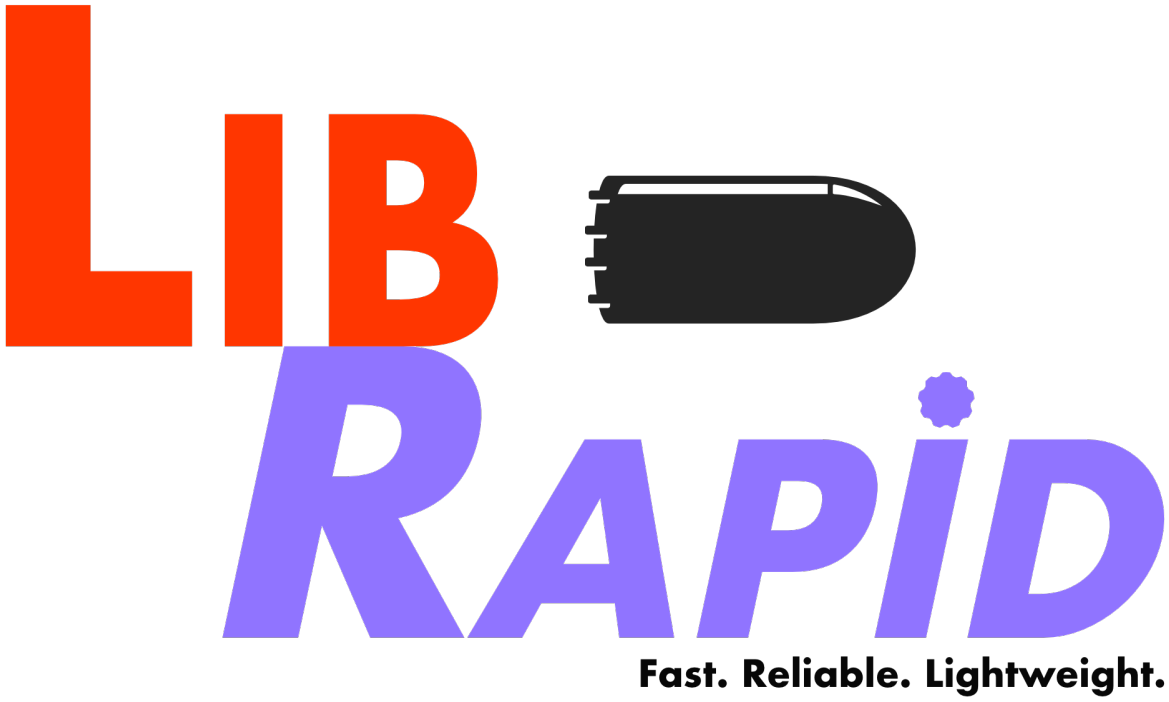
librapid

Release 0.2.7

Toby Davis

Sep 11, 2021

CONTENTS



LibRapid aims to provide fast and intuitive interface to mathematical functions and types, such as multi-dimensional arrays and neural networks.

LibRapid hopes to provide a functionally and interactively consistant interface in both C++ and Python, allowing high level code to run faster and more efficiently, while also ensuring low-level programs in C++ can be optimized in the same way.

CHAPTER

ONE

WHAT LIBRAPID IS

LibRapid is a lightweight alternative to other popular python libraries, intended to be used in both C++ and Python, with potential support for CUDA coming in the future. It is designed to be as easy to use as possible while still supporting advanced functionality and high performance.

CHAPTER

TWO

WHAT LIBRAPID IS NOT

A replacement for the well-established Python and C++ libraries that already exist. It can be used in a wide range of applications, but there are no guarantees that all functions perform exactly the correct calculations in all situations, though increasing numbers of tests are being implemented, and any bugs discovered are fixed as quickly as possible.

CHAPTER

THREE

LICENCING

LibRapid is produced under the Boost Software License, so you are free to `use, reproduce, display, distribute, execute and transmit` the software, though this is subject to some conditions, which can be found in full here: LibRapid License

CHAPTER

FOUR

INSTALLING THE PACKAGE

To install the Python package, simply open a command line window and type

```
pip install librapid
```

To install the library for C++ useage simply download the code and either copy the files to your program directory, or save them somewhere memorable (such as `C:\opt\librapid`) and add them to the include direcotires of your project.

The simplest way to download the code is via Git, using the command

```
git clone https://github.com/Pencilcaseman/librapid.git
```

CHAPTER

FIVE

USING LIBRAPID

Important: To use LibRapid in Python, a modern C++ compiler may be needed to compile the code during installation. To use LibRapid in a C++ program, a modern C++ compiler will definitely be needed.

LibRapid is intended to be incredibly easy to use in both C++ and Python. To include the library in one of your projects, simply do the following:

Listing 1: For Python programs

```
import librapid
```

Listing 2: For C++ programs

```
#include <librapid/librapid.hpp>
```

CHAPTER

SIX

USING THE DOCUMENTATION

The documentation outlines the explicit declerations for the C++ functions themselves, which may be difficult to understand for many users. Luckily, however, only a small portion of the function definitions is actually relevant to most users.

For example, take the function definition below:

template<typename **T**>
inline double `librapid::test_function`(**const** test_struct *&thing*) **const**

It may appear complicated at first, but there are only a few key things to look out for. First, the name of the function (`test_function`) will be the same in the Python and C++ libraries, except in Python, the function will be accessed as `librapid.test_function`. Next, the arguments to the function are important to know, and should be self-explanatory. In C++, the type can be seen in the function definition, and passing the same type should give error-free code. In Python, however, things can be slightly different – often, Python values can be cast into equivelent C++ types, though the types are not always clear. Below is a list of common datatypes that you may want to pass to functions:

Type of value	C++ Datatype	Python Datatype
List	`std::vector<...>`	`list, tuple`
Integer	`lr_int, int, long`	`int`
Dictionary	`std::map<a, b>`	`dict`

CHAPTER

SEVEN

CONTENTS

7.1 LibRapid NDarray

7.1.1 What is an NDarray?

An NDarray is a multi-dimensional, homogeneous collection of values which can be operated on in a variety of ways.

The simplest type of NDarray is a vector, which is a list of values. Another simple type of NDarray is a matrix, which is a grid of values.

```
// This is a vector with 3 elements
[1 2 3]

// This is a 2x3 matrix
[[1 2 3]
 [4 5 6]]
```

The LibRapid NDarray can store arrays of any dimension, though the default limit is 32 dimensions.

Warning: While it is possible to change the maximum number of dimensions an array can contain, it is not recommended due to the extraordinary amount of memory that would be required to store such an array.

7.1.2 Why use an NDarray?

The LibRapid NDarray class implements extremely optimized and efficent algorithms for many mathematical operations, such as element-wise arithmetic, dot-products, transpositions and more.

Because of this optimization, they can be used in high-intensity situations, such as in the LibRapid neural network library, without compromising the speed of the program or the range of functions available.

Additionally, when using LibRapid in C++, it is incredibly easy to manipulate the NDarray type to fit you needs, as it is fully templated and works with a wide range of datatypes, with many functions supporting cross-datatype operations.

7.1.3 How are the arrays stored?

The underlying memory of each array is stored in a contiguous memory block. To access the different elements of an array, they also store the dimensions of the array, and a set of strides which specify how far through memory one must move to increment by one value in a given axis.

The fact that the arrays are stored in this way means that many functions can be accelerated dramatically by reducing the amount of data that must be transferred. For example, to transpose an array, the stride and extent are simply reversed.

7.1.4 Documentation

Array Constructors

LibRapid arrays can be made in a wide variety of ways, allowing greater functionality and flexibility when using the library.

Some of the constructors create entirely new data, where nothing is referencing it, while others create arrays that reference the data of another, so changing a value in one will also change the value in the other.

When a new array is created with entirely new data, the memory itself is contiguous (i.e. in a single block) in memory, meaning it's incredibly fast to access and do calculations with.

Any function which takes a librapid *ndarray* object can also take a python list, tuple or scalar, which will be automatically cast to the ndarray type. This makes your code more concise and accelerates development.

Hint: If you want to use an array that isn't contiguous (e.g. it was reshaped or is part of a larger array), the `clone()` function will copy the data, but make it optimal in memory, improving performance where applicable.

---

ndarray()

Warning: doxygenfunction: Cannot find function "librapid::basic_ndarray::basic_ndarray" in doxygen xml output for project "librapid" from directory: ./doxygenoutput/xml

ndarray(shape)

Warning: doxygenfunction: Cannot find function "librapid::basic_ndarray::basic_ndarray" in doxygen xml output for project "librapid" from directory: ./doxygenoutput/xml

ndarray(shape, fill)

Warning: doxygenfunction: Cannot find function "librapid::basic_ndarray::basic_ndarray" in doxygen xml output for project "librapid" from directory: ./doxygenoutput/xml

ndarray(ndarray)

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::basic_ndarray” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

ndarray(data)

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::basic_ndarray” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::from_data” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::from_data” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

zeros_like, ones_like, random_like

Warning: doxygenfunction: Cannot find function “librapid::zeros_like” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

---

Warning: doxygenfunction: Cannot find function “librapid::ones_like” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

---

Warning: doxygenfunction: Cannot find function “librapid::random_like” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

linear(start, stop, len)

Warning: doxygenfunction: Cannot find function “librapid::linear” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

range(start, stop, inc)

Warning: doxygenfunction: Cannot find function "librapid::range" in doxygen xml output for project "librapid" from directory: ./doxygenoutput/xml

Array Arithmetic

Basic arithmetic is a key part of any array library, and these operations are often used extensively throughout programs. For this reason, a lot of work has been done to optimise the unerlying routines that support the arithmetic operations to make them as fast as possible. To make them even faster, LibRapid may use multiple threads to perform the calculations if they are available.

In both the Python and C++ libraries, the arithmetic operators `+ - * /` are all overloaded, allowing you to do things such as `(a + b) / c`, though the following methods are also implemented:

1. add(a, b)
2. sub(a, b)
3. mul(a, b)
4. div(a, b)

How arithmetic operations work

Arithmetic operations on arrays are all performed element-wise, meaning the arrays must conform to specific rules regarding their extents (see array_shapes)

For example, adding two arrays works as follows:

```
[1 2 3] + [4 5 6] = [5 7 9]
# Because 1 + 4 = 5
#     and 2 + 5 = 7
#     and 3 + 6 = 9
```

Array extents and formats

Librapid provides differnent methods for adding two arrays, which vary based on the extents of the input arrays.

A full list of the arithmetic types is below:

Name	Requirement	Example
Exact match	The arrays are identical in their extents	`[[1 2] + [[5 6] = [[ 6 8]` `[3 4]] [7 8]] [10 12]]`

Outer dimensions match	The arrays have the same extent, but have leading or trailing 1s	`[1 2] + [[[3 4]]] = [4 6]`
		
		`[[[1 2]]] + [3 4] = [[[4 6]]]`

Single value array	One of the addends is an array with only a single value	`[1 2 3] + [10] => [11 12 13]`
		
		`[10] + [1 2 3] => [11 12 13]`

Row-by-row	The dimensions `[0 .. n-1]` of one array match the dimensions of the other	`[[1 2] + [5 6] = [[ 6 8]` `[3 4]] [ 8 10]]`
Grid	Array with extent `[... 1]` and array with extent `[1 ...]`	`[1 2] + [[3] = [[4 5]` `[4]] [5 6]]`
Column-by-column	The dimensions `[1 .. n]` of one array match the dimensions `[0 .. m-1]` of the other	`[[1 2] + [[5] = [[ 6 7]` `[3 4]] [6]] [ 9 10]]`

Attention: In the Python library, the `true_div` function is overloaded, not the standard `div` (integer division) function. To perform integer division, please use `floor(a / b)`

add

Warning: doxygenfunction: Cannot find function “librapid::add” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

sub

Warning: doxygenfunction: Cannot find function “librapid::sub” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

mul

Warning: doxygenfunction: Cannot find function “librapid::mul” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

div

Warning: doxygenfunction: Cannot find function “librapid::div” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

array + value

array + array

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator+” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator+=” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

array + scalar

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator+” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator+=” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

array - value

array - array

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator-” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator-=” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

array - scalar

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator-” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator-=” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

array * value

array * array

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator*” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator*=” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

array * scalar

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator*” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator*=” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

array / value

array / array

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator/” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator/=” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

array / scalar

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator/” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator/=” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

negate

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::operator-” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

Array Manipulation

LibRapid arrays support powerful manipulation routines which support data permanence (as far as possible) and optimized implementations. They allow for complex algorithms to be implemented easily, and data accesses to be simplified dramatically.

Attention: Some manipulation routines do **NOT** result in data permanence, and will result in a copied array. Please ensure you are aware of this, as it may lead to undesired results

reshape

Warning: doxygenfunction: Cannot find function “librapid::basic_ndarray::reshape” in doxygen xml output for project “librapid” from directory: ./doxygenoutput/xml

7.2 LibRapid Neural Network

7.2.1 Documentation

INDEX

L