Homework 1

Computational Astrophysics (ASTR660)

(Due at the start of class on September 29, 2022)

Exercise 1

1. Read the tutorial for Linux commands: `https://ryanstutorials.net/linuxtutorial/`.
2. Read the tutorial for bash scripting: `https://linuxconfig.org/bash-scripting-tutorial`.
3. Read the wiki page of the CICA cluster: `https://github.com/nthu-ioa/cluster/wiki`.

Exercise 2

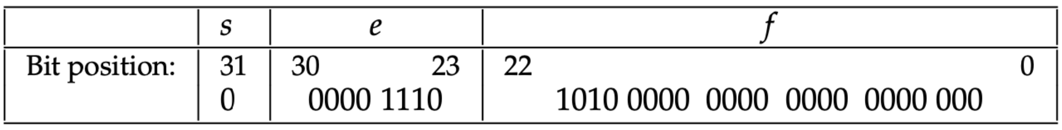
[Estimating memory usage (1 pt)]

Please estimate the total amount of memory required to simulate all stars in the Milky Way Galaxy. Assume that there are 250 billion stars in the Galaxy, and for each star, 10 double-precision variables (including a tag of the star, positions, velocities, and accelerations) need to be stored in order to perform the calculation.

Exercise 3

[Interpreting an IEEE754 floating-point number (1 pt)]

Consider a 32-bit, single-precision floating-point number, A, stored on a machine in the following way:



	s	e	f
Bit position:	31	30 23	22 0
	0	0000 1110	1010 0000 0000 0000 0000 000

(1) What are the values for the sign, exponent, and mantissa of A?

(2) Please determine the full value of A.

Exercise 4

[Writing a bash script (2 pt)]

Please write a bash script to perform the following tasks on the CICA cluster:

(1) Print a sentence to screen to request a path from user
(2) Read the path from the user
(3) Loop over the contents in the given directory and determine whether each content is a file or a directory
(4) Use variables to count the number of files and directories in the given path and print the output to screen

Please use this directory in order to verify your code: `/data/hyang/shared/astro660CompAstro/hw1/`. Please submit your code as well as the screenshot of the outputs.

Exercise 5

[Truncation and roundoff errors (3 pt)]

A classic numerical problem is the summation of a series to evaluate a function. As an example, consider the power series for the exponential of a negative argument:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + ... \tag{5.1}$$

While an infinite series is exact in a mathematical sense, to use it as an algorithm we must stop summing at some point,

$$e^{-x} \approx \sum_{n=0}^{N} \frac{(-x)^n}{n!} \tag{5.2}$$

In this exercise, we will use three different methods to implement Equation 5.2, and identify the sources of errors in the three algorithms. Please submit your code as well as the plots and answers as described below.

(1) **Method 1**: Please write a Python program to perform the summation in Equation 5.2 (write a pseudo code first if needed). Use `numpy.exp(-x)` as the true answer and verify your code with some given x (e.g., $x = 1.0$). Use `matplotlib` to make a plot of your answers as a function of N (from 1 to 100), the number of terms used in the summation. Add suitable labels to the axes of the plots; also add a title to this plot to indicate that this is for $x = 1.0$.

The relative error is defined as (answer - true answer)/true answer. Make another plot of log10(abs(relative error)) as a function of N. At what N does the relative error stop decreasing with increasing N? What causes the change of behavior at this N?

(2) **Method 2:** Method 1 is in fact not a very good algorithm because of two reasons. First, $n!$ and $(-x)^n$ can get very large and cause overflows. Also, powers and factorials are very expensive (time consuming) to evaluate on the computer. Therefore, a better approach is to realize that each term in Equation 5.2 is just the previous one times $(-x)/n$, i.e.,

$$n\text{th term} = \frac{(-x)}{n} \times (n-1)\text{th term} \tag{5.3}$$

Please modify your program in (1) to implement Method 2 (again, write a pseudo code to make sure the logic is correct). Overplot log10(abs(relative error) versus N from both methods for $x = 1.0$ using suitable line colors and linestyles as well as adding the legend.

(3) **Method 3:** One could evaluate e^{-x} using yet another algorithm:

$$e^x \approx \sum_{n=0}^{N} \frac{x^n}{n!}, \tag{5.4}$$

and $e^{-x} = 1/e^x$.

Again, implement Method 3 in your script and overplot log10(abs(relative error) versus N from all three methods for $x = 1.0$. Repeat the analyses for $x = 10.0$ and $x = 100.0$. Which of the three methods is the most precise and robust algorithm? Explain why Methods 1 and 2 fail to converge when x is large.