Computational Physics: Midterm

Sarah Roberts

October 23, 2018

Monte Carlo Integration. Monte Carlo integration is a numeric integration technique utilizing a set of n random numbers x_i on a domain from a to b to approximate the integral of a function f. The method evaluates sums over $f(x_i)$ for large n and uses these values to approximate the area under f by dividing by the number of samples n. This is is expressed as

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{n}\sum_{i=0}^{n} f(x_i)$$

The table shows the results of this simple Monte Carlo method. It lists n, the integral estimate, and percent error for

$$\int_{a=-1}^{b=1} f(x) = x^2 - 2$$

n	estimate	error
100	-3.389347	1.680415
1 000	-3.328592	0.142230
10 000	-3.337934	0.138022
100 000	-3.332738	0.017850

The following plot shows the relationship between n and percent error for the above function. Not that the x axis is logarithmic.

TODO

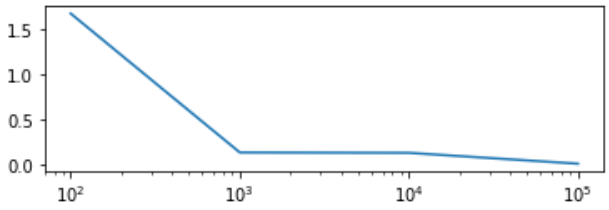

FIGURE 1. This plot show n against percent error for the integral of $f(x) = x^2 - 2$ from $x = -1$ to $x = 1$

ALGORITHM IMPROVEMENTS

Motivation. Clearly, many points are needed to generate an adequate estimation of an integral through the Monte Carlo method. This in turn contributes to a long computation time. There are many ways to increase the accuracy of the integration, and here we will discuss a method that increases accuracy and would easily lend itself to parallelization, which would in turn decrease computation time, when the overhead of threading is appropriate. Note that the code presented here is not parallel, because in Python the overhead is very high and the tested functions are simplistic enough that the computations were relatively quick.

Outline of method. The following pseudo code divides the domain of the integral $S = x \in [a, b]$ into k partitions. The size of the partitions is determined based on the the derivative $f'(x)$. The maximum allowed derivative over a region is set by the variable *thresh*. Minimizing the derivative of the function minimizes that variance over the interval, meaning there is less scatter among the x_i.

This first algorithm defines the domain intervals

```
INPUT: a, b, f, n
// a, b is the domain
// f is our function
// n is the number of points tested in each interval

PARAMETERS: threshold, epsilon
// threshold is how close the derivatives will be between steps
// epsilon is the step size

xValues = []    // array to hold the x values that correspond to divisions
divisionFinished = False    // flag that we are finished dividing the domain
xValues.append(a)              # xValues[0] = a
xValues.append(a + epsilon)    # xValues[1] = a + epsilon

i = 1
do while not divisionFinished
    // define approx value of f'( x[i-1] ) and f'( x[i] )
    d1 = Deriv(func(xValues[i-1]), func(xValues[i-1] + epsilon))
    d2 = Deriv(func(xValues[i]),  func(xValues[i] + epsilon))

    // if the derivatives are close, add x[i] to the xVals list and increment i
    if ( abs(d2 - d1) <= threshold) then
        xValues.append(xValues[i])
        i += 1
        xValues[i] = xValues[i-1]+ epsilon
    // otherwise, adjust x[i] and keep searching
    else
        xValues[i] += epsilon
    end if

    // if x[i] is close to the end of the domain, finish iterating
    if xValues[i] >= b - epsilon
        divisionFinished = True
    end if
end do

// close the domain
xValues.append(b)

// perform numerical integration (using the Monte Carlo Method)
out = 0
do i = 0, len(xValues)-1
    out += integrate(xValues[i], xValues[i+1], func, n)
end do

return out
```

This second algroithm performs Monte Carlo integration

```
INPUT: x1,x2,func,n
// x1, x2 is the domain over which integral will be evaluated
// func if the function being integrated
// n is the number of random points

// initialize integral to zero
integral = 0

// evaluate func(x) for n random numbers in the interval [x1, x2]
do i= 0, n
    integral += func(random.uniform(x1,x2))
end do

// multiply by the range and divide by number of random numbers
return = (x2-x1) * integral / n
```

General Formula. In general, define a series of domains D_i on (a, b) such that $|f'(x_{i-1}) - f'(x_i)| < t$ where t is a tolerance and the interval (a, b) is completely covered by non-overlapping D_i. Apply Monte Carlo integration to each D_i according to

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{n}\sum_{i=0}^{n} f(x_i)$$

where x_i is a random number in D_i, $a = min(x \in D_i)$ and $b = max(x \in D_i)$. Note n is a user-defined parameter.

Error Comparison. The following error analysis was done for the Monte Carlo (MC) approximation of

$$\int_{-2}^{4} g(x)\,dx = \int_{-2}^{4} x^2 + 3x + 1 = 48.0.$$

A summary of methods and their percent error is included below. Note that n is the number of random samples per step.

method	n	estimate	percent error
analytic solution	N/A	48.000000	0.000000
MC on the full domain	100	53.554147	11.57114
MC on the full domain	1 000	45.887656	4.400717
MC on the full domain	10 000	47.784414	0.449138
MC on the full domain	30 000	47.860712	0.290183
MC with 3 equal steps	10 000	47.844436	0.324091
MC with adaptive step	50	48.025826	0.053803
MC with adaptive step	75	48.003386	0.023771
MC with adaptive step	100	47.991797	0.017090
MC with adaptive step	150	47.998690	0.002730

It's expected that the Monte Carlo approximation for $n = 30,000$ on the full domain would be about equal to the approximation given by the sum of three equal domains on the same interval, which is evident in the above chart.

With the adaptive step size and $n = 50$, the percent error is less than on the full interval with $n = 30,000$. Increasing n by only 50 leads to order of magnitude reductions in the percent error, when the adaptive step

size is used. Note, however, that more calculations may be done since n gives the random number per bin created by the adaptive algorithm.

Note that to generate results comparable to the adaptive algorithm with the full domain method, $n = 100,000,000$ gives $\int_{-2}^{4} g(x) \approx 47.993326$ with 0.013905% error. This is comparable to the adaptive method with $n = 100$, which takes significantly less time to run.

Implementation

The included Python code implements both routines described in the "Outline of Method" section. Additionally, it includes the calling functions to generate much of the data included in this write-up. To decrease the run-time of this code set the flag `saveTime = true`. This will skip the most time-intensive steps of the code.