COMP 5700/5703/6700/6706 Software Process

Background

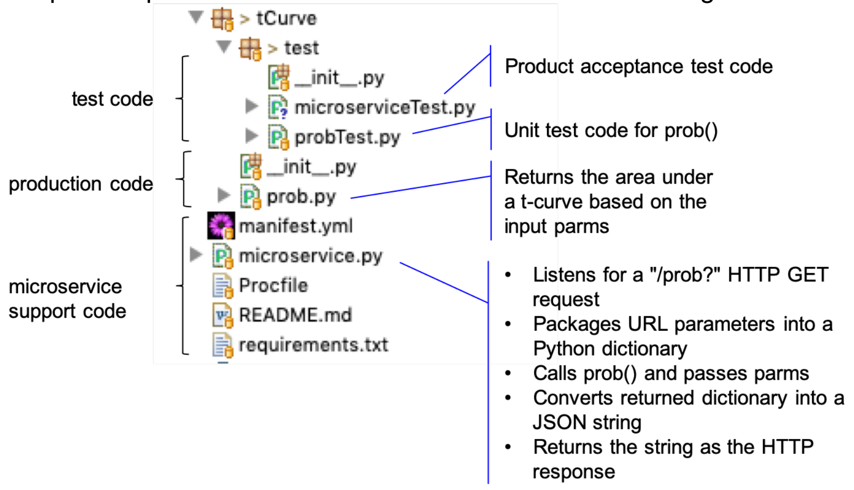
We have been constructing code that implements the Student’s t distribution based on the following equations:

$$prob = \begin{cases} if\ tails == 1\text{:}\ \ 0.5 + \left(\dfrac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\right)\displaystyle\int_0^t \left(1+\frac{u^2}{n}\right)^{-\left(\frac{n+1}{2}\right)} du \\ if\ tails == 2\text{:}\ \ 2.0 \times \left(\dfrac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\right)\displaystyle\int_0^t \left(1+\frac{u^2}{n}\right)^{-\left(\frac{n+1}{2}\right)} du \end{cases}$$

At the request of our customer, we will package our code as a microservice that responds to an HTTP GET request that can be encoded as a URL having the following format:

$$\underbrace{\text{http}}_{\text{microservice interchange protocol}}\text{://}\underbrace{\text{www.microservice.net:5000}}_{\text{name of the server hosting the microservice and port on which the microservice responds}}\text{/}\underbrace{\text{prob}}_{\text{name of the microservice}}\text{?}\underbrace{\text{n=5\&t=1.587\&tails=1}}_{\text{parameters passed to the microservice}}$$

The product portion of the deliverable is to have the following file structure:



```
▼ > tCurve
    ▼ > test
          __init__.py
        ▶ microserviceTest.py
        ▶ probTest.py
      __init__.py
    ▶ prob.py
  manifest.yml
▶ microservice.py
  Procfile
  README.md
  requirements.txt
```

test code: test/__init__.py, microserviceTest.py, probTest.py

production code: __init__.py, prob.py

microservice support code: manifest.yml, microservice.py, Procfile, README.md, requirements.txt

microserviceTest.py: Product acceptance test code

probTest.py: Unit test code for prob()

prob.py: Returns the area under a t-curve based on the input parms

microservice.py:

- Listens for a "/prob?" HTTP GET request
- Packages URL parameters into a Python dictionary
- Calls prob() and passes parms
- Converts returned dictionary into a JSON string
- Returns the string as the HTTP response

The main functionality resides in the *prob.py* file. Specifically, the file will contain an outward facing function, `prod(parmDictionary)`, where `parmDictionary` is a Python dictionary containing keys and respective values for "n", "t", and "tails".

In designing `prob`, we identified the need for additional functions that are internal (i.e., inward facing or private):

```
_gamma(x)
_calculateConstant(n)
_f(u, n)
_integrate(t, n, _f)
```

The project files are available from GitHubClassroom. The project contains code and corresponding tests for all the methods except `_integrate`.

Assignment

Use TDD to construct the method that performs the integration, `_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` is provided to you. You do not need to construct it.).

The definition of the integration function is

```
_integrate(t, n, _f)
```

where t and n correspond to the variables by the same names used earlier, and _f refers to the function.

Implement `_integrate` using Simpson's Rule for numerical integration. The algorithm is given in Python-like pseudocode below:

```
 1 epsilon = suitably small value, such as 0.001
 2 simpsonOld = 0.0
 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 = (highBound - lowBound) / s
 8     simpsonNew = (w/3) * (f(lowBound, n) + 4f(lowBound + w, n)
                          + 2f(lowBound + 2w, n) + 4f(lowBound + 3w, n)
                          + 2f(lowBound + 4w, n)
                      ... + 4f(highBound-w, n) + f(highBound, n))
 9     s = s * 2
10 return simpsonNew
```

`Note:  epsilon may have to be adjusted to provide the desired amount of accuracy`

The algorithm divides the curve into `s` slices and calculates the area of each slice based on a quadratic interpolation of points along the top of the curve. 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 in the pseudocode above. The number of terms to tally in statement 8 depends on how many times the curve is sliced, with each term having a coefficient of 1, 2, or 4. To illustrate this, suppose `lowbound`=0, `highBound`=16, and `s`=4. The value of `n` is unimportant for purposes of this discussion. The value of Statement 8 would be calculated as

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

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

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

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

Guidelines

- This effort is to be done on an individual basis. The code should be original. DO NOT use code from the Internet, friends, or previous offerings of this course.
- The following steps are needed **before** you start work on `_integrate`:
  - Start Eclipse and install the TDD plugin.
    - Provide the installation URL as follows:
      - OSX and Linux: Help --> Install New Software ...
      - Windows: Window --> Install New Software
    - Click the "add..." button, provide the following information, and click the "OK" button. Provide the information below:
      - Name: TDD Tool
      - Location: http://www.auburn.edu/~umphrda/TDD
    - Select the box "TDD Tool", click "Next", and follow the installation wizard instructions. Allow Eclipse to restart.
  - Install Python Flask
    - Go to https://palletsprojects.com/p/flask/
    - Select the "documentation" link for installation information.
      - The installation instructions imply that Flask should be installed in a virtual python environment. Do so if you feel comfortable with how virtual python environments work with Eclipse, otherwise skip the virtual environment installation and go directly to the "Install Flask" instructions.
  - Pull the project from GitHub Classroom
    - Accept the GitHub Classroom invitation given on Canvas
    - Once your repository is displayed, click the green "Clone or download" button and copy your repository URL to your clipboard.
    - From Eclipse:
      - Create a project and local Git repository. To do this...
        - Create a new PyDev project. Give it any name you wish.
        - Right click the created project in the PyDev Package Explorer, then Team --> Share
        - Check the box, "Use or create repository in parent folder of project", click on the line that shows the project name, click the "Create Repository" button, then click "Finish". It is imperative that your local repository be in your parent folder, else the TDD plugin will not work properly.
      - Pull starter code from your GitHub Classroom repository. To do this ...
        - Right click your project in the PyDev Package Explorer, then Team --> Pull...
        - Paste your GitHub Classroom URL into the text box labeled "URI: ". Provide your GitHub username and password. Click "Next"
        - Click the "New Remote" button, verify the resulting screen, click "Finish"
        - Position your cursor in the "Reference:" box and enter space. Double click on "master [branch]"
        - Set "When pulling:" to "Merge"
        - Select the box, "Configure upstream for push and pull"
        - Click the "Finish" button.
- Please employ TDD as you write code for the `_integrate` function. Demonstrate this by using the red signal-light button to indicate you are in the red-light phase of the TDD cycle and by using the green signal-light button to signal you are in the green-light phase. Use the blue signal-light button whenever you have successfully completed a green-light phase and need to clean up your code. Use the yellow signal-light button if you run a supplemental confidence-building test case, meaning, a new test case you expect to pass

but are running it anyway to assure yourself your implementation works.
– Place your `_integrate` function in file prob.py.
– Incorporate your unit tests for `_integrate` at the end of file probTest.py.
– Push our completed project to GitHub Classroom.

Grading Rubric

TDD		
	git log shows evidence of TDD. Examples: • Red-light tests focus on test construction. • Green-light tests focus on production code. • _integrate is built incrementally. • _integrate unit tests are present.	+15
	git log shows some evidence of TDD. Examples: • Red-light tests include limited amounts of production code. • Production code exceeds the scope of respective red light tests. • little evidence that _integrate is built incrementally.	+10
	git log shows little to no evidence of TDD • Little-to-no red-light/green-light commits. • _integrate appears to be built in test-last fashion.	+0
	Code is not available	+0
Acceptance Tests		
	Passes all acceptance tests	+15
	Fails one or more acceptance tests	+0
	Code is not available	+0
Algorithm		
	_integrate() fully implements the specified algorithm: • Simpson's Rule <u>and</u> • Convergent approach	+10
	_integrate() produces a correct result, but partially implements the algorithm. Examples: • Implements Simpson's rule, but doesn't adjust the number of slices (e.g., uses a fixed number of slices) • Uses some other method to calculate the area of each slice.	+5
	_integrate() is not functional, is missing, or is obviously incomplete.	+0
Delivery		
	The submission conforms to instructions and can be tested without modification.	+10
	The submission was pushed to GitHub but must be modified to test	+5
	The submission was not pushed to GitHub.	+0

score = TDD + Acceptance Tests + Algorithm + Delivery = max(50)

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.