C1 Research Computing Coursework Report

Hania Rezk (hhmmer2)

MPhil Data Intensive Science, Department of Physics

26 November 2024

Word count: [1000]

Contents

1 Introduction

This report discusses the development of a Python package that performs automatic differentiation using dual numbers. Dual numbers, represented as $x = a + b\epsilon$, consist of a real part a, and a dual part b carried by ϵ that satisfies $\epsilon^2 = 0$. These numbers are used to compute derivatives efficiently; this approach, called forward-mode automatic differentiation, is particularly valuable in fields like machine learning as it enables fast training of deep neural networks.

This report describes the step-by-step creation of this package and answers the questions of the coursework sheet. First, we will overview the global structure of the package (question 2 of CW) and present the `Dual` class which allows us to represent and perform operations on dual numbers. Secondly, a test suite that covers a meaningful range of cases was implemented to make sure our `Dual` class works as expected. Using this same class, this package differentiates a function and compares the result to the analytical and numerical derivatives to measure the precision of automatic differentiation (question 5 of CW). Finally, we will Cythonize the package and compare its performance to the pure Python version (question 8 of CW). The entirety of the code is documented according to the good practices taught in the Research Computing module.

2 Package Structure and Project Configuration

This package respects the structure seen in the Research Computing course as shown below.

```
.
+-- pyproject.toml              # Configuration
+-- README.md                   # Instructions
+-- dual_autodiff/              # Package folder with codes
|   +-- __init__.py
|   +-- version.py              # Manage package versions
|   +-- dual.py                 # Dual class
+-- dist/                       # Distribution files
+-- docs/                       # Documentation files
|   +-- Makefile                # Commands to build the documentation
|   +-- requirements.txt
|   +-- conf.py                 # Configuration of the documentation
```

```
|   +-- index.rst                # The main page of the documentation
|   +-- material/
+-- tests/                       # Test folder
|   +-- autodiff_tools.py        # Test suite for Dual class
+-- Question5.py                 # Code for question 5 of the coursework
```

3 `Dual` Class

The main file of this package is called `dual.py`. It contains the core module of our package and defines the `Dual` class. This class defines a structure for dual numbers and performs standard operations on them. An instance of this class has two attributes: one to represent the real part of our number and another to represent the dual part of our number.

Automatic differentiation exploits the fact that every computer calculation executes a sequence of elementary arithmetic operations and functions; this highlights the importance of the next part.

3.1 Operators

The `Dual` class redefines the standard operators to make them compatible with dual numbers. The operators that are redefined are: `+`, `-`, `*`, `**`, `/`, `//`, `%`, `==`, `!=`. The redefinitions consider a wide range of cases, illustrated in the `Details` section. Short-hand versions of the operators (`+=`, `-=`, etc.) are also redefined.

3.1.1 Details

- **The `__add__` and `__radd__` Methods**
  These methods redefine the `+` operator as follows:

  1. If both x and y are dual numbers, $y = a' + b'\epsilon$ and $x = a + b\epsilon$, `y.__add__(x)` is called for the operation:

  $$y + x = (a' + a) + (b' + b)\epsilon$$

  2. If y is a dual number and x is a scalar:

  $$y + x = (a' + x) + b'\epsilon$$

  3. If x is a dual number and y is a scalar, `y.__radd__(x)` is called for the operation:

  $$x + y = (a + y) + b\epsilon$$

- **The `__sub__` and `__rsub__` Methods**
  These methods redefine the `-` operator as follows:

  1. If both x and y are dual numbers, $y = a' + b'\epsilon$ and $x = a + b\epsilon$, `y.__sub__(x)` is called for the operation:

     $$y - x = (a' - a) + (b' - b)\epsilon$$

  2. If y is a dual number and x is a scalar:

     $$y - x = (a' - x) + b'\epsilon$$

  3. If x is a dual number and y is a scalar, `y.__rsub__(x)` is called for the operation:

     $$x - y = (a - y) + b\epsilon$$

- **The `__mul__` and `__rmul__` Methods**
  These methods redefine the `*` operator as follows:

  1. If both x and y are dual numbers, $y = a' + b'\epsilon$ and $x = a + b\epsilon$, `y.__mul__(x)` is called for the operation:

     $$y * x = (a'*a) + (b'*a + b*a')\epsilon$$

  2. If y is a dual number and x is a scalar:

     $$y * x = (a'*x) + b'\epsilon$$

  3. If x is a dual number and y is a scalar, `y.__rmul__(x)` is called for the operation:

     $$x * y = (a*y) + b\epsilon$$

- **The `__truediv__` and `__rtruediv__` Methods**
  These methods redefine the y/x operation as follows:

  1. If both x and y are dual numbers, $y = a' + b'\epsilon$ and $x = a + b\epsilon$, `y.__truediv__(x)` is called for the operation:

     $$y/x = \frac{a'}{a} + \frac{a * b' - b * a'}{a^2}\epsilon$$

     and returns an error if the real part of x is zero.

2. If y is a dual number and x is a scalar:
$$y/x = \frac{a'}{x} + \frac{b'}{x}\epsilon$$
and returns an error if x is zero.
3. If x is a dual number and y is a scalar, `y.__rtruediv__(x)` is called for the operation:
$$x/y = \frac{a}{y} + \frac{b}{y}\epsilon$$
and returns an error if y is zero.

- **The `__pow__` Method**
This method redefines the `**` operator as follows:
1. If both x and y are dual numbers, $y = a' + b'\epsilon$ and $x = a + b\epsilon$, `y.__pow__(x)` is defined as: $\begin{array}{l}\text{if } (b' == 0) : y**x = a'**x + \epsilon\ x*a'**(x-1)\\ \text{if } (b' \neq 0) : \text{returns an error.}\end{array}$
2. If y is a dual number and x is a scalar:
$$y**x = a'**x + x*a'**(x-1)\epsilon$$

- **The `__eq__` and `__ne__` Methods**
These methods redefine the `==` and `!=` operators as follows:
1. If both x and y are dual numbers, $y = a' + b'\epsilon$ and $x = a + b\epsilon$, the equality is evaluated as:
$$\text{if } (a == a') \text{ and } (b == b'),$$
it returns `True` for `==` and `False` for `!=`.
2. If y is a dual number and x is a scalar:
$\begin{array}{lll}\text{if } (b' == 0) : & \text{if } (x == a'), & \text{return } \texttt{True} \text{ for } \texttt{==} \text{ and } \texttt{False} \text{ for } \texttt{!=}.\\ & \text{if } (x \neq a'), & \text{return } \texttt{False} \text{ for } \texttt{==} \text{ and } \texttt{True} \text{ for } \texttt{!=}.\\ \text{if } (b' \neq 0) : & & \text{return } \texttt{False} \text{ for } \texttt{==} \text{ and } \texttt{True} \text{ for } \texttt{!=}.\end{array}$

- **The `__neg__` Method**
This method redefines the `-` operator by negating the real and dual parts of x, changing the real part to $-a$ and the dual part to $-b$.

- **Shorthand operators**
The method `__iadd__` redefines the shorthand operator `+=` using the same logic as `__add__`. Similarly, the methods `__isub__`, `__imul__`, `__itruediv__`, and `__ipow__` redefine the shorthand operators `-=`, `*=`, `/=`, and `**=`, respectively, using the same logic as their corresponding binary methods (`__sub__`, `__mul__`, `__truediv__`, `__pow__`).

3.2 Functions

The `Dual` class also defines essential functions to make them compatible with dual numbers. The essential functions defined are: exponential, sine, cosine, tangent, logarithm, ceil, floor, inverse, square, and abs. Like the case of our operators, the definition of our functions also considers illegal calls for the functions : inverse logarithm and tangent, and an error is raised to prevent a division by zero.

3.2.1 Details

The definition of essential functions lies completely on the f(a + bϵ) formula that was provided for us in the coursework sheet, which provides us with the formulas for the real part and the dual part of the returns of each of our functions:

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

The real part corresponds to $f(a)$, and the dual part corresponds to $f'(a)b$. The only work we have to do is compute the derivative of each essential functions we defined.

- Derivative of $\exp(x)$ is $\exp(x)$.
- Derivative of $\log(x)$ is $1/x$, raising a `ZeroDivisionError` when the real part of $x = 0$.
- Derivative of $\tan(x)$ is $1/\cos^2(x)$, raising a `ZeroDivisionError` when $\cos(x.real) \approx 0$ (detected using `np.isclose()`).
- Derivative of $\cos(x)$ is -$\sin(x)$.
- Derivative of $\sin(x)$ is $\cos(x)$.
- Derivative of $square(x)$ is $2x$.
- Derivative of $inverse(x)$ is -$1/x^2$, raising a `ZeroDivisionError` when the real part of $x = 0$.

For `abs(x)`, `floor(x)`, and `ceil(x)`, we just apply these to the real part and the dual part of the dual number x.

4 Test Suite

To ensure the reliability of our `Dual` class, a comprehensive test suite was developed. The `autodiff_tools.py` file in the `tests` folder executes 31 tests on our class, with each test corresponding to a specific function in the dual.py file. We tested a variety of dual numbers with both integer and float values for their real and dual parts. For functions that raise errors, invalid inputs were also tested to ensure the correct errors were produced. To run the tests with pytest, run the following command in your terminal from the project repository:

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

5 Differentiating a function

In this section, we use dual numbers to compute the derivative of the function $f(x) = \log(\sin(x)) + x^2 \cos(x)$ (question 5 of the coursework). We define two functions in our `question5.py` file:

1. A function to compute the numerical derivative of $f(x)$ with a step size h, returning $\frac{f(x+h)-f(x)}{h}$.
2. A second function that evaluates the analytical derivative of the function at a point x (which I calculated manually):

$$f'(x) = \frac{\cos(x)}{\sin(x)} + 2x\cos(x) - x^2\sin(x).$$

To find the derivative using dual numbers, we simply initialize a dual number $x_1 = \text{Dual}(x, 1)$ and then apply the functions defined in our Dual class to evaluate

$$f_1 = (x_1.\cos()*x_1.\text{square}() + (x_1.\sin()).\log()).$$

Since $b = 1$, the derivative with respect to x is the dual part of f_1, as shown by the formula provided in the coursework:

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

5.1 Observations

We observe that the analytical derivative and the derivative computed through dual numbers are exactly the same, with the same values up to the 16th decimal point, which is the maximum precision for numpy (since the Dual class uses `float64` by default for both the real and dual parts of the dual numbers).

I have also defined a time decorator to see how long each function takes to execute. Both the analytical derivative and the dual numbers derivative are computed in a time order of 1e-5 seconds.The analytical one seems to compute slightly faster, which is expected as the analytical approach involves pre-computed derivatives, while dual numbers compute the derivative, from scratch, using function evaluation. This illustrates the usefulness of dual numbers, as they save a lot of effort and eventually time when differentiating.

Figure 1 below illustrates that as the step size gets smaller, the numerical derivative approaches the true analytical value (the plot was obtained using the `numerical_values()` function in `question5.py`). Additionally, we observe that beyond a certain threshold, when the step size becomes too small, the numerical derivative values deviate from the analytical value.

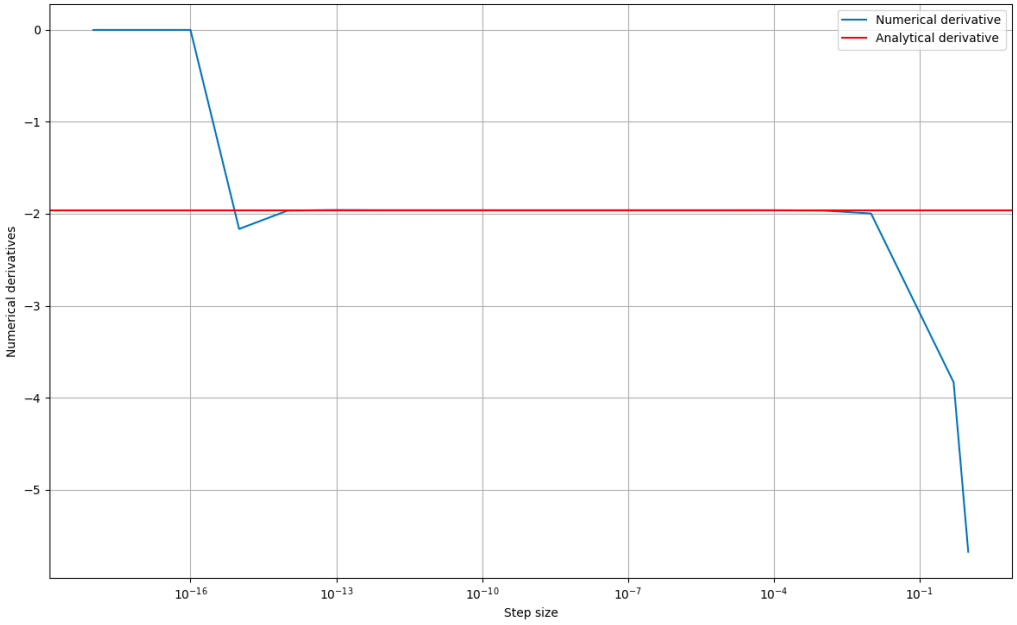

Figure 1:

More precisely, the `interval()` function examines what happens in the interval of steps $[1e-16, 1e-13]$. Figure 2 shows that for smaller step sizes than $1e-13$, the numerical derivatives start to oscillate heavily, which is caused by the limitations of floating-point precision and round-off errors that become more pronounced as the step size gets infinitely small.

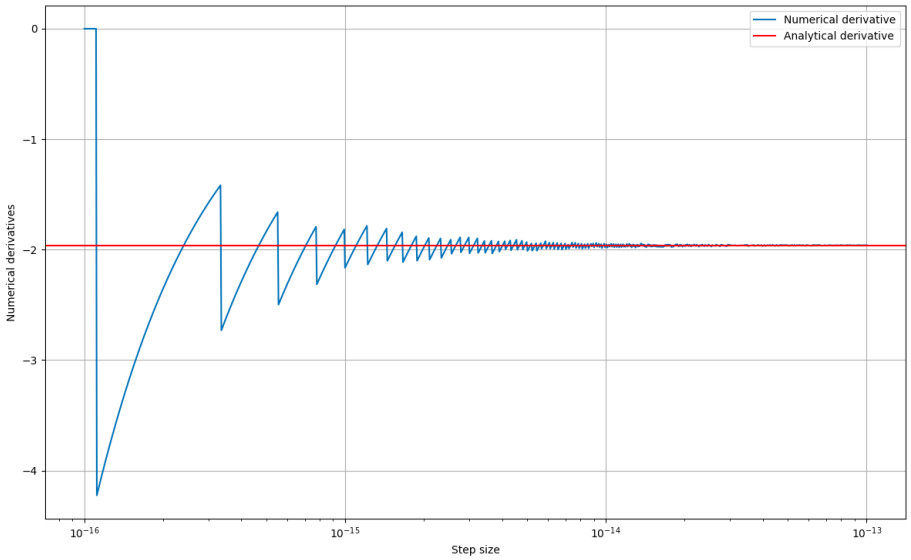

Figure 2:

6 Documentation

Since the first file created in my package, I made sure to document my code using docstrings. Each function includes a docstring that describes its purpose, parameters and returns values, which was very helpful when generating the Sphinx documentation. The structure mirrors the `Read the Docs` page of the C1 course, as I used a theme provided by Read the Docs ('sphinx_rtd_theme'), and copied the structure of the docs folder in the GitHub repository of the research computing course created by Dr. Boris Bolliet. To generate the html pages of the documentation, please run the following command in your terminal from docs folder (present in the project repository):

```
make html
```

The documentation can be viewed by opening docs/build/html/index.html in a web browser.

7 Questions 8-9

8 Summary

9 Declaration of AI Tools

ChatGPT was the only AI tool used in the creation of this project. It was primarily used to correct spelling mistakes in this report, and correct my .tex file (source of this LATEX report). However, the entire report was written by me and I didn't use any sentence reconstruction recommendations. ChatGPT was also used to generate values of the dual numbers used in my tests and provided me with the results of the operations on these numbers: It did not write the code or any part of it, it just gave me an output that looks like this:

Example of Dual Numbers Addition:

$$x = \text{Dual}(5, 3), \quad y = \text{Dual}(2, 4), \quad x + y = \text{Dual}(7, 7)$$

10 References

1. `https://en.wikipedia.org/wiki/Automatic_differentiation`
2. `https://github.com/borisbolliet/ResearchComputing/tree/main/docs`