5 — PHY 494: Homework assignment (40 points total)

Due Saturday, Feb 20, 2016, 5pm.

Submission is now to your **private GitHub repository**. Follow the link provided to you by the instructor in order for the repository to be set up: It will have the name *ASU-CompMethodsPhysics-PHY494/assignments-2016*-YourGitHubUsername and will only be visible to you and the instructor/TA. Follow the instructions below to submit this (and all future) homework.

You should have already git-cloned your assignment repository.[1] Enter the repository and run the script `scripts/update.sh` (replace *YourGitHubUsername* with your GitHub username):

```
cd assignments-2016-YourGitHubUsername
bash ./scripts/update.sh
```

It should create three subdirectories[2] `assignment_05/Submission`, `assignment_05/Grade`, and `assignment_05/Work` and also pull in the PDF of the assignment and an additional file.[3]

To submit your assignment, commit and push a PDF, text file or Jupyter notebook inside the `assignment_05/Submission` directory and **name it `hw05.ipynb` (or `hw05.pdf`)**. *Commit any other additional files exactly as required in the problems.*

Failure to adhere to the following requirements may lead to homework being returned ungraded with 0 points for the problem.

- Homeworks must be legible and intelligible (write complete English sentences when you are asked to explain or describe).
- If you are required to submit code, it must run without errors under Python 3 with the anaconda distribution.

[1] See last homework; basically

```
repo="assignments-2016-YourGitHubUsername"
git clone https://github.com/ASU-CompMethodsPhysics-PHY494/${repo}.git
```

[2] If the script fails, file an issue in the Issue Tracker for PHY494-assignments-skeleton and just create the directories manually.

[3] When you run `bash ./scripts/update.sh` you should see a new file `assignment_05/Submission/test_calculus.py` being created. If this is *not* the case, run the following commands first to manually update (fixes a bug):

```
git pull \
  https://github.com/ASU-CompMethodsPhysics-PHY494/PHY494-assignments-skeleton \
  master
bash ./scripts/update.sh
```

- If you are required to submit data then the data files must be formatted exactly as required to be machine parseable.

This assignment contains **bonus problems**. A bonus problem is optional. If you do it you get additional points that count towards this homework's total, although you can't get more than the maximum number of points. If you don't do it you can still get full points. Bonus problems and bonus points are indicated with an asterisk "*".

5.1 Your own `calculus` module (10 points)

At the end of the lecture on integration (see 07_integration.ipynb) you started creating your own `calculus` module. Complete this problem as part of this homework assignment:

Create the calculus module (a file named `calculus.py`).[4] At a minimum it should contain the integration functions `trapezoid()` and `simpson()` (numpy versions preferred) and the differentiation functions `D_fd()` and `D_cd()`. You can add more functions if you like.

Make sure that when you are inside the `assignment_05/Submission` directory that you can execute the following Python commands (e.g., start `ipython` or a notebook):

```
import numpy as np
import calculus

# testing integration
print(calculus.trapezoidal(np.cos, -10*np.pi, np.pi/2, 1001))
print(calculus.simpson(np.cos, -10*np.pi, np.pi/2, 1001))

# testing differentiation
t_values = np.array([0.1, 1, 100], dtype=np.float64)
print(calculus.D_fd(np.cos, t_values, 1e-6))
print(calculus.D_cd(np.cos, t_values, 1e-6))
```

and that the functions produce the correct values.

The `assignment_05/Submission` directory will also contain the file `test_calculus.py`, provided by the instructor. This file contains the tests that will be used to check your answer.[5] You can run the tests yourself with the command (inside the same directory)

```
nosetests -v test_calculus.py
```

You should get *OK* at the end and no tests with status *FAILED* or *ERROR*.

[4] Reminder: the file should be inside the `assignment_05/Submission` directory and must be committed as part of the homework.

[5] `test_calculus.py` contains so-called *unit tests*; professional code comes with hundreds of such tests, which are written together with the actual code. They run the code and compare to known answers and raise an error if computed value does not agree with the expected one. Writing tests during development is, together with using version control, the most important software engineering technique to achieve and maintain high quality code.

5.2 Theoretical error analysis of integration algorithms (15 points)

Derive the approximate dependence of the relative integration error ϵ on the number of intervals N.[6]

(a) Estimate the *approximation error* of the *trapezoid rule* (for the interval $[x_i, x_i + h]$)

$$I_i(h) = \int_{x_i}^{x_i+h} f(x)\,dx \approx \frac{h}{2}(f(x_i + h) + f(x_i)) \tag{1}$$

by following these steps:

(i) Expand $f(x)$ as a Taylor series around the interval midpoint $\eta_i = x_i + \frac{h}{2}$. **[2 points]**

(You may check your answer by calculating in particular the two interval endpoints: $f(x_i) = \sum_{n=0}^{\infty} \left(-\frac{h}{2}\right)^n \frac{1}{n!} f^{(n)}(x_i + \frac{h}{2})$ and $f(x_i + h) = \sum_{n=0}^{\infty} \left(\frac{h}{2}\right)^n \frac{1}{n!} f^{(n)}(x_i + \frac{h}{2})$.)

(ii) Use the Taylor expansion in the trapezoid rule Eq. 1, simplify as much as possible, and show that in the resulting expansion of $I_i(h)$ odd orders of $f^{(n)}$ cancel. **[2 points]**

(iii) Estimate the absolute algorithmic error of the interval, $\mathcal{E}_i$, (crudely!) as the first term in the expansion of $I_i(h)$ that is higher than linear in h. The total absolute error $\mathcal{E}$ is then the interval error $\mathcal{E}_i$ multiplied with the number of intervals N. The relative error is estimated as

$$\epsilon = \frac{\mathcal{E}}{f}. \tag{2}$$

Show that for the trapezoidal rule the relative error is on the order of

$$\epsilon_{\text{app}} = \frac{(b-a)^3}{N^2} \frac{f^{(2)}}{f} \tag{3}$$

(where $f^{(2)}$ and f indicate dependence on the function itself and its second derivative). **[3 points]**

(b) The *round-off error* will be modelled with a random-walk as

$$\epsilon_{\text{ro}} = \sqrt{N}\epsilon_m \tag{4}$$

and the total error is

$$\epsilon_{\text{tot}} = \epsilon_{\text{ro}} + \epsilon_{\text{app}} \tag{5}$$

[6] See *Computational Physics*, **5.11** for background.

where ϵ_{app} is given by Eq. 3. Determine N so that ϵ_{tot} (Eq. 5) is minimized for double precision ($\epsilon_m = 10^{-15}$).

You may assume that this is the case when round-off and approximation error are the same. You may also set the function size and length scale to 1, i.e.,

$$\frac{f^{(n)}}{f} = 1 \tag{6}$$

$$b - a = 1. \tag{7}$$

Also determine the total error for this optimum N. **[4 points]**

(c) BONUS: Determine the optimum N and ϵ_{tot} (as in (b)) for the trapezoidal rule in *single precision*. **[bonus +2*]**

(d) For *Simpson's rule*, the algorithmic relative error is on the order of

$$\epsilon_{\text{app}} = \frac{(b-a)^5}{N^4}\frac{f^{(4)}}{f} \tag{8}$$

Using the same approach as in (b), determine the optimum N and ϵ_{ro} for Simpson's rule (in double precision). **[4 points]**

(e) BONUS: Determine the optimum N and ϵ_{tot} (as in (d)) for Simpson's rule in *single precision*. **[bonus +2*]**

5.3 Numerical error analysis of integration algorithms (15 points)

In an experiment we measured the number of particles that entered a counter per unit time, $dN(t)/dt$. Assume we recognize that the number of particles decreases exponentially with time:

$$\frac{dN}{dt} = e^{-t}. \tag{9}$$

(a) BONUS: Show analytically that the number of particles after one time unit, $N(t = 1)$, is given by

$$N(1) = \int_0^1 e^{-t} = 1 - e^{-1}. \tag{10}$$

[bonus +1*]

In the following you are going to study the error behavior of three integration algorithms:

Trapezoid rule use your own `calculus.trapezoid()`

Simpson's rule use your own `calculus.simpson()`

Gauss quadrature A more advanced algorithm (see *Computational Physics*, **5.12** for background); use `scipy.integrate.fixed_quad(f, a, b, n)`. The scipy function behaves a bit differently from the functions that we defined so I suggest you define a new function `gauss_quad()`

```
import scipy.integrate

def gauss_quad(f, a, b, n):
    """Fixed order Gauss-Legendre quadrature.

    Arguments
    ---------
    f : function
        function to be integrated, must be able to work on arrays
    a, b : floats
        interval endpoints
    n : integer
        number of integration points (order)
    """
    integral, junk = scipy.integrate.fixed_quad(f, a, b, n=n)
    return integral
```

This new function takes the same arguments as our functions and just returns the integral in the same way as our functions. If you use it, you can write code that can directly use either your functions from your `calculus` module or `gauss_quad()`.

(b) Compute the relative error

$$\epsilon = \left| \frac{N_{\text{numerical}} - N(1)}{N(1)} \right| \tag{11}$$

where $N(1)$ is the exact result Eq. 10 and $N_{\text{numerical}}$ is the numerical estimate of the same integral using the three algorithms described above.

For the number of intervals $n = \{3, 5, 11, 21, 41, 81, 161, 321, 641, 1281\}$ calculate ϵ for the three integration algorithms and collect the data in a table such as

n	ϵ_{trapez}	$\epsilon_{\text{Simpsons}}$	ϵ_{Gauss}
2	...	...	...
5	...	...	...
...	...	...	...

[8 points]

(c) BONUS: Write out the table from (b) as a file named `Submission/integration_error.dat` with the following format:

```
   2   ...  ...  ...
   5   ...  ...  ...
 ...
1281   ...  ...  ...
```

i.e., four columns, separated by at least one space, and the first column corresponding to n, and the subsequent ones to the errors as in (b). **[bonus +2*]**

(d) Make a log-log plot of the errors of the three algorithms. Plot the three graphs into one plot and add a legend that indicates the algorithm, and add axes labels. Use a range of n values of `np.arange(3, 1000, 2)`. **[7 points]**

Briefly explain if the plots of the *trapezoid rule* and *Simpson's rule* algorithms are consistent with your error estimate in Problem 5.2. **[2 points]**