Doc

Introduction

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

Differentiation, the process of finding a derivative, is one of the most fundamental operations in mathematics. It measures the rate of change of a function with respect to a variable. Computational techniques of calculating differentiations have broad applications in many fileds including science and engineering which used in finding a numerical solution of ordinary differential equations, optimization and solution of linear systems. Besides, they also have many real-life applications, like edge detection in image processing and safety tests of cars.

There are three popular ways to calculate the derivative:
1. Numerical Differentiation: Finite Difference
2. Symbolic Differentiation
3. Automatic Differentiation

Symbolic Differentiation and Finite Difference are two ways to numerically compute derivatives. Symbolic Differentiation is precise, but it can lead to inefficient code and can be costly to evaluate. Finite Difference is quick and easy to implement, but it can create round-off error, the loss of precision due to computer rounding of decimal quantities, and truncation error, the difference between the exact solution of the original differential equation.

Automatic Differentiation is more efficient than two of other methods mentioned prior. While it utilizes the concept of dual number, it achieves machine precision without costly evaluation, and therefore is widely used.

Background

1. Basic Calculus

- Product Rule

Product rule is a formula used to find the derivatives of products of two or more functions. The product rule can be expressed as:

$$\frac{\partial}{\partial x}(uv) = u\frac{\partial v}{\partial x} + v\frac{\partial u}{\partial x}.$$

- Chain Rule
  Chain rule is a formula to compute the derivative of a composite function.
  The chain rule can be expressed as:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}.$$

2. Dual Numbers

- A dual number consists of two parts: *real* (denoted as a) and *dual* (b),
  usually written as

$$z = a + b\epsilon,$$

  where $\epsilon \neq 0$ is a nilpotent number with the property $\epsilon^2 = 0$.

3. Automatic Differentiation

- Automatic Differentiation refers to a general way of taking a program which computes a value, and automatically constructing a procedure for computing derivatives of that value. The derivatives sought may be first order (the gradient of a target function, or the Jacobian of a set of constraints), higher order (Hessian times direction vector or a truncated Taylor series), or nested. There are two modes in Automatic Differentiation: the forward mode and the reverse mode.
- Evaluation Trace of a Function: All numeric evaluations are sequences of elementary operations. The evaluation of f at some point x = (x1, ..., xn) can be described by a so-called evaluation trace $v_{k-m} = x_k$, for $k = 1, 2, ..., m$, where each intermediate results v_j are functions that depend on the independent variables x.
- *Elementary functions*: The set of elementary functions has to be given and can, in principle, consist of arbitrary functions as long as these are sufficiently often differentiable. All elementary functions will be implemented in the system together with their gradients.
- *Evaluating (forward) trace of a function*: All numeric evaluations are sequences of elementary operations. The evaluation of a function f at a given point $x = (x_1, \ldots, x_n)$ can be described by a so-called evaluation trace $v_0 = v_0(x), \ldots, v_m = v_m(x)$, where each intermediate result v_j is the result of an ementary operation and a function that depends on the independent variables x.

4. Forward Mode of Automatic Differentiation

- Forward automatic differentiation divides the expression into a sequence of differentiable elementary operations. The chain rule and well-known differentiation rules are then applied to each elementary operation.
- Forward automatic differentiation computes a tangent trace of the directional derivative

$$D_p v_j = (\nabla y_i)^T p = \sum_{j=1}^{m} \frac{\partial y_i}{x_j} p_j$$

  for each intermediate variable v_j at the same time as it performs a forward evaluation trace of the elementary pieces of a complicated $f(x)$ from the inside out.
  Note that the vector p is called the seed vector which gives the direction of the derivative.
- Implementation with dual numbers: by its properties, a dual number can encode the primal trace and the tangent trace in the real and dual parts, respectively.

$$z_j = v_j + D_p v_j \epsilon$$

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 a function $f(u(t), v(t))$; we first apply the chain rule to for each piece, we get:

$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial t} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial t}$$

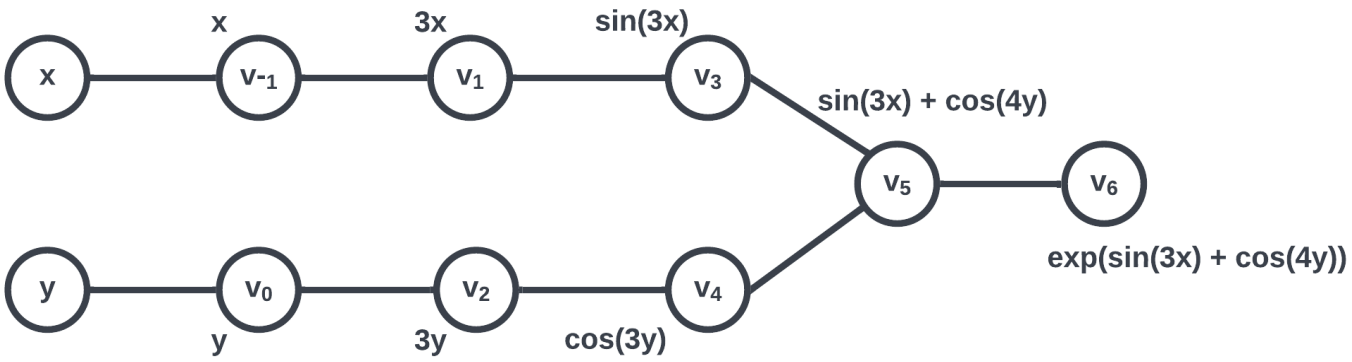
At a lower level, the implementation of AD requires breaking down the original function into smaller pieces known as elementary functions. For instance, consider function

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

Then, f can be broken down into five elementary functions:

$$g_1(z) = 3z, \tag{1}$$
$$g_2(z) = 4z, \tag{2}$$
$$g_3(z) = \sin(z), \tag{3}$$
$$g_4(z) = \cos(z), \tag{4}$$
$$g_5(z) = \exp(z). \tag{5}$$

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



Note that the subsequent node v_i stores the intermediate result 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 which we will not describe it here.

How to use team20ad

This package is distributed through the Python Package Index (PyPI), and hence the user can install it with:

```
python -m pip install team20ad
```

In addition, they need to install and import all the dependable packages including `numpy`, `scipy`, `pandas`, and `matplotlib`.

To use the `team20ad` package, one can import the module by:

```
from team20ad.forward_ad import ad
```

The user will be able to instantiate an AD object and compute the differentation as follows:

```
f = some_function_to_be_differentiated
x = some_value_to_evaluate
ad_obj = ad() # instantiate an automatic differentiation object
res = ad_obj.forward(f, x) # compute derivative of f evaluated at x using forward mode AD
```

Software Organization

For now at this phase of the project, our software directory is tentatively structured as follows:

```
team20/
|-- docs/
|  |-- milestone1.md
|  \-- milestone1.pdf
|-- LICENSE
|-- README.md
|-- pyproject.toml
|-- .github/workflows/
|       |-- coverage.yml
|       \-- test.yml
|-- tests/
|       |-- check_coverage.sh
|       |-- run_tests.sh
|       |-- forward_ad/
|       |  |-- test_forward.py
|       |  \-- test_dual.py
|       \-- overloads/
|          \-- test_overloads.py
\-- src/
        \-- team20ad/
          |--   __init__.py
          |--   __main__.py
          |--   example.py
          |--   forward_ad/
          |    |-- __init__.py
          |    |-- forward.py
          |    \-- dualNumber.py
          \--   overloads/
                   |-- __init__.py
                   \-- function_overloads.py
```

Currently, we plan to have two modules: one for implementing the forward mode of automatic differentiation and the other for defining function overloads (more on this under the Implementation section). The `forward_ad` module will include an implementation of `DualNumber` class, which is necessarily for the forward mode computation.
Note that an implementation of computational graph is optional for forward AD.

As such, we will have correponding tests `test_forward.py` and `test_overloads.py`, which are located under the `tests/forward_ad` and `tests/overloads` directories, respectively, and which will be configured to run automatically using GitHub workflows after each push to the `main` branch of development.

As the development progresses, we expect the directory structure to change and the documentation to update accordingly.

Considering that the whole scheme of auto-differentiating will rely heavily on mathematical computations, we will use `numpy`, `scipy`, `pandas`, and `math` modules for implementations and calculations, along with (possibly) `matplotlib` for graphical representations.

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 `DualNumber` class which will serve as the lower level structure of the forward AD class implementation. This class will implement basic function verloaders such as `__add__()` and their reverse counterparts such as `__radd__()`.
Along with the `DualNumber` class, we will implement the `ForwardAD` class which will serve as a function decoration for computing the derivatives.
We will then implement the function overloading module that will handle elementary functions such as $\sin$, $\cos$, $\log$, and $\exp$.

The tentative name attributes and methods for each class are listed below:

- ForwardAD:
  - Name attribute: Dpf
  - Methods: `__init__()`, `__call__()`
- DualNumber:
  - Name attributes: `real`, `dual`, `_supported_scalars`
  - Methods: `__init__()`, `__repr__()`, `__add__()`, `__mul__()`, `__radd__()`, `__rmul__()`
- function_overloads:
  - Methods: `sin()`, `cos()`, `tan()`, `exp()`, `log()`, `pow()`

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.
These vectors will be represented by `numpy` arrays.

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

License

This package will be devloped and released under the `MIT` license which is a copyleft.
The reasons behind choosing this license are that we want the software to be free and encourage others to contribute to open, public communities; while providing some degree of flexibility to developers like us.

Feedback

Milestone 1

Introduction(2/2):

- Would be great to write more in the intro. Elaborate on issues that exist with other techniques and the applications AD is useful for.

Differentiation, the process of finding a derivative, is one of the most fundamental operations in mathematics. It measures the rate of change of a function with respect to a variable. Computational techniques of calculating differentiations have broad applications in many fileds including science and engineering which used in finding a numerical solution of ordinary differential equations, optimization and solution of linear systems. Besides, they also have many real-life applications, like edge detection in image processing and safety tests of cars.

There are three popular ways to calculate the derivative:
1. Numerical Differentiation: Finite Difference
2. Symbolic Differentiation
3. Automatic Differentiation

Symbolic Differentiation and Finite Difference are two ways to numerically compute derivatives. Symbolic Differentiation is precise, but it can lead to inefficient code and can be costly to evaluate. Finite Difference is quick and easy to implement, but it can create round-off error, the loss of precision due to computer rounding of decimal quantities, and truncation error, the difference between the exact solution of the original differential equation.

Automatic Differentiation is more efficient than two of other methods mentioned prior. While it utilizes the concept of dual number, it achieves machine precision without costly evaluation, and therefore is widely used.

Background(2/2):

- Make sure to use the latex format correctly.
  - Noted and fixed (see above).
- idea of using an example to explain the introduction is nice but you need to elaborate/explain more Chain Rule, seed vectors, evaluation of forward trace, and how all these help in computing the derivatives. You should also make sure to be explicit about how the dual numbers will be useful (showing an example of the use of dual numbers for AD will help a lot).

1. Basic Calculus

- Product Rule
  Product rule is a formula used to find the derivatives of products of two or more functions. The product rule can be expressed as:

$$\frac{\partial}{\partial x}(uv) = u\frac{\partial v}{\partial x} + v\frac{\partial u}{\partial x}.$$

- Chain Rule
  Chain rule is a formula to compute the derivative of a composite function.
  The chain rule can be expressed as:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}.$$

2. Dual Numbers

- A dual number consists of two parts: *real* (denoted as a) and *dual* (b),
  usually written as

$$z = a + b\epsilon,$$

  where $\epsilon \neq 0$ is a nilpotent number with the property $\epsilon^2 = 0$.

3. Automatic Differentiation

- Automatic Differentiation refers to a general way of taking a program which computes a value, and automatically constructing a procedure for computing derivatives of that value. The derivatives sought may be first order (the gradient of a target function, or the Jacobian of a set of constraints), higher order (Hessian times direction vector or a truncated Taylor series), or nested. There are two modes in Automatic Differentiation: the forward mode and the reverse mode.

- *Elementary functions*: The set of elementary functions has to be given and can, in principle, consist of arbitrary functions as long as these are sufficiently often differentiable. All elementary functions will be implemented in the system together with their gradients.
- Evaluation Trace of a Function: All numeric evaluations are sequences of elementary operations. The evaluation of f at some point x = (x1, ..., xn) can be described by a so-called evaluation trace $v_{k-m} = x_k$, for $k = 1, 2, ..., m$, where each intermediate results v_j are functions that depend on the independent variables x.
- *Evaluating (forward) trace of a function*: All numeric evaluations are sequences of elementary operations. The evaluation of a function f at a given point $x = (x_1, \ldots, x_n)$ can be described by a so-called evaluation trace $v_0 = v_0(x), \ldots, v_m = v_m(x)$, where each intermediate result v_j is the result of an elementary operation and a function that depends on the independent variables x.

4. Forward Mode of Automatic Differentiation

- Forward automatic differentiation divides the expression into a sequence of differentiable elementary operations. The chain rule and well-known differentiation rules are then applied to each elementary operation.
- Forward automatic differentiation computes a tangent trace of the directional derivative

$$D_p v_j = (\nabla y_i)^T p = \sum_{j=1}^{m} \frac{\partial y_i}{x_j} p_j$$

  for each intermediate variable v_j at the same time as it performs a forward evaluation trace of the elementary pieces of a complicated $f(x)$ from the inside out.
  Note that the vector p is called the seed vector which gives the direction of the derivative.
- Implementation with dual numbers: by its properties, a dual number can encode the primal trace and the tangent trace in the real and dual parts, respectively.

$$z_j = v_j + D_p v_j \epsilon$$

How to use(3/3):

- Please do not submit versions that are not proofread. Pay attention to typos. Is not acceptable to have typos on the "import statement". Would be great to explain more about how the user can import the dependent packages.

This package is distributed through the Python Package Index (PyPI), and hence the user can install it with:

```
python -m pip install team20ad
```

In addition, they need to install and import all the dependable packages including `numpy`, `scipy`, `pandas`, and `matplotlib`.

To use the `team20ad` package, one can import the module by:

```
from team20ad.forward_ad import ad
```

The user will be able to instantiate an AD object and compute the differentation as follows:

```
f = some_function_to_be_differentiated
x = some_value_to_evaluate
ad_obj = ad() # instantiate an automatic differentiation object
res = ad_obj.forward(f, x) # compute derivative of f evaluated at x using forward mode AD
```

Software Organization(2/2):

- Also, use the tree to have a better visual for the directory structure.
- you should talk about the package distribution, the tests and where they will be located.
- You should mention the modules and what their functionality will be.
    - Please see complete changes above.
    - tree:

```
team20/
|-- docs/
|  |-- milestone1.md
|  \-- milestone1.pdf
|-- LICENSE
|-- README.md
|-- pyproject.toml
|-- .github/workflows/
|       |-- coverage.yml
|       \-- test.yml
|-- tests/
|       |-- check_coverage.sh
|       |-- run_tests.sh
|       |-- forward_ad/
|       |  |-- test_forward.py
|       |  \-- test_dual.py
|       \-- overloads/
|          \-- test_overloads.py
\-- src/
        \-- team20ad/
          |--   __init__.py
          |--   __main__.py
          |--   example.py
          |--   forward_ad/
          |    |-- __init__.py
          |    |-- forward.py
          |    \-- dualNumber.py
          \--   overloads/
                   |-- __init__.py
                   \-- function_overloads.py
```

Currently, we plan to have two modules: one for implementing the forward mode of automatic differentiation and the other for defining function overloads (more on this under the Implementation section). The `forward_ad` module will include an implementation of `DualNumber` class, which is necessarily for the forward mode computation.

Note that an implementation of computational graph is optional for forward AD.

As such, we will have correponding tests `test_forward.py` and `test_overloads.py`, which are located under the `tests/forward_ad` and `tests/overloads` directories, respectively, and which will be configured to run automatically using GitHub workflows after each push to the `main` branch of development.

Implementation(3/4):

- You need to elaborate more on the core data structure, the methods, and how you will overload the elementary operations.
- You should also mention what each class will do in more detail and possibly give an example.
- Elementary functions are another big topic you can write more about.
- You should talk more about the dual numbers and the reverse functions (e.g. radd).
  You should explain what each class will be doing and how.
- Big emphasis on how exactly you will implement it.
  - Please see the revised writing of this section above. The fundamental concepts of elementary functions and dual numbers were added to the Background section.

License(0/2):

- Please add the license and the reason for using those.
  - This package will be devloped and released under the `MIT` license which is a copyleft.
  - The reasons behind choosing this license are that we want the software to be free and encourage others to contribute to open, public communities; while providing some degree of flexibility to developers like us.