Homework 2

Computational Astrophysics (ASTR660)

(Due at the start of class on October 13, 2022)

Exercise 1

1. Read the style guide for Python: `https://pep8.org`.

2. Read the Fortran cheat sheet for Python programmers: `https://github.com/wusunlab/fortran-vs-python`.

3. Further reading – Fortran tutorial: `https://fortran-lang.org/en/learn/quickstart/`.

Exercise 2

[Implementing Monte Carlo integration (1 pt)]

In class we wrote a short Python program for evaluating the area of a unit circle using the Monte Carlo method.

(1) Please complete `pi5.py` if you have not done so. Please make sure that (i) your code is reproducible, and (ii) the x and y values are drawn from independent random sequences.

(2) Please compute the relative error $\mathcal{E}$ as a function of the number of points used, N. Please plot $\log_{10}|\mathcal{E}|$ vs. $\log_{10} N$ for the range of $N = 10$ to $N = 10^7$ and see if the error scales as $1/\sqrt{N}$ (overplot the expected scaling using a dashed line).

(3) Modify your code to compute this integral:

$$I = \int_0^2 x^2\, dx.$$

Repeat (2) and verify that the relative error has the expected scaling with N.

Exercise 3

[Error assessment for integration algorithms (1 pt)]

(1) The truncation errors of the midpoint/trapezoid rules for numerical integration scales with the number of subintervals used as $\epsilon_{\rm tr} \sim 1/N^2$. The roundoff error can be expressed as $\epsilon_{\rm ro} \sim \sqrt{N}\epsilon_{\rm m}$, where $\epsilon_{\rm m}$ is the machine precision ($\epsilon_{\rm m} = 10^{-7}$ for 32-bit machines and $\epsilon_{\rm m} = 10^{-15}$ for 64-bit machines). For the midpoint/trapezoid rules, please find the optimal number of subintervals N that could minimize the total error, $\epsilon_{\rm tot} = \epsilon_{\rm tr} + \epsilon_{\rm ro}$ on a 32-bit and 64-bit computer, respectively.

(2) The truncation error of the Simpson's rule for numerical integration scales with N as $\epsilon_{\text{tr}} \sim 1/N^4$. Please repeat (1) and find the optimal N that could minimize the total error.

(3) Combining results from (1) and (2), what is the minimum total error one could obtain for each method for 32-bit and 64-bit computers, respectively?

Exercise 4

[Implementing Trapezoid and Simpson's methods for integration (2 pt)]

In class we wrote a Fortran program to compute the area of a unit circle using the midpoint rule.

(1) Please complete `pi4.f90` if you have not done so. Please compute the relative error as a function of the number of subintervals used, N. Please plot $\log_{10}|\mathcal{E}|$ vs. $\log_{10} N$ for the range of $N = 10$ to $N = 10^7$ and see if the error of the midpoint rule scales as $1/N^2$.

(2) Compile `pi4.f90` with and without the flag `-fdefault-real-8` and observe how the plot in (1) changes. Are the results consistent with what you found in Exercise 3?

(3) Modify the code to use the trapezoid method instead. Repeat (1) and (2) for the trapezoid rule.

(4) Modify the code to use the Simpson's method instead. Repeat (1) and (2) and see if the error of the Simpson's rule scales as $1/N^4$.

Exercise 5

[Error assessment for numerical differentiation (1 pt)]

(1) Because differentiation subtracts two numbers close in value, we will assume that the roundoff error for differentiation is machine precision:

$$f' \approx \frac{f(x+h) - f(x)}{h} \approx \frac{\epsilon_{\text{m}}}{h} \tag{5.1}$$

$$\Rightarrow \epsilon_{\text{ro}} \approx \frac{\epsilon_{\text{m}}}{h}. \tag{5.2}$$

The truncation error of the forward-difference algorithm for numerical differentiation is $\mathcal{O}(h)$:

$$\epsilon_{\text{tr}} \approx \frac{f^{(2)}h}{2}, \tag{5.3}$$

where $f^{(2)}$ is the second derivative of function f.

Assuming f is well behaved and $f^{(2)} \sim 1$, find the optimal h which minimizes the total error, $\epsilon_{\text{tot}} = \epsilon_{\text{tr}} + \epsilon_{\text{ro}}$, for a 64-bit machine.

(2) The truncation error of the central-difference algorithm for numerical differentiation is $\mathcal{O}(h^2)$:

$$\epsilon_{\text{tr}} \approx \frac{f^{(3)}h^2}{24}, \tag{5.4}$$

where $f^{(3)}$ is the third derivative of function f.

Assuming $f^{(3)} \sim 1$, find the optimal h which minimizes the total error, $\epsilon_{\text{tot}} = \epsilon_{\text{tr}} + \epsilon_{\text{ro}}$, for a 64-bit machine.

(3) Summarizing results from (1) and (2), which of the two methods could give smaller total errors for well behaved functions? How do they compare in terms of the step size h required to reach minimum errors?

Exercise 6

[Implementing forward-difference and central-difference methods for differentiation (2 pt)]

(1) Using your preferred programming language, implement the forward- and central-difference algorithms to differentiate the function $\cos(x)$ at $x = 0.1, 1.0$, and 100.

(2) Vary the step size h and record the relative errors $\mathcal{E}$. Reduce h until it equals the machine precision. Plot $\log_{10}|\mathcal{E}|$ vs. $\log_{10} h$. Do the results (slopes, minimum error, optimal h) agree with what you obtained in Exercise 5?