Computational Physics I
Problem Sheet 02
(19PHA904)

Loughborough University

Attempt a solution without notes in the first instance keeping a list of things you need to practice and revise.

Questions marked (*) are beyond the material of the week but should be attempted to prepare for the following week and solved within the following three weeks. Those marked (**) are considered very hard and/or are open ended. Revisit these questions regularly throughout the semester as solving them is very beneficial.

Explicit solutions will not be provided.

Last updated: 2020-10-12

---

1. Write a `for` loop that prints the first 10 odd numbers.

2. Write a `for` loop that evaluates the sum

$$S(N) = 4\sum_{k=1}^{N} \frac{(-1)^{k+1}}{2k-1}$$

   for some N of your choosing.

3. Now write a function that computes the above sum and takes N as an input argument. As N gets larger what does the value of this sum converge to?

4. Write a function that evaluates the above sum but does not use any loops [hint: you may find a solution to last weeks problem sheet of use]. (*) Which is better and why [you should test your hypothesis by running code]?

5. Consider the following equation that defines a complex sequence $\{z_n\}$:

$$z_{n+1} = z_n^2 + c$$

   where $c \in \mathbb{C}$ is some fixed complex number and $z_0 = 0$. Use your complex number code from last week to investigate this sequence for a number of different values of c. You should find, for example, that with $c = 1$ that $z_n \to \infty$ while for $c = -1$ the sequence remains finite.

6. Consider making an array of integers using

```
var test: [Int]
```

and then using `.append(0)` and a `for` loop to make it N integers long – for some N (you may want to use a function that takes N as an argument). By running code for large N - compare and contrast this method of creating an array against the more compact code:

```
var test = [Int](repeating: 0, count: N)
```

Note: initialising arrays is common in scientific programming.

7. Consider the d-dimensional vectors $\boldsymbol{u}$ and $\boldsymbol{v}$:

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{bmatrix} \text{ and } \boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{bmatrix} \text{ where } \boldsymbol{u}+\boldsymbol{v} := \begin{bmatrix} u_1+v_1 \\ u_2+v_2 \\ \vdots \\ u_d+v_d \end{bmatrix} a \text{ and } a\boldsymbol{u} := \begin{bmatrix} au_1 \\ au_2 \\ \vdots \\ au_d \end{bmatrix}$$

for some number a ($:=$ means equal to by definition). Write a `DoubleVector` struct that can be used to represent a vector of type `Double` with the above properties. Write test code that verifies your `struct` works consistently with the axioms of a vector space which can be found in the Definition section of `https://en.wikipedia.org/wiki/Vector_space`.

8. Copy and adapt your answer to the previous question to work with your `Complex` `struct` you wrote last week to make a `ComplexVector`. Again – test that this works.

9. Extend `DoubleVector` and `ComplexVector` to include a dot product (also referred to as inner or scalar product). If you do not know how this is defined – use the internet – note that the definition for complex vectors includes conjugation.

10. Note how much code is duplicated in form for Q7-Q9.This is poor practice – why?

11. Write a `CartesianCoordinate struct` to store the coordinates (x, y, z). Add functionality to add coordinates together and multiply by a scalar. Thinking like a physicist – what advantage might such an approach have over recycling the `DoubleVector struct` you have already defined? (*) add a `galileanTransform` function to this `struct` for a frame moving with velocity (v_x, v_y, v_z) and time `time` `https://en.wikipedia.org/wiki/Galilean_transformation`.

(*) 12. Investigate and use operator overloading to add (`+`,`*`,`-`,`/`) to extend your `Complex struct` to have this functionality. Test your code works.

(**) 13. Add to your `Complex` struct a power infix operator (`**`). Test your code works. (You will need to define the operator with this line at the start of your code, note that the precedence is not correct but will work if you are careful.)

```
infix operator **: MultiplicationPrecedence
```

(**) 14. Add to your `struct` the infix operators `"+"` for vector addition, `*` for multiplication by a number and · (bullet) for dot product (is the last example actually sensible?) Access the unicode characters via ctrl+cmd+space.

15. StackOverflow is a really useful resource for programmers – even if the examples are given for another programming language! Read how-to-map-the-indexes-of-a-matrix-to-a-1-dimensional-array-c. Now write a `DoubleMatrix` and `ComplexMatrix struct`. Each should have the following functionality:

    - Adds matrices together to return another matrix.
    - Multiply a matrix by a number to return a matrix.
    - Multiply a matrix by a vector and return a vector.
    - Multiply a matrix by a matrix and return a matrix.

    Don't forget to add the appropriate citation as a comment in your code!

(**) 16. Extend your matrix `struct` to calculate the determinant of the matrix `https://en.wikipedia.org/wiki/Determinant`. Your code should check that the input matrix is square.

(**) 17. Extend your matrix `struct` to calculate the adjoint of the matrix `https://en.wikipedia.org/wiki/Adjugate_matrix`. Your code should check that the input matrix is square.

(**) 18. Extend your matrix `struct` to calculate the inverse of the matrix `http://mathworld.wolfram.com/Gauss-JordanElimination.html`. Your code should check that the input matrix is square.

(***) 19. For experienced/keen programmers only (we will address this subject again later in the module). The problem of duplicated code duplicated in Q7-Q9 and elsewhere can be mitigated using generics, associated types and protocols. Write a generic `Vector` and `Matrix` `struct` that works with a protocol `Number`.