Doc

Introduction

Automatic Differentiation (AD) is to solve for the derivative of a function at a given point of estimate. It utilizes the concept of dual number to achieve an accuracy better than numeric differentiation. It is also more efficient than symbolic differentiation.

Background

In the most general case, a function can have more than one coordinate. To evaluate this function, we would take the sum of the partial derivatives with respect to each said coordinate. To illustrate, consider function $f(u(t), v(t))$; we first apply the chain rule to for each piece, we get:

$$\frac{df}{dt} = \frac{\partial f}{\partial u}\frac{du}{dt} + \frac{\partial f}{\partial v}\frac{dv}{dt}$$

At a lower level, the implementation of AD requires breaking down the original function into smaller pieces known as elementary functions. For instance, consider function

$$f(x, y) = exp(sin(3x) + cos(4y))$$

Then these would be its elementary functions:

$$g(z) = 3z$$

$$g(z) = 4y$$

$$g(z) = sin(z)$$

$$g(z) = cos(z)$$

$$g(z) = exp(z)$$

Furthermore, the order of evaluating these elementary functions can be organized into a computational graph:

Note the subsequent nodes named v_i; these store the intermediate results of evaluating each elementary function. After evaluating each v_i, we get a sequence from v_1 to v_8; this is called the primal trace. Similarly, following the computational graph, if we instead evaluate the derivative at each step, the resulting sequence would be called the dual trace. Such is essentially the procedure of automatic differentiation, specifically the forward mode, which we plan to implement. Note that there exists a counterpart named reverse mode; we will not describe it here.

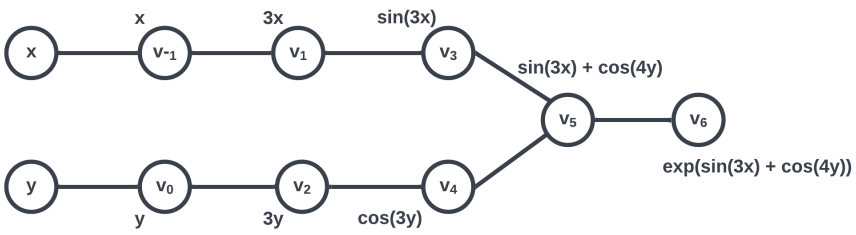

Figure 1: Example Computational Graph

How to use TEAM20AD

Given that this package will be distributed with pyPI, the user will first need to install the package.

```
python -m pip install TEAM20AD
```

They would need to import all the dependable packages, namely numpy, scipy, pandas, and matplotlib.

When it's time to use the package, they will import by:

import TEAM20AD ad tad

User will be able to instantiate the AD objects as follows:

```
f = some_function
x = some_value
Ad = tad()
res = tad.forward(f, x)
```

Software Organization

For now at this phase of the project, our directory structure will be as follows (tentative):

```
team20/
docs
milestone1
    (other milestones)
LICENSE
README.md
untitled.yml
    test
        test.py
    src
```

```
__init__.py
__main__.py
adfun.py
api.py
utils.py
```

As our team continues with the development, we expect the directory structure to change accordingly.

Considering that the whole scheme of auto differentiating will rely heavily on mathematical computations, we will use numpy, scipy, pandas, and math for calculations, along with matplotlib for graphics.

We plan on keeping track of the test suites by having all the tests in the ./test directory.

As of now, we plan to distribute the package using PyPI following PEP517/PEP518.

Implementation

The first class we need and that will implement first - at least at the naive conceptual level - is the TAD class that will serve as the first user instantiated object in calculating the auto differentiation. We will then implement the utility class Util that will have overwritten methods for all the primitive arithmetic along with sin/cos computation. Dual numbers will be an object of itself containing the necessary components.

As for the name attributes and methods for classes, we tentatively plan to have the following:

- TAD:
  - Name Attribute: values, derivatives, shape
  - Methods: constructor (**init**), **add**, **sub**, **mul**, **div**
- Util:
  - For util methods, we will update as necessary.
- Graph:
  - We expect to need and implement a graph class to resemble the computational graph in forward mode.
  - Method: graph
- Calculate:
  - Here's a separate calculate class for basic operator overloading:
  - Methods: cos(tad), sin(tad), tan(tad), sqrt(tad), exp(tad), log(tad, b)

As for the handling of $f : \mathbb{R}^m -> \mathbb{R}$ and $f : \mathbb{R}^m -> \mathbb{R}^n$, we will have a high-level function object in form of vectors to compute the Jacobian.

We will need to depend on the libraries mentioned above, namely numpy, scipy, and matplotlib.