Homework 6

November 27, 2020

---

1 Problem 1

We will follow the discussion from Section **6.14.14**. Namely, we will calculate the quantity I_p, which is the incomplete Beta function. By definition, we know that this function is an integral of the form

$$I_x(a,b) = \frac{\int_{t=0}^{x} t^{a-1}(1-t)^{b-1}\mathrm{d}t}{B(a,b)} \ ,$$

as stated on page **270**. To this end, we will substitute the given values from **1**. Before computing the integral in the numerator of this expression, observe that the term in the denominator $B(a,b)$ can be expressed with the following formula with a term from the Gamma Function, in which

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \ .$$

From here, we will relate terms from the general formula of the Incomplete Beta Function to our purposes. In particular, $k = 7$ and $n = 22$. This gives

$$\begin{aligned} &\Rightarrow \mathbf{P}(>7) = I_p(8,15) \\ &= \frac{\int_{t=0}^{p} t^{7}(1-t)^{14}\mathrm{dt}}{B(8,15)} \ . \end{aligned}$$

In Python we can directly compute this quantity, and therefore the probability of interest, with the 'betainc' function. It gives

```
>>> import scipy as sp
>>> special.betainc(8,15,1/sp.pi)
0.3999670617689579
```

So we have obtained the desired probability, which completes **1**.

2 Problem 2

We will calculate the given double integral numerically by making use of our previous work in HW 3. Namely, we know that the minor and major axes of the given ellipse approximately hit our coordinate axes at 45 degree angles, so the x and y values of the bounds for the integral that we will use will be approximately the same. From HW 3, we used $x \approx \pm 0.8$, so we will also use $y \approx \pm 0.8$. With this range of x and y we will now approximately impose a rectangle that our Python function will use. In this function, we will try to approximate the rectangle over which we will perform the integral with these x and y values.

But before proceeding, we have to define our function $f(x, y)$ more precisely. Specifically, we must redefine $f(x, y)$ to take on values inside the ellipse; on the other hand, once we are integrating $f(x, y)$ over regions of the rectangle that are outside of the ellipse, we will set $f(x, y) \equiv 0$. We will implement this definition of $f(x, y)$ with

```
>>> def f(x,y):
...     if (5*x**2 - 6*x*y + 5*y**2 <=2): return np.exp(-x*np.sin(y))
...     else: return 0
...
```

This demonstrates one possible way through which we could define $f(x, y)$ so that it vanishes outside of the given ellipse. Defining f in a similar way and directly applying the 'mcquad' method gives that

```
>>> result, error = mcquad(\
...     lambda x: np.exp(-x[0]*np.sin(x[1])) if (5*x[0]**2 - 6 * x[0]*x[1] + 5 * x[1]**2 <2) else 0.,
...     npoints = 400000, xl = [-0.8,-0.8] , xu = [0.8,0.8])
>>> result
1.4495172179181568
>>> error
0.0018543127202367107
```

So with this code, we have captured the idea that I have mentioned; the last output above for 'error' is the estimated error that we wanted after performing the Monte Carlo Integration. Also, observe from the error that I have listed that we have approximately obtained what was listed. For this choice of N that is given in the code above (around $N = 400,000$), the actual error of this computation of the given integral is about $|1.449026\cdots - \text{result}| = |1.449026\cdots - 1.44855842377314\cdots| \approx 0.00046757622256854425 \approx 5 \times 10^{-3}$.

But for the estimated error to be around 0.01, we get that

```
>>> result, error = mcquad(\
...     lambda x: np.exp(-x[0]*np.sin(x[1])) if (5*x[0]**2 - 6 * x[0]*x[1] + 5 * x[1]**2 <2) else 0.,
...     npoints = 1300000, xl = [-0.8,-0.8] , xu = [0.8,0.8])
>>> error
0.0010283398900496332
>>> result
1.450005968126791
>>> result - 1.449026
0.0009799681267910199
```

This code provides the same exact results that I have given previously except for the relative error being 0.001 on the dot. We have given the 2 errors that we wanted and the value of N, which completes **2**.