uarray: Expand unumpy overall coverage

Personal Information

Name: João Ferreira
University: Faculdade de Ciências da Universidade de Lisboa
Program: Master's degree in Data Science (2019 - 2021)
GitHub Username: joaosferreira
Timezone: GMT

About me

The journey that led me to apply to GSoC

It was during my bachelor's degree in Computer Science that I first had real contact with programming. I quickly developed a taste for programming, especially after some courses on data structures and algorithms. I remember being quite fascinated at how simple programming statements could be used to create higher concepts like queues and the many ways it was possible to sort elements.

Mid-way during my degree I started learning more about machine learning and its many possibilities. Also, I started reading some stuff about AI and how we were going to reach singularity soon. It was fascinating stuff and my mind was blown at the time. It sounded like a promising and interesting field, which made me want to get on the AI bandwagon. It was good motivation. At the time I imagined that I could apply this stuff in disease treatment and help the world somehow, which I still do, not that sure about the singularity part though.

I was introduced to GSoC by some of my colleagues that had participated. The idea of spending the summer contributing to open-source software and being paid was very appealing. I always wanted to participate but I never managed to apply because I would always get caught in university stuff and just excuse myself that I was too busy and didn't have enough time. Also it seemed like something quite challenging which I was not sure I was prepared for despite the encouragement of my colleagues. I would usually regret it later for not applying.

Since then I moved to a new university to pursue a master's in Data Science, where I am now, currently in my first year. In a strange kind of twist of fate, my current teachers' research focuses on machine learning techniques applied to healthcare which was what led

me here to begin with. Maybe I will actually help the world, who knows? Also, I've decided to take up on that old challenge and apply to GSoC.

What to I hope to get out of GSoC

For some time now my programming interests have been geared more towards machine learning and scientific computing in general. I'm always happy whenever I'm able to combine programming and some form of mathematics which is one of the reasons I like machine learning so much. I choose this project exactly because it's related to the core NumPy library which is something I use frequently and already have some familiarity with. This is a good opportunity for me to deepen my knowledge on this library and get to know other libraries for working with multi-dimensional arrays. This would also substantially improve my Python coding skills which is my bread-and-butter programming language.

In the process of applying for this project I experienced with open-source software development for the first time by doing my first PRs so in a way I already got something out of GSoC, and that is a good thing.

What happens after GSoC and overall future plans

I don't really believe in making plans on the long run but I always need to have some sense of where I'm going or else I feel lost. After the summer I'm supposed to start writing my thesis which, if all goes well, will be mentored by a professor in TUDelft, a university in the Netherlands. We still haven't discussed the theme but she does research on computational regulomics and cancer so It might be related to that, which makes me excited so I'm looking forward to that. After that I will probably look for a job in data science.

I also intend to keep contributing to open-source software as much as possible after the summer and hopefully keep close contact with my mentor organization.

Code contribution

Pull requests

Add math constants Pi and Euler's number #43

This PR added two math constants, pi and euler's number. This came out of necessity while writing tests for multimethods on universal functions. These tests are addressed in the PR below.

Test results of some universal functions #44

Added tests for some multimethods on common universal functions, more specifically, math operations (e.g., add, substract, multiply, divide, etc). My intention was to write some tests

for the derivatives computed by the udiff so these tests were originally targeted at that project as testbed for the other tests. I opened this PR after one of the code maintainers alerted me to the fact that these tests belonged in the unumpy project.

Add tests #5

This was the first PR that I opened which resulted in the one mentioned above. This didn't get merged as it was superseded by the mentioned PR.

Issues

Add math constants Pi and Euler's number #42

This issue is related to the PR of the same name.

Add some logical functions #45

This issue addressed the need for more appropriate functions for comparing floating-point values. This resulted in a PR that added multimethods that covered these logic functions which at the time were needed for the tests mentioned above.

Missing return statement #52

While studying the codebase for the unumpy project I encountered a function which didn't have a return statement. It was fixed soon after by one of the code maintainers.

Add tests #4

This was the first issue that I opened. I initially intended to work on the udiff project but I ended up changing my focus to the unumpy project. This issue was closed by some PRs done by other GSoC applicants.

Project information

Sub-org name: uarray

Project Abstract

My project proposal is to expand the overall coverage of the unumpy API. unumpy is an API for array computing that intends to follow in close resemblance the core NumPy API and that is built on top of the uarray library. It's mission is to hopefully become a generic backend system for the core NumPy API and provide other libraries that are built on top of unumpy the benefits of being able to run against different backend implementations in an easy and versatile way. Currently the unumpy library is incomplete as a mirror for the core NumPy API so I intend to work on expanding its current coverage by adding around 150 function stubs and a JAX backend.

Timeline

Community Bonding Period (May 4 - June 1)

During this period I intend to build a greater familiarity with the project's core libraries, uarray and unumpy, and the other array libraries used as backends. Also take time to talk with my mentor/s, fellow GSoC colleagues and the array community.

Coding Period

First Coding Round (June 1 - July 3)

This round I will focus on writing multimethods that target different parts of the core NumPy API. Each multimethod is normally defined by two functions, a default implementation (not always required) and a function stub for the multimethod. The default implementation is passed as an argument to the stub which itself is decorated by the create_multimethod partial function defined in uarray. The decorator takes as the first positional argument an argument replacer for the decorated function.

An example of a multimethod:

```
def _isclose_default(a, b, rtol=1e-05, atol=1e-08, equal_nan=False):
    ret = absolute(a - b) <= (atol + rtol * absolute(b))
    if equal_nan:
        ret |= isnan(a) & isnan(b)

    return ret



@create_numpy(_first2argreplacer, default=_isclose_default)
@all_of_type(ndarray)
def isclose(a, b, rtol=1e-05, atol=1e-08, equal_nan=False):
    return a, b
```

Code written by hameerabbasi.

Work for each week will be focused on a specific group of routines according to its functionality.

Week 1 (June 1 - June 5)

Add missing mathematical constants and their aliases and refactor the existing ones (these were written by me in a previous PR). Most of these constants should be defined as

singleton classes. Also there are some functions for checking class equality in array elements that need their respective multimethods to be defined.

Constants

- **inf (Inf, Infinity, PINF, infty):** IEEE 754 floating point representation of (positive) infinity.
- **nan (NaN, NAN):** IEEE 754 floating point representation of Not a Number (NaN).
- **NINF:** IEEE 754 floating point representation of negative infinity.
- **NZERO:** IEEE 754 floating point representation of negative zero.
- **PZERO:** IEEE 754 floating point representation of positive zero.
- **e:** Euler's constant, base of natural logarithms, Napier's constant.
- **euler_gamma:** γ = 0.57721566490153286060651209008240243104 21...
- **newaxis:** A convenient alias for None, useful for indexing arrays.
- **pi:** pi = 3.14159265358979323846264 33...

Note: These definitions should live in a separate file (e.g., _math_constants.py).

Logic functions

Array contents

- **isinf:** Shows which elements are positive or negative infinity.
- **isposinf:** Shows which elements are positive infinity.
- **isneginf:** Shows which elements are negative infinity.

Array type testing

- **iscomplex:** Returns a bool array, where True if input element is complex.
- **iscomplexobj**: Check for a complex type or an array of complex numbers.
- **isfortran:** Check if the array is Fortran contiguous but not C contiguous.
- **isreal:** Returns a bool array, where True if input element is real.
- **isrealobj:** Return True if input is not a complex type or an array of complex numbers.
- **isscalar:** Returns True if the type of the input is a scalar type.

Week 2 (June 8 - June 12)

Add multimethods for array creation.

Ones and zeros

- **empty:** Return a new array of given shape and type, without initializing entries.
- **empty_like:** Return a new array with the same shape and type as a given array.
- **identity:** Return the identity array.
- **ones_like:** Return a new array of given shape and type, filled with ones.
- **zeros_like:** Return an array of zeros with the same shape and type as a given array.
- **full_like:** Return a full array with the same shape and type as a given array.

From existing data

- **asanyarray:** Convert the input to an ndarray, but pass ndarray subclasses through.

- **ascontiguousarray:** Return a contiguous array (ndim >= 1) in memory (C order).
- **copy:** Return an array copy of the given object.
- **frombuffer:** Interpret a buffer as a 1-dimensional array.
- **fromfile:** Construct an array from data in a text or binary file.
- **fromfunction:** Construct an array by executing a function over each coordinate.
- **fromiter:** Create a new 1-dimensional array from an iterable object.
- **fromstring:** A new 1-D array initialized from text data in a string.
- **loadtxt:** Load data from a text file.

Week 3 (June 15 - June 19)

Add more multimethods for array creation.

Numerical ranges

- **geomspace:** Return numbers spaced evenly on a log scale (a geometric progression).
- **mgrid:** nd_grid instance which returns a dense multi-dimensional “meshgrid”.
- **ogrid:** nd_grid instance which returns an open multi-dimensional “meshgrid”.

Building matrices

- **diag:** Extract a diagonal or construct a diagonal array.
- **diagflat:** Create a two-dimensional array with the flattened input as a diagonal.
- **tri:** An array with ones at and below the given diagonal and zeros elsewhere.
- **tril:** Lower triangle of an array.
- **triu:** Upper triangle of an array.
- **vander:** Generate a Vandermonde matrix.

Week 4 (June 22 - June 26)

Add multimethods for array manipulation.

Basic operations

- **copyto:** Copies values from one array to another, broadcasting as necessary.

Changing number of dimensions

- **broadcast:** Produce an object that mimics broadcasting.
- **expand_dims:** Expand the shape of an array.
- **squeeze:** Remove single-dimensional entries from the shape of an array.

Changing kind of array

- **asfarray:** Return an array converted to a float type.
- **asfortranarray:** Return an array (ndim >= 1) laid out in Fortran order in memory.
- **asarray_chkfinite:** Convert the input to an array, checking for NaNs or Infs.
- **asscalar:** Convert an array of size 1 to its scalar equivalent.
- **require:** Return an ndarray of the provided type that satisfies requirements.

Joining arrays

- **dstack:** Stack arrays in sequence depth wise (along third axis).

Splitting arrays

- **split:** Split an array into multiple sub-arrays.
- **array_split:** Split an array into multiple sub-arrays.
- **dsplit:** Split array into multiple sub-arrays along the 3rd axis (depth).
- **hsplit:** Split an array into multiple sub-arrays horizontally (column-wise).
- **vsplit:** Split an array into multiple sub-arrays vertically (row-wise).

Week 5 and First Evaluation (June 29 - July 3)

Add more multimethods for array manipulation.

Tiling arrays

- **tile:** Construct an array by repeating A the number of times given by reps.
- **repeat:** Repeat elements of an array.

Adding and removing elements

- **delete:** Return a new array with sub-arrays along an axis deleted.
- **insert:** Insert values along the given axis before the given indices.
- **append:** Append values to the end of an array.
- **resize:** Return a new array with the specified shape.
- **trim_zeros:** Trim the leading and/or trailing zeros from a 1-D array or sequence.

Rearranging elements

- **flip:** Reverse the order of elements in an array along the given axis.
- **fliplr:** Flip array in the left/right direction.
- **flipud:** Flip array in the up/down direction.
- **reshape:** Gives a new shape to an array without changing its data.
- **roll:** Roll array elements along a given axis.
- **rot90:** Rotate an array by 90 degrees in the plane specified by axes.

Deliverables

- Math constants
- Multimethods for logic functions
- Multimethods for array creation
- Multimethods for array manipulation

Second Coding Round (July 3 - July 31)

During this round I intend to build upon the work started in the previous round by covering more functions of the core NumPy API.

Week 6 (July 6 - July 10)

Add multimethods for mathematical functions.

Trigonometric functions

- **degrees:** Convert angles from radians to degrees.
- **radians:** Convert angles from degrees to radians.
- **unwrap:** Unwrap by changing deltas between values to 2*pi complement.

Rounding

- **around:** Evenly round to the given number of decimals.
- **round_:** Round an array to the given number of decimals.
- **fix:** Round to nearest integer towards zero.

Sums, products, differences

- **cumprod:** Return the cumulative product of elements along a given axis.
- **cumsum:** Return the cumulative sum of the elements along a given axis.
- **nancumprod:** Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one.
- **nancumsum:** Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
- **ediff1d:** The differences between consecutive elements of an array.
- **cross:** Return the cross product of two (arrays of) vectors.
- **trapz:** Integrate along the given axis using the composite trapezoidal rule.

Other special functions

- **i0:** Modified Bessel function of the first kind, order 0.
- **sinc:** Return the sinc function.

Week 7 (July 13 - July 17)

Add more multimethods for mathematical and functional programming functions.

Arithmetic operations

- **float_power:** First array elements raised to powers from second array, element-wise.

Handling complex numbers

- **angle:** Return the angle of the complex argument.
- **real:** Return the real part of the complex argument.
- **imag:** Return the imaginary part of the complex argument.
- **conjugate:** Return the complex conjugate, element-wise.

Miscellaneous

- **convolve:** Returns the discrete, linear convolution of two one-dimensional sequences.
- **clip:** Clip (limit) the values in an array.

- **nan_to_num:** Replace NaN with zero and infinity with given finite numbers.
- **real_if_close:** If complex input returns a real array if complex parts are close to zero.
- **interp:** One-dimensional linear interpolation.

Functional programming

- **apply_along_axis:** Apply a function to 1-D slices along the given axis.
- **apply_over_axes:** Apply a function repeatedly over multiple axes.
- **vectorize:** Generalized function class.
- **frompyfunc:** Takes an arbitrary Python function and returns a NumPy ufunc.
- **piecewise:** Evaluate a piecewise-defined function.

Week 8 (July 20 - July 24)

Add multimethods for indexing routines.

Generating index arrays

- **np.c_:** Translates slice objects to concatenation along the second axis.
- **r_:** Translates slice objects to concatenation along the first axis.
- **s_:** A nicer way to build up index tuples for arrays.
- **where:** Return elements chosen from one of the inputs depending on a condition.
- **indices:** Return an array representing the indices of a grid.
- **ix_:** Construct an open mesh from multiple sequences.
- **ravel_multi_index:** Converts a tuple of index arrays into an array of flat indices, applying boundary modes to the multi-index.
- **unravel_index:** Converts a flat index or array of flat indices into a tuple of coordinate arrays.
- **diag_indices:** Return the indices to access the main diagonal of an array.
- **diag_indices_from:** Return the indices to access the main diagonal of an n-dimensional array.
- **mask_indices:** Return the indices to access (n, n) arrays, given a masking function.
- **tril_indices:** Return the indices for the lower-triangle of an (n, m) array.
- **tril_indices_from:** Return the indices for the lower-triangle of a given array.
- **triu_indices:** Return the indices for the upper-triangle of an (n, m) array.
- **triu_indices_from:** Return the indices for the upper-triangle of a given array.

Week 9 and Second Evaluation (July 27 - July 31)

Add more multimethods for indexing routines.

Indexing-like operations

- **take:** Take elements from an array along an axis.
- **take_along_axis:** Take values from the input array by matching 1d index and data slices.
- **choose:** Construct an array from an index array and a set of arrays to choose from.
- **diagonal:** Return specified diagonals.
- **select:** Return an array drawn from elements in a list of choices, depending on given conditions.

- **lib.stride_tricks.as_strided:** Create a view into the array with the given shape and strides.

Inserting data into arrays

- **place:** Change elements of an array based on conditional and input values.
- **put:** Replaces specified elements of an array with given values.
- **put_along_axis:** Put values into the destination array by matching 1d index and data slices.
- **putmask:** Changes elements of an array based on conditional and input values.
- **fill_diagonal:** Fill the main diagonal of the given array of any dimensionality.

Iterating over arrays

- **nditer:** Efficient multi-dimensional iterator object to iterate over arrays.
- **ndenumerate:** Multidimensional index iterator.
- **ndindex:** An N-dimensional iterator object to index arrays.
- **nested_iters:** Create nditers for use in nested loops
- **flatiter:** Flat iterator object to iterate over arrays.
- **lib.Arrayterator:** Buffered iterator for big arrays.

Deliverables

- Multimethods for mathematical functions.
- Multimethods for functional programming functions.
- Multimethods for indexing routines.

Third Coding Round (July 31 - August 28)

In the first week of the final round of coding I intend to finish the work developed in the prior rounds by writing the last multimethods of the project. After that I move on to write a backend provider for the JAX library. This work will be scheduled for the second week of this round and after that I will focus the rest of the time wrapping up the project by organizing all multimethods in appropriate modules and writing tests for the new multimethods.

Week 10 (August 3 - August 7)

Add multimethods for statistics functions.

Order statistics

- **amin:** Return the minimum of an array or minimum along an axis.
- **amax:** Return the maximum of an array or maximum along an axis.
- **percentile:** Compute the q-th percentile of the data along the specified axis.
- **nanpercentile:** Compute the qth percentile of the data along the specified axis, while ignoring nan values.
- **quantile:** Compute the q-th quantile of the data along the specified axis.
- **nanquantile:** Compute the qth quantile of the data along the specified axis, while ignoring nan values.

Averages and variances

- **median:** Compute the median along the specified axis.
- **average:** Compute the weighted average along the specified axis.
- **mean:** Compute the arithmetic mean along the specified axis.
- **nanmedian:** Compute the median along the specified axis, while ignoring NaNs.
- **nanmean:** Compute the arithmetic mean along the specified axis, ignoring NaNs.
- **nanstd:** Compute the standard deviation along the specified axis, while ignoring NaNs.
- **nanvar:** Compute the variance along the specified axis, while ignoring NaNs.

Correlating

- **corrcoef:** Return Pearson product-moment correlation coefficients.
- **correlate:** Cross-correlation of two 1-dimensional sequences.
- **cov:** Estimate a covariance matrix, given data and weights.

Histograms

- **histogram:** Compute the histogram of a set of data.
- **histogram2d:** Compute the bi-dimensional histogram of two data samples.
- **histogramdd:** Compute the multidimensional histogram of some data.
- **bincount:** Count number of occurrences of each value in array of non-negative ints.
- **histogram_bin_edges:** Function to calculate only the edges of the bins used by the histogram function.
- **digitize:** Return the indices of the bins to which each value in input array belongs.

Week 11 (July 10 - July 14)

Add a JAX backend provider. JAX is a library that uses XLA to compile and run NumPy programs on GPUs and TPUs.

A backend provider makes use of three protocols:

- **__ua_domain__:** String containing the domain of the backend (e.g., numpy).
- **__ua_function__:** Function that defines the implementation of a multimethod by checking if an implementation for the given method exists in the _implementations dictionary of that backend.
- **__ua_convert__:** Function for converting a dispatched argument. This function is applied to all arguments by its wrapper.

The first two protocols are mandatory and the last one is optional.

Example of protocols for the Dask backend:

```
__ua_domain__ = "numpy"



def __ua_function__(method, args, kwargs):
    if method in _implementations:
        return _implementations[method](*args, **kwargs)

    if not hasattr(da, method.__name__):
        return NotImplemented

    return getattr(da, method.__name__)(*args, **kwargs)



@wrap_single_convertor
def __ua_convert__(value, dispatch_type, coerce):
    if dispatch_type is ndarray:
        if not coerce:
            return value
        return da.asarray(value) if value is not None else None

    return value
```

Code written by hameerabbasi.

Week 12 (August 17 - August 21)

Organize multimethods in groups given their functionality into appropriate modules and start writing tests for the new multimethods. Also integrate JAX into the tested backends.

Final week, Code Submission and Final Evaluation (August 24 - August 28)

Try to cover as many multimethods as possible with tests, finish the project and submit.

Other commitments

Exams

Given the current pandemic situation, all presential classes in Portugal have been suspended indefinitely and have been substituted by remote classes. All evidence points to classes not returning to normal during the rest of the academic year and there is still debate on how the end of semester evaluations will be carried out.

As of now the period of exams goes from June 3 to July 4. This coincides with the first coding round of GSoC. Nevertheless, I will do my best to commit 30+ hours per week during this time and, if necessary, compensate in the following rounds. I am really looking forward to participating in GSoC and if given this opportunity I would live and breathe this project.

Thank you for taking the time to read this.