C1 Research Computing - Coursework Assignment

Raunaq Rai (rsr45@cam.ac.uk)
Department of Physics, University of Cambridge

18 December, 2024

1 Introduction

This report details the development and implementation of a Python package, `dual_autodiff`, designed for automatic differentiation using dual numbers. The package computes derivatives efficiently while supporting mathematical operations such as trigonometric, logarithmic, and exponential functions.

The approach builds on the concept of forward-mode automatic differentiation, which is essential in fields like optimization, computational physics, and machine learning. This technique traces its roots to the foundational work by Wengert [1], who introduced a systematic way to compute derivatives using intermediate variables. More recently, Baydin et al. [2] surveyed the use of automatic differentiation in machine learning, emphasizing its importance in training deep neural networks.

To enhance performance, a Cython-optimized version, `dual_autodiff_x`, was also developed. This document covers the structure of the project, the mathematical principles behind dual numbers, and the implementation details of the package.

2 Setting Up the Development Environment

Apple Silicon devices primarily use the ARM64 architecture, which can pose challenges when working with scientific computing tools built for x86_64. To ensure compatibility with these tools, I configured the development environment to run in x86_64 mode. This step was crucial for enabling seamless execution of packages and tools designed for x86_64 systems.

To achieve this:

- **Rosetta Installation:** Rosetta, an emulation layer by Apple, was installed to facilitate running x86_64 binaries on ARM-based devices. This was achieved using:

```
/usr/sbin/softwareupdate --install-rosetta
```

- **Configuring Terminal:** The Terminal application was set to run in Rosetta mode, ensuring compatibility with x86_64 libraries and tools.

- **Creating an x86_64 Conda Environment:** A dedicated Conda environment was created with all required dependencies for developing and testing the package.

This setup allowed for consistent development and ensured the compatibility of tools and libraries required for this project.

3 Theoretical Background

3.1 Dual Numbers

Dual numbers can be defined as truncated Taylor series of the form:

$$x = v + \dot{v}\epsilon,$$

where $v, \dot{v} \in \mathbb{R}$, and ϵ is a nilpotent number such that $\epsilon^2 = 0$ and $\epsilon \neq 0$. Here:

- v: Represents the *primal value*.
- $\dot{v}$: Represents the *derivative* or *tangent value*.

As explained by Baydin et al. [2], arithmetic operations with dual numbers align naturally with symbolic differentiation principles:

$$(x_1 + \dot{x}_1\epsilon) + (x_2 + \dot{x}_2\epsilon) = (x_1 + x_2) + (\dot{x}_1 + \dot{x}_2)\epsilon,$$

$$(x_1 + \dot{x}_1\epsilon)(x_2 + \dot{x}_2\epsilon) = x_1x_2 + (x_1\dot{x}_2 + \dot{x}_1x_2)\epsilon.$$

3.2 Automatic Differentiation

Automatic differentiation (AD) leverages dual numbers to compute derivatives efficiently. For a function $f(x)$, substituting $x = v + \dot{v}\epsilon$ yields:

$$f(x) = f(v + \dot{v}\epsilon) = f(v) + f'(v)\dot{v}\epsilon.$$

The derivative $f'(v)$ is embedded in the coefficient of ϵ, enabling simultaneous evaluation of function values and derivatives.

This principle extends to composite functions via the chain rule:

$$f(g(v + \dot{v}\epsilon)) = f(g(v)) + f'(g(v))g'(v)\dot{v}\epsilon.$$

4 Implementation of Dual Numbers and Operations

4.1 Overview of the `Dual` Class

The `dual.py` file implements the `Dual` class, the core of the `dual_autodiff` package. This class defines dual numbers and supports operations such as addition, subtraction, multiplication, and division.

4.1.1 Arithmetic Operations

The `Dual` class overrides arithmetic operators for seamless integration. For example:

```
x = Dual(2, 1)
y = Dual(3, 2)
print(x + y)  # Output: Dual(real=5, dual=3)
```

4.1.2 Mathematical Functions

The `Dual` class also implements key mathematical functions such as:

- Trigonometric functions (`sin`, `cos`, `tan`).
- Exponential and logarithmic functions (`exp`, `log`).
- Hyperbolic functions (`sinh`, `cosh`, `tanh`).
- Square root (`sqrt`).

For example:

```
x = Dual(2, 1)
result = x.sin()
print(result)  # Output: Dual(real=0.9092..., dual=-0.4161...)
```

4.2 Utility Functions

To enhance usability:

- `functions.py` provides aliases for mathematical functions.
- `base.py` includes helper functions like:
  - `is_dual_instance(value)`: Checks if a value is a `Dual` instance.
  - `ensure_dual(value)`: Wraps non-`Dual` values into a `Dual` object.

5 Project Structure and Packaging

5.1 Repository Organization

The repository follows good practices for Python projects:

- `dual_autodiff/`: Core implementation.
- `tests/`: Unit tests.
- `docs/`: Documentation.
- `pyproject.toml`: Modern project configuration file.
- `requirements.txt`: Lists Python dependencies.
- `environment.yaml`: Defines the Conda environment.

5.2 Building and Installing the Package

The `pyproject.toml` file specifies build metadata:

- `[build-system]`: Declares build tools.
- `[project]`: Contains metadata like name and dependencies.
- `[tool.setuptools_scm]`: Handles dynamic versioning.

To install the package, the following commands were used:

```
pip install build
python -m build
pip install -e .
```

6 Conclusion

By combining the theoretical principles of dual numbers with practical implementation strategies, the `dual_autodiff` package provides an efficient solution for automatic differentiation. The structured approach to repository organization and environment setup ensures robustness and compatibility across platforms.

References

[1] RE Wengert. "A Simple Automatic Derivative Evaluation Program". In: *Communications of the ACM* 7.8 (1964), pp. 463–464.

[2] Atılım Güneş Baydin et al. "Automatic Differentiation in Machine Learning: A Survey". In: *Journal of Machine Learning Research* 18.153 (2018), pp. 1–43.