Lab 1

Monte Carlo Integration

Lab Objective: *Implement Monte Carlo integration to estimate integrals. Use Monte Carlo Integration to calculate the integral of the joint normal distribution.*

Some multivariable integrals which are critical in applications are impossible to evaluate symbolically. For example, the integral of the joint normal distribution

$$\int_{\Omega} \frac{1}{\sqrt{(2\pi)^k}} e^{-\frac{\mathbf{x}^T\mathbf{x}}{2}}$$

is ubiquitous in statistics. However, the integrand does not have a symbolic antiderivative. This means we must use numerical methods to evaluate this integral. The standard technique for numerically evaluating multivariable integrals is *Monte Carlo Integration*. In the next lab, we will approximate this integral using a modified version of Monte Carlo Integration. In this lab, we address the basics of Monte Carlo Integration.

Monte Carlo integration is radically different from 1-dimensional techniques like Simpson's rule. Whereas Simpson's rule is purely computational, Monte Carlo integration relies on probability to calculate the integral. Although it converges slowly, Monte Carlo integration is frequently used to evaluate multivariable integrals because the higher-dimensional analogs of methods like Simpson's rule are inefficient.

A Motivating Example

Suppose we want to numerically compute the area of a circle of radius 1. From analytic methods, we know the answer is π. Empirically, we can estimate this quantity by randomly choosing points in a 2×2 square. The percent of points that land in the inscribed circle, times the area of the square, should approximately equal the area of the circle (see Figure 1.1).

We do this in NumPy as follows. First generate 500 random points in the square $[0, 1] \times [0, 1]$.

```
>>> numPoints = 500
>>> points = np.random.rand(2, numPoints)
```

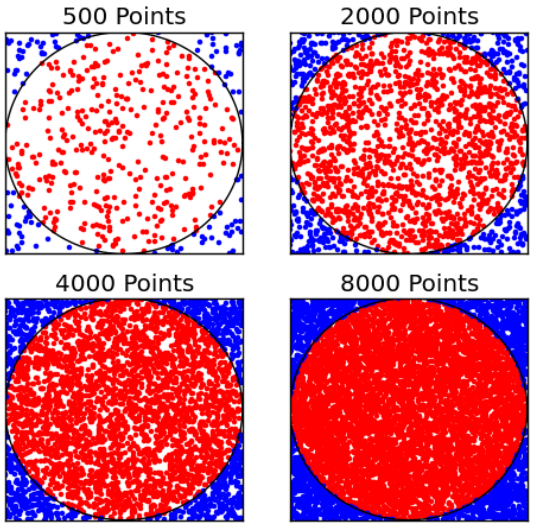

Figure 1.1: Finding the area of a circle using random points

We rescale and shift these points to be uniformly distributed in $[-1,1]\times[-1,1]$.

```
>>> points = points*2-1
```

Next we compute the number of points in the unit circle. The function `np.hypot(a, b)` returns the norm of the vector (a,b) where a and b are the x- and y-components, respectively.

```
>>> # Create a mask of points in the circle
>>> circleMask = np.hypot(points[0,:], points[1,:]) <= 1
>>> # Count how many there are
>>> numInCircle = np.count_nonzero(circleMask)
```

Finally, we approximate the area.

```
>>> # Area is approximately (area of the square)*(num points in circle)/(total ↩
    num points)
>>> 4.*numInCircle/numPoints
3.024
```

This differs from π by about 0.117.

Problem 1. Write a function that estimates the volume of the unit sphere. Your function should have a keyword argument `numPoints` that defaults to 10^5. Your function should draw `numPoints` points uniformly from $[-1,1]\times[-1,1]\times[-1,1]$ to make your estimate. Your answer should be approximately 4.189.

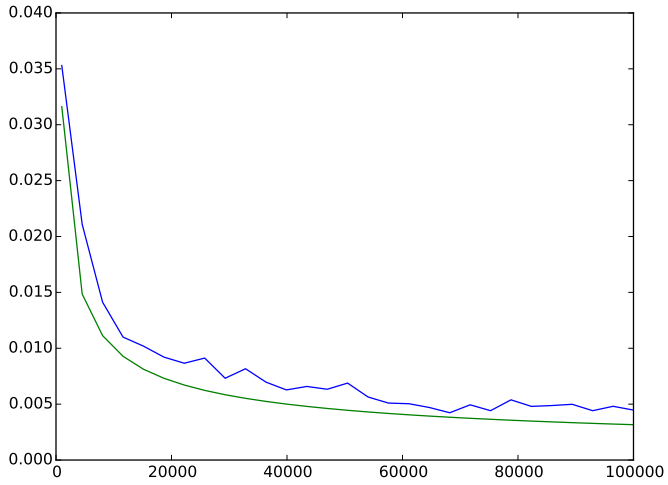

Figure 1.2: The Monte Carlo integration method was used to compute the area of a circle of radius 1. The blue line plots the average error in 100 runs of the Monte Carlo method on N sample points, where N appears on the horizontal axis. The green line is a plot of $1/\sqrt{N}$.

We analyze the error of the Monte Carlo method by repeating this experiment for many values of `numPoints` and plotting the errors. The result is the blue line in Figure 1.2. The error appears to be proportional to $1/\sqrt{N}$ where $N =$ `numPoints` (the green line in Figure 1.2). This means that to divide the error by 10, we must sample *100 times* more points.

This is a slow convergence rate, but it is independent of the number of dimensions of the problem. This dimension independence is what makes the Monte Carlo method useful for multivariable integrals.

Monte Carlo Integration

You can calculate the area of the unit circle with the following integration problem:

$$\text{Area of unit circle } = \int_{[-1,1]\times[-1,1]} f(x,y)dA$$

where

$$f(x,y) = \begin{cases} 1 & \text{if } (x,y) \text{ is in the unit circle} \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

This method essentially draws a box around the function to estimate the integral. This works fine if we know the bounds of the function. However, if we don't know the bounds of the function we don't know the dimensions of the box much be to encapsulate the function.

Luckily, we can still use a random-points method as above to approximate any integral, even if we don't know the bounds. Suppose we wish to evaluate

$$\int_\Omega f(x)\,dV.$$

We can approximate this integral using the formula

$$\int_\Omega f(x)\,dV \approx V(\Omega)\frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{1.2}$$

where x_i are uniformly distributed random vectors in Ω and $V(\Omega)$ is the volume of Ω. This is the formula for Monte Carlo integration.

In our example, Ω was the box $[-1,1]\times[-1,1]$ and f was the function defined in (1.1). Then $\sum_{i=1}^{N} f(x_i)$ is the number of points in the unit circle, N is the total number of points, and (1.2) is the same as the formula we derived previously.

The intuition behind (1.2) is that $\frac{1}{N}\sum_{i=1}^{N} f(x_i)$ approximates the average value of f on Ω. We multiply this (approximate) average value by the volume of Ω to get the (approximate) integral of f on Ω.

As a 1-dimensional example consider the integral

$$\int_0^1 x\,dx \approx (1-0)\frac{1}{N}\sum_{i=1}^{N} x_i = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

The integral on the left-hand-side is $1/2$. In the approximation on the right-hand-side, x_i is drawn from a uniform distribution on $[0,1]$. The average of N such draws will converge to $1/2$.

Problem 2. Monte Carlo Integration is particularly useful when trying to approximate integrals that would be difficult to calculate otherwise. Write a function that approximates the following integral:

$$\int_1^5 \left|\sin(10x)\cos(10x)+\sqrt{x}\sin(3x)\right| dx$$

Your function should accept a keyword argument `numPoints` that defaults to 10^5. Your answer should be approximately 4.502.

Problem 3. Implement Monte Carlo integration with the following function. Your implementation should be robust enough to integrate any function $f:\mathbb{R}^n\to\mathbb{R}$ over any interval in $\mathbb{R}^n$. Your implementation should run the Monte Carlo algorithm several times and return the average of those runs. Test your function by recalculating Problem 1 and 2.

```
def mc_int(f, mins, maxs, numPoints=500, numIters=100):
    """Use Monte Carlo integration to approximate the integral of f
    on the box defined by mins and maxs.

    Inputs:
        f (function) - The function to integrate. This function should
            accept a 1-D NumPy array as input.
        mins (1-D np.ndarray) - Minimum bounds on integration.
        maxs (1-D np.ndarray) - Maximum bounds on integration.
```

```
        numPoints (int, optional) - The number of points to sample in
            the Monte Carlo method. Defaults to 500.
        numIters (int, optional) - An integer specifying the number of
            times to run the Monte Carlo algorithm. Defaults to 100.

    Returns:
        estimate (int) - The average of 'numIters' runs of the
            Monte Carlo algorithm.

    Example:
        >>> f = lambda x: np.hypot(x[0], x[1]) <= 1
        >>> # Integral over the square [-1,1] x [-1,1]. Should be pi.
        >>> mc_int(f, np.array([-1,-1]), np.array([1,1]))
        3.1290400000000007
    """
```

Hints:

1. To create a random array of points on which to evaluate `f`, first create a random array of points in $[0,1] \times \ldots \times [0,1]$. Then multiply this array by a vector of the lengths of the sides to stretch it the right amount in each direction. Finally, add the appropriate vector to shift the points to the right location.

2. You can evaluate `f` on an array of points using `np.apply_along_axis()` as follows:

```
# to evaluate the function f using the rows of vecs as input
>>> f = lambda x: la.norm(x)
>>> vecs = np.array([[1,1,1],[0,2,1],[0.5,0.5,0.5],[1,0,1]])
>>> np.apply_along_axis(f,1,vecs)
array([ 1.73205081,  2.23606798,  0.8660254 ,  1.41421356])
```

   In this example, we chose the axis parameter to be 1 to evaluate the rows of the matrix. If you would like a refresher on axes, see Lab **??**.

Problem 4. The exact value of the integral of

$$f(x,y,z,w) = \sin(x)y^5 - y^3 + zw + yz^3$$

on $[-1,1] \times [-1,1] \times [-1,1] \times [-1,1]$ is 0. Run the function `mc_int()` you wrote in Problem 3 on f with 100, 1000, and 10000 sample points. Use the default value of 100 iterations for your approximations. Print the errors of your estimates to the terminal.

A Caution

You can run into trouble if you try to use Monte Carlo integration on an integral that does not converge. For example, we may attempt to evaluate

$$\int_0^1 \frac{1}{x}$$

with Monte Carlo integragtion using the following code.

```
>>> k = 5000
>>> np.mean(1/np.random.rand(k,1))
21.237332864358656
```

Since this code returns a finite value, we could assume that this integral has a finite value. In fact, the integral is infinite. We could discover this empirically by using larger and larger values of k, and noting that Monte Carlo integration returns larger and larger values.