The Willing Kingdon Clifford Algebra Library

MARTIN ROELFS, Flanders Make, Belgium

Kingdon is a new Python package designed to seamlessly integrate Geometric Algebra (GA) into existing workflows. Unlike existing GA libraries, Kingdon is input-agnostic, and hence supports GA's over e.g. PyTorch tensors, NumPy arrays, or SymPy symbolic expressions, to name but a few. Furthermore, it prioritizes performance by optimizing operators and leveraging input sparsity for Just-In-Time compiled expressions. Additionally its visualization capabilities in Jupyter notebooks using Ganja.js align with the rapid prototyping workflow common to scientific research.

1 INTRODUCTION

The Python programming language is very popular in the scientific community - amongst other things - because of its ease of use and rapid prototyping. This has resulted in a flourishing ecosystem of tooling for e.g. data analysis such as `numpy`, `scipy` and `pandas`; machine learning such as `PyToch` and `TensorFlow`, etc. Many of these tools are used to analyse problems involving geometry, in fields ranging from computer vision to astronomy; problems whose solutions would benefit from Geometric Algebra (GA) support. However, existing GA projects in Python tend to be aimed towards one specific purpose. For example, the popular `clifford` library assumes the input to be numerical, while the `galgebra` library can only be used for symbolic manipulations, and the `tfga` library is explicitly coupled to `TensorFlow`. While these specialized packages are all powerful within their own domain, they are not conducive to the rapid prototyping workflow for which Python is famous. For example, it is not possible to setup calculations using simple numerical examples or symbolical inputs, and to then simply change the input to `PyTorch` tensors.

This is the main motivation behind the new `kingdon` Python package; the goal is to make it easy to add GA to any existing workflow. Therefore `kingdon` was designed to be agnostic to input types, such that the exact same code will work on e.g. `PyTorch` tensors, `numpy` arrays, or `sympy` symbolic expressions, to name but a few. Moreover, `kingdon` was written with performance in mind: it symbolically optimizes GA's unary and binary operators and leverages the sparseness of the input in order to create Just-In-Time compiled expressions of optimal computational complexity. While this on-the-fly code generation incurs some initial cost upon first execution, subsequent evaluations will be faster, and so this investment will pay of when the goal is to e.g. train a neural network or analyze astronomically large amounts of data. Furthermore, in keeping with the rapid prototyping mindset, `kingdon` can visualize scenes within `jupyter` notebooks using the popular `ganja.js` package.

This talk will introduce the `kingdon` library and its design principles by giving some real life examples of its usage in applications at Flanders Make.

2 DESIGN PHILOSOPHY

The design philosophy behind `kingdon` is characterized by the following key principles:

- Agnostic to input types, such that `kingdon` plays well together with existing mathematics libraries. If the input types support addition, subtraction, multiplication, division and square root operations, then they can be used as multivector coefficients in `kingdon`. This includes popular datatypes such as `PyTorch` tensors, `numpy` arrays, or `sympy` symbolic expressions.
- While GA is famed for its multivectors, it is actually very rare that one needs to perform operations between full multivectors. More often than not, only parts of a multivector are needed. For example, in 3DPGA points, lines, and planes are encoded using trivectors,

Author's address: Martin Roelfs, Department of Physics, Flanders Make, Kortrijk, Belgium, martin.roelfs@flandersmake.be.

bivectors and vectors respectively, and therefore need far fewer coefficients than a full multivector. The most one ever needs in typical geometric applications is an even or odd multivector, since transformations are performed by conjugating an element with an even or odd versor. The `kingdon` library is designed to always take advantage of such sparsity of the input.

- All the common unary and binary operators of GA are symbolically optimized to create Just-In-Time compiled expressions which are (close to) computationally optimal.
- Clean API.
- Python is slow, so we want to do as little of it as possible. When stringing GA operators together to form more complicated expressions, the user can choose to have `kingdon` form this expression into a new Just-In-Time compiled function such that the cost of using python is incurred only once.

3 TALK OUTLINE

(1) **5m - Introduction**
Introduction to the `kingdon` library, what part of the Python GA market it aims for, and the design principles behind it.

(2) **15m - Examples**
The main body of the talk will consist of usage examples. If time permits, we might even do some of these as live coding demonstrations.
We will start by giving some simple examples of `kingdon`'s API and `ganja.js` integration. We then proceed to give some examples of how `kingdon` has been used within Flanders Make so solve industrial problems. Lastly, we show how easy it is to use `kingdon` with `PyTorch` tensors to solve problems in 3D geometry, which demonstrates `kingdon`'s potential for use in geometric deep learning.

(3) **5m - Conclusion & Outlook**
The talk is concluded with an overview of `kingdon`'s capabilities, as well as with future improvements.