MPM assessment 2

Numerical package development

For this assessment you have been provided with an empty repository. The goal of the assessment is to develop a numerical software package capable of computing some transcendental and special functions from series expansions.

Package specifications

The package being developed should be called `acsefunctions`. Within the repository, a suitable `environment.yml` (for use with Conda) and `requirements.txt` file should be present. From within a suitable environment, it should be possible to install the package by executing

```
pip install -e .
```

Following this, it should be possible to execute the test suite (further details below) via

```
pytest tests/
```

Problem specifications - part 1 [30 marks]

The value of piece-wise continuous and differentiable functions can be approximated at a location x through using a set of known values of the function and its derivatives at some location a. This is of course the Taylor series:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n.$$

When evaluating such expansions numerically, we then must truncate the expansion at some suitable $n = N$.

For the first part of this assessment, write functions that utilize the Taylor series to compute e^x, $\sin(x)$, $\cos(x)$ and $\tan(x)$. For each function, carefully consider the API. Each function should have a suitable docstring. Further, each function should have an associated set of tests to ensure its accuracy over a suitable range of cases.

The four above functions should be exposed to the user and importable in a python script via, e.g.

```
from acsefunctions import exp, sin, cos, tan
```

You may wish to make use of utility functions. These do not need to be exposed to the user.

Problem specifications - part 2 [30 marks]

Special functions frequently appear in Physical and Engineering applications, arising from solutions to partial differential equations. One commonly seen special function, arising from the solution to Bessel's differential equation, is the Bessel Function

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha},$$

where the `gamma function`, $\Gamma(z)$, appearing in the above is defined as

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} \mathrm{d}t,$$

for $\Re(z) > 0$. See also this wikipedia article.

Now, add functions to compute the factorial, the `gamma function` and Bessel's function.

These three functions should meet the same specifications as outlined for the functions in part 1.

Documentation, CI and optimisation [40 marks]

Finally, some features to make your repository 'more professional' will be added. To your repository add

- A test to test your docstrings. This should be run as part of your default python test suite.
- GitHub actions workflows that, upon a commit to the main or a PR'd branch, execute your python tests and ensure your repository is `flake8` compliant. These tests should be constructed such that they ensure the validity of your `conda` environment file and your `requirements.txt`.
- Documentation in the form of sphinx compilable documentation and a jupyter notebook outlining how to use your package and some illustrative examples of each function.
- A GitHub actions workflow to test (and hence ensure the validity of) your documentation notebook.

- A jupyter notebook comparing and discussing the execution of your functions (for a suitable set of values) compared to the Numpy/SciPy equivalents. Here you should also explore the truncation level vs error.
- Replace the text in the original README with a quick intro and installation instructions for your package.

Closing remarks

- For your package and the repository as a whole, carefully consider the quality of your code. Within each section marks will be reserved for code quality and efficiency. **Note however**, that you will not be penalised for choosing a sub-optimal 'mathematical definition' from which to implement any function, only having chosen a particular method how well that method is implemented. For example, it does not matter whether you choose to implement the Gamma function via Euler's definition, Weierstrass's definition or some other method/definition.
- All functions should work for vector inputs. That is, the user should be able to pass inputs as a numpy array (or array like) object.
- Numpy can be used for defining arrays and basic operations (multiplication, addition, division etc.) but no function based numpy routines should be used.
- The method of implementing the Gamma function is up to you: you may use one of the convergent series/products from the Wikipedia article or directly evaluate the integral.
- You may import SciPy/Numpy for testing the various functions you write. You should also ensure your reproduce some well know cases for which analytical results exist, e.g. $\sin(\pi/4) = \sqrt{2}/2$.
- All tests and workflows should be passing at the time of your final submission.
- Remember to ensure that the final version of your repository is `flake8` compliant.
- Keep the sphinx documentation up to date.
- Ensure that your environment.yml and requirements.txt files have been updated appropriately to reflect any new dependencies you've added.
- Jupyter notebooks can be tested using the nbval package.

- If you utilize ChatGPT or any other external tool/reference, this should be clearly indicated via comments in your code and in a suitable references file (e.g. links to conversations with ChatGPT should be included in your references as with assessment 1).