4D Use Case: Utilities

## Overview

Some basic utility functions are provided in the “UTILITIES” directory, useful to multiple scripts in different phases.

## CSV Auto Reader

Reads a simple CSV (comma-separated text) file with a top-line header, auto-detects the data type in each column (text, int, float), and returns a numpy array with named columns containing the data in the CSV file. Typically used as a function call by other Python modules

Usage in Python:

```

def read\_simple\_csv(csv\_file, picklefile=None, print\_help=False, verbose=False, string\_length\_size=None):

Inputs:

csv\_file: path to an existing .csv text file to read.

picklefile (optional): path to an output dump file to which to save the Python numpy array (using the python ‘pickle’ library. (This is typically unused.)

print\_help (optional): Print a help message along with processing the file.

verbose (optional): If true, increase the verbosity of the output.

string\_length\_size (optional): Positive integer. By default, the numpy “string” fields are set to be equal to the length of the longest individual value in each field. For instance, if the longest string in the “name” field is 28 characters long, that numpy field is given the dtype “S28”. If the user prefers a longer string field in the numpy array, they can specify it here.

Output:

Numpy array with named columns (based upon the names of the headers) and fields of type “numpy.int”, “numpy.float” or “numpy.string”

```

Usage standalone:

```

usage: python CSV\_auto\_reader.py [-h] [--print\_help] [--verbose] csv\_file picklefile

Read a simple CSV file (1 header line, consistent N columns, unique column

names, no quoted fields, int/float/string values). Save the output to an

optional python picklefile, if provided.

positional arguments:

csv\_file Path to the CSV file to be read

picklefile An optional picklefile, to save the output

optional arguments:

-h, --help show this help message and exit

--print\_help Print help message explaining usage (ignores other arguments)

--verbose Increase output verbosity

```

## CSV Writer

Takes a numpy array with named columns, or a list of arrays and a separated list of column names, and writes the output to a CSV text file.

Usage in Python:

```

def write\_csv(data, csv\_file, header=None, verbose=False, print\_help=False):

Inputs:

data: a numpy array of values, with our without a set of named fields

csv\_file: path to a .csv text file to write out the data

header (optional): An iterable of field names to assign to each column of the data (useful if fields in ‘data’ are unnamed)

print\_help (optional): Print a help message

Output:

If successful, CSV document created containing data from ‘data’ variable

No return value.

```

Usage standalone:

*(Standalone usage not supported.)*

## Eprint

Simple function for printing output to stderr rather than stdout.

Usage in Python:

```

eprint(“achoo”)

* Simple wrapper function for outputting text to stderr instead of stdout.

```

Usage standalone:

*(Standalone usage not supported.)*

## Dependencies

The code requires Python 2.7+ to execute.

## License

(TODO: Fill in the open-source license to be used here.)

## Acknowledgements

This project is funded by the NSF’s ICEBERG Cyber-Infrastructure grant.

## Revisions:

1.0 – 2019.02.10 – Draft document by Mike MacFerrin