots(shape = 'square', size = 2, color = color.green)
for i in arange(1, max+1):
    x = random.random()        #creates floats between 0 and 1
    y = random.random()
    if ((x*x + y*y) < 1):              #stone hits the pond
        hits = hits+1
        PiCalc = 4.*hits/i                    #calculate area
        print 'i= ',i ,' PiCalc=' , PiCalc,'x,y=',x,',',y
        pts.plot(pos=(i,PiCalc))
```

Implementation Excel: Stones.xls

Implementation, Vensim PondRHL.mdl



(01) area = 4*Count Hits/Time	(06) random y= RANDOM UNIFORM(-1 , 1 , seed)
(02) Count Hits= INTEG (Stone Throws, 0)	(07) SAVEPER = TIME STEP The frequency with which output is stored.
(03) FINAL TIME = 100 The final time for the simulation.	(08) seed= 68111
(04) INITIAL TIME = 1 The initial time for the simulation.	(09) Stone Throws= IF THEN ELSE(random x*random x+random y*random y <1 , 1 , 0)
(05) random x= RANDOM UNIFORM(-1 , 1 , seed)	(10) TIME STEP = 1 The time step for the simulation.

Assessment

Because we are throwing the stones randomly, and simulating that with random numbers, we can apply statistical analysis to this problem. We're not going to bother you with that sort of stuff here, but just state the results that such an analysis tells us that we expect that *on the average* the relative error in the computed answer to vary like

$$\text{Relative Error} \simeq \frac{1}{\sqrt{N}}, \tag{7}$$

where *N* is the number of stones thrown.

Now listen up! When we say "on the average" we mean that you need to run the simulation or experiment multiple times, and then take the average of all the results. This tends to remove much of the random fluctuations. How many simulations should you average over? Not too many. As a crude rule of thumb, if you use *N* points in a single simulation, then some number of simulations smaller than $\sqrt{N}$ should be good (even as small as $\sqrt{\sqrt{N}}$).

All of this means that to obtain two good decimal places in your answer, as we have asked you to do above, your relative error would be 1 in 100, that is 1/100 = 0.1. Accordingly, we can use Equation (7) above to determine an approximate value for the number of stones that you need to throw to obtain two good decimal places:

$$\frac{1}{100} \cong \frac{1}{\sqrt{N}} \tag{8}$$

To solve for *N,* we square both sides of this equation to remove the square root:

$$\frac{1}{10{,}000} \cong \frac{1}{N} \tag{9}$$

$$N \cong 10{,}000 \tag{10}$$

1. See if the number of points you needed to obtain two places of precision is within a factor of two or three of the value deduced from Equation (10).

2. The best way is to make multiple trials and take the average of the value of the value of π obtained for each. So let's say that you want to use $N \cong 10{,}000$ points. The $\sqrt{10{,}000}$ = 100, and the $\sqrt{100}$ = 10. So take the average of more than 10 and less than 100 trials.
3. This same type of analysis would indicate that you would need a million, 1,000,000 points to obtain another place of precision, that is, 100 times more points than needed for two places of precision. Compare this prediction with the increase in the number of points you needed in order to obtain an additional place of precision.
4. We have used a box of unit size that touches the sides of the semicircle whose area we are computing. Change the program so that the stones are thrown within a box of side 2. Comment on how this changes the statistics (and explain if you can).

Extension: General Functions

We have seen how to determine the area of a circular pond. We can follow the same technique to determine the area of an arbitrary pond shape, such as the one in Figure 1, as long as we have some equation or graph that defines the perimeter so that we can tell if the stone falls within the area. Although we are strong supporters of Future Farmers of America, the technique we have been describing has broader applicability that just to lily ponds, and in fact is what mathematicians would call the evaluation of a definite integral of the type studied in calculus:

$$\int_a^b f(x)\,dx \tag{11}$$

To see this connection, consider the left of Figure 3 where we have plotted an arbitrary function *f(x).* The definite integral in Equation (11) is just the area under the curve of *f(x) versus x* between the limits *x=a* and *x=b*. We show this area in the middle pane of Figure 3 with stripes.

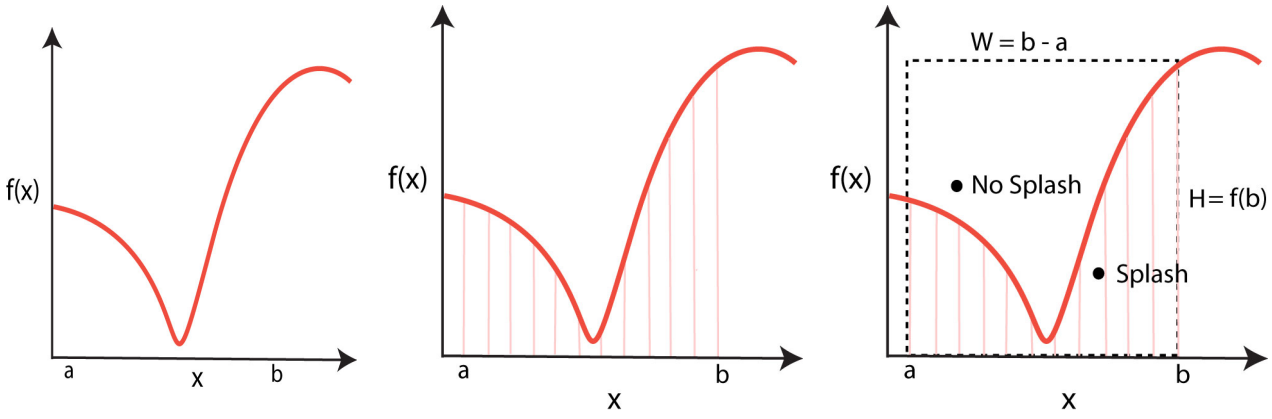


Figure 3 *Left:* The function we wish to integrate from *a* to *b. Center:* The integral is equal to shaded area under the curve. *Right:* a rectangle of height *H* and width *W* that encloses the area we wish to determine. Stones landing below the curve are accpted while those above the curve are rejected.

If we look again at the middle of Figure 3, we note that the striped area is just what we have been calling a "pond", with its perimeter defined by the function, the vertical lines at *x=a* and *x=b*, and the *x* axis. So here are the steps to follow in order to determine the area of this type of pond:

1. A dashed box is drawn to enclose the function. The exact size of the box does not really matter, but making it too big will lead to an excessively large number of rejected stones, and thus lower

precision. In the present case we have chosen the box to have a width *W = b-a* and a height *H = f(b).*

2. Throw "stones" within the box by successively picking pairs of random numbers (r_{2i}, r_{2i-1}), and then use these numbers to determine where a stone lands within the *f(x) versus x* plot:

(12) $$(x_i,\ y_i\ ) = (r_{2i},\ r_{2i-1}),\quad i = 1, 2, 3, \ldots$$

3. Note that usually the computer's algorithm generates random number in the range $0 \le r_i \le 1$, and so unless your box is a square with sides 1, the random numbers you generate will not cover the entire box. That being the case, all you have to do is scale the computer's random numbers to fit your box. For example, for a rectangular box with sides of length *L* and width *W*, you would pick

(13) $$(x_i,\ y_i\ ) = (L \times r_{2i},\quad W \times r_{2i-1}),\quad i = 1, 2, 3, \ldots$$

4. Note where the stones land. Those that land below the curve of *f(x)* increase the value of N_{splash} by 1. Those stones that land above the curve get added to the value of N_{box}.

5. After a good number of stones have been thrown, we can again apply Equation (2) to determine the area under the curve:

(14) $$\int_a^b f(x)\,dx = A_{box} \times \frac{N_{splash}}{N_{splash} + N_{box}} = (W \times H) \times \frac{N_{splash}}{N_{splash} + N_{box}}$$

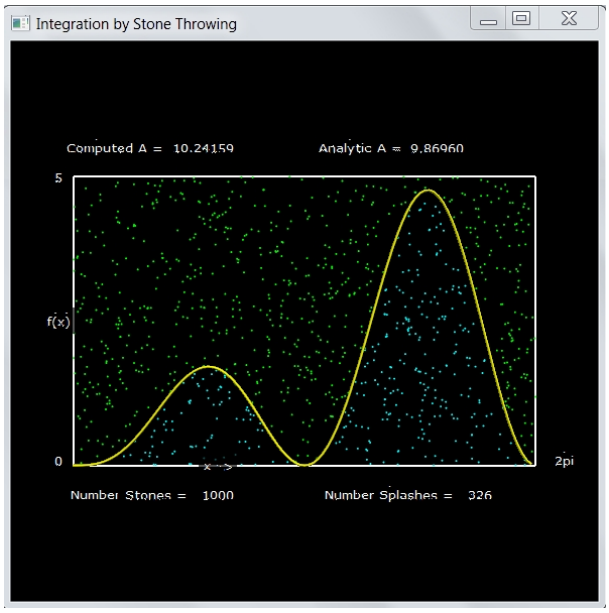


Figure 4 The output from `StoneThrowing.py` showing the function $x\ sin^2 x$ being integrated and the randomly distributed splashes. The poor agreement between computed and analytic answers indicates that a large number of splashes is needed (easy to change in the code).

Implementation: StoneThrowingF(x).py

```
#  StoneThrowingF(x).py: Monte-Carlo integration via stone throwing

import random
from visual.graph import *

N        = 100   # points to plot the function
graph    = display(width=600,height=600,title='Integration by Stone Throwing')
xsinx    = curve(x=list(range(0,N)), color=color.yellow, radius=0.5)
pts      = label(pos=(-30, -50), text='x ->', box=0)
pts      = label(pos=(-60, -60), text='Number Stones = ', box=0)      # labels
pts2     = label(pos=(-30, -60), box=0)
```

```
inside   = label(pos=(30,-60), text='Number Splashes = ', box=0)
inside2  = label(pos=(60,-60), box=0)
arealbl  = label(pos=(-65,60), text='Computed A = ', box=0)
arealbl2 = label(pos=(-35,60), box=0)
areanal  = label(pos=(30,60), text='Analytic A = ', box=0)
zero     = label(pos=(-85,-48), text='0', box=0)
fofx     = label(pos=(-85,0), text='f(x)', box=0)
five     = label(pos=(-85,50), text='5', box=0)
twopi    = label(pos=(90,-48), text='2pi',box=0)

def fx (x):  return x*sin(x)*sin(x)      # Function to integrate

def plotfunc():             # Plot function & box
    incr = 2.0*pi/N
    for i in range(0,N):
        xx         = i*incr
        xsinx.x[i] = ((80.0/pi)*xx-80)
        xsinx.y[i] = 20*fx(xx)-50
    box            = curve(pos=[(-80,-50), (-80,50), (80,50)
                   ,(80,-50), (-80,-50)], color=color.white)       # box

plotfunc()                          # area of box = height * width = 5*2pi
j            = 0
Npts         = 1001                    # points generated inside the box
analyt       = pi**2                        # Analytical integral
areanal.text = 'Analytic A = %8.5f'%analyt     # Output analytical integral
genpts       = points(size=2)
for i in range(1,Npts):                    # generates points inside box
    rate(500)                 # slow process down to see point generation
    x = 2.*pi*random.random()                                  # 0=< x <= 2pi
    y = 5.*random.random()                                       # 0=< x <= 5
    xp = x*80./pi-80.
    yp = 20.0*y-50.
    pts2.text = '%4s' %i
    if y <= fx(x):                                                  # splash
           j += 1                                  # increase numbr splashes
           genpts.append(pos=(xp,yp), color=color.cyan)
           inside2.text='%4s'%j                       # Label for splashes
    else:  genpts.append(pos=(xp,yp), color=color.green)
    boxarea = 2.*pi*5.                            # Area of box propto Npts
    area = boxarea*j/(Npts-1)                      # Area of curve propto j
    arealbl2.text = '%8.5f'%area            # write current computed area
```

Exercise

Extend the stone throwing program so that it calculates the definite integral

(15) $$I = 2\int_0^1 x^2 \mathrm{dx}$$

Make sure to throw enough stones so that you obtain three good decimal places of precision. In this case we have an analytic expression with which to compare

(16) $$I = \left|_0^1 \frac{x^3}{3} = 1/3\right.$$

Where's Computational Thinking?

- Using random numbers to do mathematics opens up a whole new way of thinking about math (*stochastic mathematics*). In the present case we can use stone throwing to evaluate the integral of an arbitrary function numerically in a very simple way.
- As we deduce from the very large number of points needed to obtain even middling precision, the technique may be simple but it is not efficient. This is typical of the type of trade off that occurs in computational thinking.
- As is typical in computational thinking, when presented with a new technique we test it using a case where we know the analytic answer. Once we have confidence in the technique we can go ahead and apply it to cases where analytic answers are not available.
- The logical basis of the stone—throwing algorithm of using ratios of areas and numbers is simple and intuitive, but also profound. Calculating π as the ratio of numbers provides a new insight into mathematics (it is not obvious why this should be π, the ratio of a circle's circumference to its diameter).
- The algorithm's use of a rejection technique in which some random numbers are acceptable while other are not, represents a whole new way of using the computer to do calculations. It is by no means the direct evaluation of a function, but it is computational thinking. In addition, using a rejection technique opens new avenues for using random number to simulate or calculate natural processes.
- Being able to see how computer-generated random numbers can somehow be equated with physical events.
- Viewing the mathematical process of integration as equivalent to determining an area numerically.
- Simulating an experiment that can be performed on the computer as well as in the physical world.

Where's the Math?

All of this is math. Determining areas is called integration in calculus. The stone throwing and rejection algorithm is stochastic math, that is, math involving chance.

The project is clearly only an approximate way of evaluating integrals, with the answers obtained getting more precise as more stones are thrown. Understanding just how the error decreases as the number of points being used increases requires some understanding of statistics. We give the result but do not try to derive it.

The way in which π arises from use of the equation of a circle is rather profound.

10. Summary and Conclusions

12 References

[CP] Landau, R.H., M.J. Paez and C.C. Bordeianu, (2008), *A Survey of Computational Physics,* p 289-297, Princeton Univ. Press, Princeton.