3 — PHY 494: Homework assignment (46 points total)

Due Thursday, Feb 4, 2016, 1:30pm.
Submit a PDF or text file through Blackboard (name it `lastname_firstname_hw3.pdf`). Homeworks must be legible or may otherwise be returned ungraded with 0 points.

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 "*".

3.1 Representation of numbers in the computer (10 points)

(a) Convert the decimal number -2.65625_{10} to binary. What is the binary representation and how many bits do you need? **[2 points]**

(b) Convert the decimal number $+0.8_{10}$ to binary. What is the binary representation and how many bits do you need? **[2 points]**

(c) What is the smallest and largest unsigned integer that can be represented with 8 bits (= 1 byte)? **[2 points]**

(d) In the following, show how you arrive at your answer:

(i) Using the information in the notebook 04-numbers-and-errors.ipynb and the representation $(-1)^s \times 2^{e-1023} \times 1.f$, derive the largest value that can be represented with an IEEE 754 *double* floating point number. **[2 points]**

(ii) For the smallest (in absolute value), non-zero IEEE 754 *double* numbers, $e = 0$ and the value is defined as $(-1)^s \times 2^{e-1022} \times 0.f$. What is the smallest non-zero, absolute value that an IEEE *double* can represent?[1] **[2 points]**

(e) BONUS: Determine experimentally the machine precision ϵ_m for numpy 128-bit floats (`dtype=np.float128`). Show your code and result. **[bonus +4*]**

3.2 NumPy (14 points)

(a) Work through the NumPy tutorial.[2]. Do the examples while you read it. Formulate three questions about numpy — things that you don't understand, that look strange to you, "how do I do X", ...[3] **[3 points]**

(b) Answer the following questions:

[1] The answer is not 0 (looking for non-zero) and not "minus the answer from the previous question" (looking for the absolute value).

[2] Some stuff such as the `ix_()` function is fairly esoteric for beginners but almost everything else is what you should be familiar with for your daily work with arrays.

[3] And no, you can't use the questions further down in this problem.

(i) What is the dimension and the size (i.e. the number of indices in each dimension) of the array `a = np.mgrid[0:5:11j, -10:10:101j, -4:1:1]`? **[1.5 points]**

(ii) List three ways (with Python examples) to create an array. **[1.5 points]**

(iii) How do NumPy array operations such as $+$, $-$, $*$, $/$ ... differ from linear algebra operations (i.e. scalar product, vector/matrix multiplication, ...)? **[2 points]**

(c) Given the three arrays

```
sx = np.array([[0,  1], [1,  0]])
sy = np.array([[0, -1j],[1j, 0]])
sz = np.array([[1,  0], [0, -1]])
```

(i) What is the result of `sx * sy * sz`? Explain what happened. **[2 points]**

(ii) Use `np.dot()` to multiply the three arrays (like $\sigma_x \cdot \sigma_y \cdot \sigma_z$). Show the code that you are running and the final result. Explain what happened. **[2 points]**

(iii) BONUS: Compute the "commutator" $[\sigma_x, \sigma_y] := \sigma_x \cdot \sigma_y - \sigma_y \sigma_x$.[4] **[bonus +2*]**

(iv) Given a "state vector"

$$\mathbf{v} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -i \end{pmatrix}$$

calculate the "expectation value" $\mathbf{v}^\dagger \cdot \sigma_y \cdot \mathbf{v}$ using NumPy. [5] Show your calculation and your result. **[2 points]**

3.3 Errors in numerical summation (22 points)

In class and in the notebook 04-numbers-and-errors.ipynb an algorithm to compute $\sin x$ was derived, based on the infinite series

$$\sin x = \lim_{N \to +\infty} \sum_{n=1}^{N} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!}. \tag{1}$$

In this algorithm, the nth term a_n was computed from the $(n-1)$th term a_{n-1}. As convergence criterion we demanded that

$$\left| \frac{a_n}{\sum_{k=1}^{n} a_k} \right| < \epsilon_m. \tag{2}$$

[4] These are the Pauli matrices that describe the three components of the spin operator for a spin 1/2 particle, $\hat{\mathbf{S}} = \frac{\hbar}{2}\boldsymbol{\sigma}$. The fact that any two components of the spin operator do not commute is a fundamental aspect of quantum mechanics.

[5] The hermitian conjugate $\mathbf{v}^\dagger$ is `v.conjugate().T` where `v.T` is shorthand for `v.transpose()`. It turns out that you don't need the transposition when you use `np.dot()` but I include it here for conceptual clarity.

Including `transpose()` comes at a minor performance penalty — check with `%timeit` if you are curious.

You have a working implementation in the function `sin_series()` in the notebook.[6]

(a) Plot (1) your result and the (for our purposes) exact solution from `np.sin()` as a function of x, (2) the maximum `n` used in the calculation, (3) the relative error

$$\delta = \left| \frac{\text{sin\_series}(x) - \sin x}{\sin x} \right|, \tag{3}$$

and the absolute error

$$\Delta = \left| \text{sin\_series}(x) - \sin x \right|. \tag{4}$$

(You can use the existing code as a starting point and add a new calculation for Δ and a new plot.)

(i) Plot in the range $-2\pi \leqslant x \leqslant 2\pi$ (step 0.1) and **[2 points]**

(ii) in the range $-50\pi \leqslant x \leqslant 50\pi$. **[2 points]**

(iii) Make a table with headings x, $\max n$, sin_series, δ, Δ and show results for five values of x that, in your opinion, best illustrate and quantify the behavior seen in the plots. **[3 points]**

(b) Show that for sufficiently small values of x, your algorithm converges (the changes are smaller than the tolerance level) and it converges to the correct answer. (Use your results from (a) and possibly calculate additional specific values of x and add the results to the table.) **[3 points]**

(c) Compare the number of decimal places of precision obtained (δ) with that expected from Eq. 2. **[2 points]**

(d) Show that there is a range of larger values for which the algorithm[7] converges but that it converges to the wrong answer. Calculate any results that you need for your answer and add them to the table. **[2 points]**

(e) Now make use of the identity $\sin x = \sin(x + 2n\pi)$ (where n is any integer) to compute $\sin x$ for large values of x where the algorithm would otherwise diverge. Show your code. Show the plot for $-50\pi \leqslant x \leqslant 50\pi$ and also show results for five values of x that illustrate the improvement in behavior. **[4 points]**

(f) Implement the "bad algorithm" for calculating $\sin x$ whereby you calculate each term a_n explicitly with factorials (use `math.factorial()` in the `math` library) and powers in x. Show your code. Compare the "good" and the "bad" algorithms by showing plots across a range of values where both algorithms give results and by showing some results explicitly in a table as above. Briefly discuss your results. **[6 points]**

[6] Try the example values and plotting routines in the notebook. Use the code as a pattern for your own work.

[7] Consider the algorithm as implemented, namely without using the identity $\sin x = \sin(x + 2n\pi)$.