UNIVERSITY OF CAMBRIDGE

MPhil in Data Intensive Science

Michaelmas Term 2024

C1 Research Computing Coursework

B. Bolliet

*Attempt **all** parts of the coursework.*

The anticipated number of marks allocated to each part of a question is indicated in the right margin, to be assessed from the both the code repository and your report.

This coursework should be submitted via a GitLab repository which will be created for you. Place all of your code and your report into this repository. The report should be in pdf format and placed in a folder named `report`*. You will be provided access to your repository until 23:59pm on Wednesday the 18th of December, after which we will assess whatever the repository contains at that time.*

You are expected to submit code and associated material that demonstrates good software development practices as covered in the C1 Research Computing module.

Your report should not exceed 3000 words (including tables, figure captions and appendices but excluding references); please indicate its word count on its front cover. You are reminded to comply with the requirements given in the Course Handbook regarding the use of, and declaration of use of, autogeneration tools.

Failure to include the wordcount and a declarations of use of autogeneration tools will result in automatic loss of marks.

Preamble

In this coursework you will create a package that performs automatic differentiation using dual numbers.
This approach to compute derivatives with dual numbers is known as forward-mode automatic differentiation. Automatic differentiation is at the core of learning algorithms for deep neural networks. The forward mode is efficient for functions with relatively few input variables but many output variables, think of a deep neural network with a few nodes in the input layer and many nodes in the output layer. Formally dual numbers are

like complex numbers, but instead of a real and imaginary part, they have a real part and a dual. In the same way the complex part of complex numbers is carried by i, the dual part of a dual number is carried by ϵ. The major difference with complex numbers is that instead of the imaginary unit $i^2 = -1$, we have $\epsilon^2 = 0$.

Consider the dual number $x = a + b\epsilon$. Then, $x^2 = (a + b\epsilon)^2 = a^2 + 2ab\epsilon$. Note that the derivative of x^2 with respect to x is $2ax$, thus, with $b = 1$, the dual part of x^2, i.e., $2ab\epsilon$, is the derivative of x^2 with respect to x evaluated at $x = a$. In fact, you can show that any function $f(x)$ can be extended to dual numbers and that its dual part is its derivative. You have:

$$f(a + b\epsilon) = f(a) + f'(a)b\epsilon$$

We refer to the part of the course on differentiable programming and the Wikipedia page for more details, and references therein.

There is a total of 150 marks for this coursework.

Tasks

1 Create project repository structure

The project repository should be called `dual_autodiff`. It is a Python project and you should follow the structure according to good practices. Note that we use a `pyproject.toml` file for the project configuration, rather than `setup.py`. [5]

2 Write project configuration

What should the `pyproject.toml` file contain? Make the file accordingly. [5]

3 Implement dual numbers and operations

The main file of your project should be called `dual.py`. It contains a `Dual` class. The `Dual` class defines the Dual numbers. For instance:

```
x = Dual(2, 1)
print(x.real, x.dual)
```

should print `2, 1`.

Furthermore, the `Dual` class defines the mathematical operations. For instance:

```
x = Dual(2, 1)
y = Dual(3, 2)
print(x + y)
```

should print `Dual(real=5, dual=3)`. Similarly, for multiplication, subtraction and division.

The `Dual` class should also define the `sin`, `cos`, `tan`, `log`, `exp` and other essential functions. For instance:

```
def sin(self):
```

and `x.sin()` should return `Dual(real=0.9092..., dual=-0.4161...)` for `x = Dual(2, 1)`. [10]

4 Make the code into a package

Your code should be installable with `pip install -e .` from inside the project folder. Then, you should be able to import the package with `import dual_autodiff as df` from anywhere (using the same environment where you installed the package). [10]

5 Differentiate a function

Consider the function $f(x) = \log(\sin(x)) + x^2 \cos(x)$. Compute its derivative at $x = 1.5$ using dual numbers and compare it to the analytical derivative. Compute also the numerical derivative using an increasingly smaller step size. What do you observe? To answer this question, you should present one or more plots that illustrate your observations and their meaning. [15]

6 Implement a test suite

In the `tests` folder, implement a comprehensive test suite for the `Dual` class, including its operations and the functions in `autodiff_tools`. Ensure that the tests are meaningful and cover a meaningful range of cases. The test suite should be executable with `pytest`, for example:

```
pytest -s tests/*
```

[15]

7 Write project documentation with Sphinx

Use Sphinx as seen in class. Documentation should be generated from the `docs` folder, after running `make html` in the terminal. The html pages should feature all what you have implemented with clear explanations and examples. The documentation should consist of the docstrings and a tutorial notebook `dual_autodiff.ipynb`. The notebook should feature meaningful examples and show users how to use the package and what it can do. [15]

8 **Cythonize the package**

Cythonize the Python package you have made. To do so, create a different, separate directory that you call `dual_autodiff_x`. Note that here, you will need a `setup.py` file in addition to the `pyproject.toml` file. Eventually the package should be installable with `pip install -e .` from inside the project folder `dual_autodiff_x`. [20]

9 **Compare the performance of the pure Python version and the Cythonized version**

In the notebook `dual_autodiff.ipynb`, compare the performance of the pure Python version and the Cythonized version. Do you observe a performance difference? If so, why? If not, why not? Your answer must be quantitative and must include at least one plot to illustrate your observations and conclusions. [20]

10 **Create wheels for Linux**

Use `cibuildwheel` (which uses Docker internally) to create specific wheels of `dual_autodiff_x` for Linux. You should create two wheels for:

- `cp310-manylinux_x86_64`
- `cp311-manylinux_x86_64`

Your wheels should not contain the `*.pyx` files (i.e., the source code) but only `.so` and `*.pyd` files. You can check this by doing:

```
unzip wheelhouse/<name of wheel>.whl -d
   wheel_contents
```

and check the files inside `wheel_contents`. [15]

11 **Upload the wheels to the your GitLab project repository**

Upload the wheels to this GitLab project repository.

After downloading the wheels from your gitlab repository, the package should be installable from wheels, on a linux machine (like CSD3 or your own laptop using a Docker image), using the following command:

```
pip install dual_autodiff_x_<name_of_wheel>.whl
```

and should run correctly the examples of the tutorial notebook. [20]

END OF PAPER