COMP 5700/6700/6706 Software Process
Fall 2015

Outcomes

Primary: To give you exposure to Test Driven Development

Background

We have been constructing code in class that implements Student's t distribution based on the following equations:

$$p(t,n,tails) = \begin{cases} tails = 1 & \left(\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\right)\int_{-\infty}^{t}\left(1+\frac{u^2}{n}\right)^{-\left(\frac{n+1}{2}\right)}du \\ tails = 2 & \left(\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\right)\int_{-t}^{t}\left(1+\frac{u^2}{n}\right)^{-\left(\frac{n+1}{2}\right)}du \end{cases}$$

Working in collaboration with our customer, we packaged our code in a class named "`TCurve`" having the following methods:

```
__init__(n)
p(t, tails)
gamma(x)
LHP(n)
f(u, n)
integrate(f, n, lowBound, highBound)
```

The accompanying files, `TCurve.py` ***and*** `TCurveTest.py`, contain code and corresponding tests for all the methods except `integrate`.

Assignment Use TDD to construct `integrate`. `integrate` calculates the area under the curve for the equation, $\left(1+\frac{u^2}{n}\right)^{-\left(\frac{n+1}{2}\right)}$. (Note that `f` implements the equation for you. You do not need to construct it.). Please implement `integrate` with Simpson's Rule for numerical integration. The algorithm is given in Python-like pseudocode below:

```
 1 epsilon = 0.001
 2 simpsonOld = 0
 3 simpsonNew = epsilon
 4 s = a value of your choice (a good starting value is 4)
 5 while (abs((simpsonNew - simpsonOld ) / simpsonNew) > epsilon):
 6     simpsonOld = simpsonNew
 7     w = (lowBound - highBound) / s
 8     simpsonNew = (w/3) * (f(lowBound) + 4f(lowBound + w)
                          + 2f(lowBound + 2w) + 4f(lowBound + 3w)
                          + 2f(lowBound + 4w)
                       ... + 4f(highBound-w) + f(highBound))
 9     s = s * 2
10 return simpsonNew
```

The algorithm divides the curve into `s` slices and calculates the area of each slice based on a quadratic interpolation of points. It then doubles the number of slices and recalculates the area. If the two calculations are not close enough in value -- where "close enough" is defined by the value of epsilon -- the number of slices is doubled and the area recalculated. This proceeds until the two values are within the prescribed error tolerance.

It isn't important to understand the mathematics underlying how the area is calculated. But, it is important to understand statement 8. The number of terms to tally in statement 8 depends on how many times the curved is sliced, with each term having a coefficient of 1, 2, or 4. To illustrate this, suppose `lowbound`=0, `highBound`=16 and `s`=4. Statement 8 would be calculated as

```
simpsonNew = (4/3) * (f(0) + 4f(4) + 2f(8) + 4f(12) + f(16))
```

If we were to double `s` to 8, Statement 8 would be calculated as

```
simpsonNew = (2/3) * (f(0) + 4f(2) + 2f(4) + 4f(6) + 2f(8) +
                      4f(10) + 2f(12) + 4f(14) + f(16))
```

Note that the number of terms will be `s`+1. The beginning and final term have a coefficient of 1. The second, fourth, sixth, etc. term have a coefficient of 4. The third, fifth, seven, etc. term have a coefficient of 2.

Hints

- You'll see from the `p` method how `integrate` should be invoked. Notice that one of the parameters to `integrate` is a function.
- `integrate` may be tricky to test if you start by using $\left(1+\frac{u^2}{n}\right)^{-\left(\frac{n+1}{2}\right)}$ in your initial TDD cycles. Consider using a simpler integral, one that you know the result. In case you've forgotten your calculus, below are some sample functions and the result you should expect when integrating from 0 to t:

function	value of t	value of n	expected result
`def f(u, n):` `  return u`	1.0	doesn't matter	1/2
`def f(u, n):` `  return u**2`	1.0	doesn't matter	1/3
`def f(u, n):` `  return u**6`	1.0	doesn't matter	1/7
`def f(u, n):` `  return u**100`	1.0	doesn't matter	1/101

Guidelines

- This effort is to be done on an individual basis. The code should be original. DO NOT use code from the Internet.
- Upload two separate files to Canvas: `TCurve.py` and `TCurveTest.py`

Extra Information

Selected values of the t-distribution are shown in the table below:

	p(a)=	0.6	0.7	0.85	0.9	0.95	0.975	0.99	0.995
	p(a/2)=	0.2	0.4	0.7	0.8	0.9	0.95	0.98	0.99
degrees of freedom	1	0.3249	0.7265	1.9626	3.0777	6.3138	12.7062	31.8205	63.6567
	2	0.2887	0.6172	1.3862	1.8856	2.9200	4.3027	6.9646	9.9248
	3	0.2767	0.5844	1.2498	1.6377	2.3534	3.1824	4.5407	5.8409
	4	0.2707	0.5686	1.1896	1.5332	2.1318	2.7764	3.7469	4.6041
	5	0.2672	0.5594	1.1558	1.4759	2.0150	2.5706	3.3649	4.0321
	6	0.2648	0.5534	1.1342	1.4398	1.9432	2.4469	3.1427	3.7074
	7	0.2632	0.5491	1.1192	1.4149	1.8946	2.3646	2.9980	3.4995
	8	0.2619	0.5459	1.1081	1.3968	1.8595	2.3060	2.8965	3.3554
	9	0.2610	0.5435	1.0997	1.3830	1.8331	2.2622	2.8214	3.2498
	10	0.2602	0.5415	1.0931	1.3722	1.8125	2.2281	2.7638	3.1693
	15	0.2579	0.5357	1.0735	1.3406	1.7531	2.1314	2.6025	2.9467
	20	0.2567	0.5329	1.0640	1.3253	1.7247	2.0860	2.5280	2.8453
	30	0.2556	0.5300	1.0547	1.3104	1.6973	2.0423	2.4573	2.7500

The table gives the one- and two-tailed probabilities for value of t at various degrees of freedom. For example, suppose we want to find the probability where t=1.1342 and degrees of freedom=6. Go along the row designed as "6" and find 1.1342. The top of column where 1.1342 was found gives the

probabilities. “p(a)” indicates the probability for one tail and “p(a/2)” indicates the probability for two tails. Given this, the one-tailed probability is 0.85 and the two-tailed probability is 0.7.