WORKSHEET 2

SOLVING ORDINARY DIFFERENTIAL EQUATIONS

Consider the classical simple harmonic oscillator equation,

$$\frac{d^2y}{dx^2} = -y.$$

It can be written as two first-order ODEs by defining $z \equiv dy/dx$. In fact, this is true in general, and the main problem we want to solve today is to numerically solve the system of coupled equations

$$\frac{dy}{dx} = f(x, y, z)$$

and

$$\frac{dz}{dx} = g(x, y, z).$$

For the SHO, we have $f(x, y, z) = z$ and $g(x, y, z) = -y$. We'll also need boundary conditions – let's suppose that at $x = 0$ we have $y(0) = 1$ and $z(0) = 0$.

A. EULER'S METHOD

First, use Euler's method to solve the equations:

$$y_{n+1} = y_n + k_n,$$

where $k_n = hf(x_n, y_n, z_n)$ (and $h = \Delta x$ is the step-size), and

$$z_{n+1} = z_n + \ell_n,$$

where $\ell_n = hg(x_n, y_n, z_n)$.

Hints:

- Set h to be some small number, say 0.01.
- Use Python functions for $f(x, y, z)$ and $g(x, y, z)$ so you can easily reuse the code.
- Use `np.arange` to create your x_n array so you can specify the step-size, and then use the length of that array to create arrays for y_n and z_n (using `np.zeros`). You'll need an endpoint for x as well – let's take $x_N = 50$.
- Use a loop to advance the solution forward; within the loop, compute k_n and ℓ_n, and then use those to compute y_{n+1} and z_{n+1}.
- Plot your solution, along with the analytic solution (it should be $y(x) = \cos(x)$, right?). How does varying h change things?

B. FOURTH ORDER RUNGE-KUTTA

Now solve it again, but use the fourth order Runger-Kutta method. Recall that

$$\begin{aligned}
k_1 &= hf(x_n, y_n, z_n) \\
k_2 &= hf(x_n + h/2, y_n + k_1/2, z_n + \ell_1/2) \\
k_3 &= hf(x_n + h/2, y_n + k_2/2, z_n + \ell_2/2) \\
k_4 &= hf(x_n + h, y_n + k_3, z_n + \ell_3),
\end{aligned}$$

and similarly for the ℓs. Then the solution is advanced by

$$y_{n+1} = y_n + k_1/6 + k_2/3 + k_3/3 + k_4/6,$$

and similarly for z_{n+1}.

C. Clean Up and Reusability

Create a function for your ODE solvers, with a pattern that looks like:

```
def ode_solver(x_start, x_stop, h, y0, z0, f, g).
```

Finally, create a "library" file, called `phy4910.py`, which contains your two solvers (maybe other things, too!). Test it and make sure it works.