**How to Use FADiff**[Will and Gerald to research format of existing autodiff libraries]

We expect the use of our package, “FADiff”, to be largely through its API. Where necessary or practical, our API may permit the use of objects and functions from NumPy or other widely-used external libraries. However, for certain areas of our implementation, we expect our package to require the exclusive use of internally defined objects and methods. For example, we might prohibit users from using external libraries for elementary functions (e.g., sin and cos) and only allow them to use our package’s implementations for such functions. This may help to reduce the potential for issues further on in the development process such as disuse and misuse of our package’s operator-overloaded functions, among other things. We will be clear in our documentation on how the user should use our package’s API including the proper use of variables and methods. Nonetheless, only one header should be needed to import and use our entire package. An example demonstrating the use of our package in pseudocode is shown below:

import FADiff as ad

ad.do\_auto\_diff(function, point\_of\_evaluation)

a\_fxn = x - ad.exp(-2.0 \* ad.sin(4.0 \* x))

elementary\_derivative = ad.derive\_sin(x^2)

**Software Organization** [Will and Gerald to research]

* What will the directory structure look like?

|cs107project

+---src/

|---+---\_\_init\_\_.py

Fadiff/

\_\_init\_\_.py

Fadiff.py

includes/ or libs/

docs/

tests/

examples/

README.md

LICENSE.md

.travis.yml

setup.py

* What modules do you plan on including? What is their basic functionality?

Our FADiff package will contain a module named FADiff.py and another called ElemDerivs.py (short for “elementary derivatives”). FADiff.py will contain our main automatic differentiation class and ElemDerivs.py will contain functions that calculate the derivatives of all the elementary functions in our package such as sin and cos. ElemDerivs.py will be imported by FADiff.py. We also plan to use external module dependencies like NumPy or similar for handling matrices, vectors, and other linear algebra data types as well as for elementary functions. As explained in the “How to Use FADiff” section of this document, our implementation may need to limit the use of these external packages and/or only use them internally “under the hood” in certain areas.

* Where will your test suite live? Will you use TravisCI? CodeCov?

--TravisCI

* How will you distribute your package (e.g. PyPI)? [seriously refer to other resources]
  + Explanation of how to package, submit, and serve via PyPI: <https://packaging.python.org/tutorials/packaging-projects/>
  + I think we should go for this method as it supports just running the install script after cloning. If we’re going to do that then we should just make it work as a quick python3 pip install PACKAGE\_NAME
* How will you package your software? Will you use a framework? If so, which one and why? If not, why not?
  + just packaging as a series of scripts which is clonable via GitHub and one line installable via PyPI
* Other considerations?
  + Do we want to support some way of visualizing the operations that are going on such as the symbol tables we saw in class or compute graphs for forward/backward diff?
  + How “pretty” do we want our output to be? Console tables, colors, etc…
  + After finishing Homework 4 and some upcoming lectures on various software topics such as containers, we may consider revising our software organization later on.

**Implementation** [Will and Gerald to research]

Discuss how you plan on implementing the forward mode of automatic differentiation.

* What are the core data structures? [TBD]

We plan to use a list as our main data structure to hold the nodes / intermediate steps (including input parameters and output functions) of an evaluation trace. An element in the list would represent one node / intermediate step in a trace and may be a list itself or perhaps an instance of a class. It would also contain pertinent data for that particular node including what parent or child nodes it has if any. We are also considering using a tuple to hold the value of an elementary function at an intermediate step and the value of its derivative since they are handled together in forward mode. Other data structures we plan to use include a NumPy array, int, and float. Furthermore, we may discover uses for other data structures that could supplant or augment parts of our implementation later on such as a tree, hash map, or dictionary.

* What classes will you implement? [TBD]
  + An automatic differentiation class
  + A class to hold functions that calculate derivatives of elementary functions
* What method and name attributes will your classes have? [TBD]

Class [name]...

* + Attributes --
    - input values
      * function(s)
      * point(s) at which to evaluate the function(s)
    - seed vector
    - Jacobian
  + Methods --
    - \_\_init\_\_
    - Parsing
      * This may not be a needed method as we are hoping that our operator-overloading functions can handle this for us. If we find a need for it later on, we would maybe put this functionality in a class of its own.
    - do\_auto\_diff(function, point\_of\_evaluation)
      * The user will need to provide the function argument
      * We’ll have a default seed value if the user doesn’t include one in as an argument.
      * Returns the derivative at the point specified
    - look\_up\_node(node)
      * Parameters:
      * Returns:
    - dunder methods will be used for operator overloading so that our package’s objects can work with mathematical and any needed operations.
      * Parameters:
      * Returns:
    - The class that contains the functions for taking derivatives of elementary functions will have a method for each elementary function (e.g., sin, cos, etc.)
    - getters and setters
      * Can be used to access and change the seed vector. Will clarify with our client, David, on what requirements there are for handling the seed vector.
      * Can be used to return a particular node
      * Can be used to find an elementary operation in the evaluation trace

Class [name]…

* + Attributes
    - …
  + Methods
    - ...
* What external dependencies will you rely on? [TBD]
  + NumPy for linear algebra data types like matrices and vectors
  + Possibly Sphinx for auto-rendering our documentation
* How will you deal with elementary functions like sin, sqrt, log, and exp (and all the others)?
  + Our implementation’s API would contain functions for evaluating all of the elementary functions which could involve some operator overloading. Internally (i.e., “under the hood”), these functions will utilize NumPy or a similar external library for the evaluation of elementary functions. For calculating derivatives of elementary functions, we will create our own methods for evaluating them. If a function is not in our API then our package can’t recognize it and an error will be raised if such a function is used.
    - We expect our implementation to be similar to NumPy syntax in its use. e.g., instead of np.sin ours would be ad.sin where ad is the class instance of our auto package
      * See “how to use” section at beginning of document