NumPy style arrays for C++? [closed]

Asked 8 years, 11 months ago Active 4 months ago Viewed 66k times

91

30

Closed. This question does not meet Stack Overflow guidelines. It is not currently accepting answers.

Want to improve this question? Update the question so it's on-topic for Stack Overflow.

Closed 6 months ago.

Improve this question

Are there any C++ (or C) libs that have NumPy-like arrays with support for slicing, vectorized operations, adding and subtracting contents element-by-element, etc.?

c++ arrays dynamic-arrays

Share Follow

asked Jun 23 '12 at 12:15
Llamageddon
2,679 ● 2 ● 17 ● 41

2 Armadillo? – Oliver Charlesworth Jun 23 '12 at 12:17

1 As far as I know numpy uses LAPACK. While that is written in Fortran, there are c++ bindings available. Never used either of those though. – Voo Jun 23 '12 at 12:54

slower than using a C/C++ library. – Kevin Jun 13 '15 at 6:28

13 Answers

Active Oldest Votes

67

Here are several free software that may suit your needs.

1. The GNU Scientific Library is a GPL software written in C. Thus, it has a C-like allocation and way of programming (pointers, etc.). With the GSLwrap, you can have a C++ way of programming, while still using the GSL. GSL has a BLAS implementation, but you can use ATLAS instead of the default CBLAS, if you want even more performances.
2. The boost/uBLAS library is a BSL library, written in C++ and distributed as a boost package. It is a C++-way of implementing the BLAS standard. uBLAS comes with a few linear algebra functions, and there is an experimental binding to ATLAS.
3. eigen is a linear algebra library written in C++, distributed under the MPL2 license (starting from version 3.1.1) or LGPL3/GPL2 (older versions). It's a C++ way of programming, but more integrated than the two others (more algorithms and data structures are available). Eigen claim to be faster than the BLAS implementations above, while not following the de-facto standard BLAS API. Eigen does not seem to put a lot of effort on parallel implementation.
4. Armadillo is LGPL3 library for C++. It has binding for LAPACK (the library used by numpy). It uses recursive templates and template meta-programming, which is a good point (I don't know if other libraries are doing it also?).
5. xtensor is a C++ library that is BSD licensed. It offers A C++ API very similar to that of NumPy. See https://xtensor.readthedocs.io/en/latest/numpy.html for a cheat sheet.

These alternatives are really good if you just want to get data structures and basic linear algebra. Depending on your taste about style, license or sysadmin challenges (installing big libraries like LAPACK may be difficult), you may choose the one that best suits your needs.

Share Follow

edited Sep 18 '20 at 15:39
Caio S. Souza
91 ● 1 ● 11

answered Jun 23 '12 at 13:10
nojhan
1,042 ● 9 ● 11

21 Sadly, none of these provide anything as general and convenient as numpy arrays. Numpy arrays are arbitrary-dimensional and support things like `a[:4,::-1,:,19] = b[None,-5:,None]` or `a[a>5]=0` and similar, as well as having a huge set of array and index manipulation functions available. I really hope somebody makes something like that for C++ some day. – amaurea Mar 1 '14 at 11:56

4 OpenCV also has a Matrix type that can have arbitrary dimensional size; column/row ranges (`a.colRange(4,7).rowRange(4,8)` for `a[4:7,4,8]`) and condition mask (`a.setTo(cv::Scalar(0), a>5)` for `a[a>5]=0`) – xaedes Dec 15 '16 at 16:39

3 @amaurea check out the answer on xtensor below, which enables all of the above. – Quant May 14 '17 at 20:59

1 I've had to use Eigen in a recent project and I have to say that while it seems to be efficient, the syntax is absolutely terrible. There is none of that amazing Python slicing syntax available. For example, if you have a 1D vector x and want to manipulate the first n elements, you have to use x.head(n). Don't even ask about manipulating an arbitrary slice of x, you'll need a good old for-loop to do that. This is just one of the many clunky and inconvenient examples I could name. – Alex May 30 '17 at 23:34

57

Try out xtensor. (See the NumPy to Xtensor Cheat Sheet).

xtensor is a C++ library meant for numerical analysis with multi-dimensional array expressions.

xtensor provides

- an extensible expression system enabling numpy-style broadcasting.
- an API following the idioms of the C++ standard library.
- tools to manipulate array expressions and build upon xtensor.

Example

Initialize a 2-D array and compute the sum of one of its rows and a 1-D array.

```
#include <iostream>
#include "xtensor/xarray.hpp"
#include "xtensor/xio.hpp"

xt::xarray<double> arr1
```

```
xt::xarray<double> res = xt::view(arr1, 1) + arr2;

std::cout << res;
```

Outputs

```
{7, 11, 14}
```

Initialize a 1-D array and reshape it inplace.

```
#include <iostream>
#include "xtensor/xarray.hpp"
#include "xtensor/xio.hpp"

xt::xarray<int> arr
  {1, 2, 3, 4, 5, 6, 7, 8, 9};

arr.reshape({3, 3});

std::cout << arr;
```

Outputs

```
{{1, 2, 3},
 {4, 5, 6},
 {7, 8, 9}}
```

Share Follow

edited Dec 3 '17 at 8:39

answered Nov 6 '16 at 11:23

Quant
1,393 ● 13 ● 21

all work fine. On the other hand, it is still *very* experimental and many features haven't been implemented yet.

7

Here's a simple implementation of the de Casteljau algorithm in C++ using DyND arrays:

```
#include <iostream>
#include <dynd/array.hpp>

using namespace dynd;

nd::array decasteljau(nd::array a, double t){
    size_t e = a.get_dim_size();
    for(size_t i=0; i < e-1; i++){
        a = (1.-t) * a(irange()<(e-i-1)) + t * a(0<irange());
    }
    return a;
}

int main(){
    nd::array a = {1., 2., 2., -1.};
    std::cout << decasteljau(a, .25) << std::endl;
}
```

I wrote a blog post a little while back with more examples and side-by-side comparisons of the syntax for Fortran 90, DyND in C++, and NumPy in Python.

Disclaimer: I'm one of the current DyND developers.

Share Follow

edited May 19 '16 at 22:14

answered Oct 27 '15 at 17:48

IanH
8,740 ● 1 ● 26 ● 31

Eigen is a good linear algebra library.

3 http://eigen.tuxfamily.org/index.php?title=Main_Page

you need?

Share Follow

answered Jun 23 '12 at 12:20
Frédéric Terrazzoni
1,990 ● 16 ● 25

3 The syntax of Eigen is pretty terrible though. There is none of that smooth slicing syntax you find in Numpy. And it's not a general n-dimensional array library, it's more for 1D vectors and 2D matrices only. The fact that they have VectorXd for 1D arrays and MatrixXd for 2D arrays repels me already. – Alex May 30 '17 at 23:50

3

This is an old question. Still felt like answering. Thought might help many, Especially pydevs coding in C++.

If you have already worked with python numpy, then NumCpp is a great choice. It's minimalistic in syntax and has got similar functions or methods as py numpy.

The comparison part in the readme doc is also very very cool.

> NumCpp

```
nc::NdArray<int> arr = {{4, 2}, {9, 4}, {5, 6}};
arr.reshape(5, 3);
arr.astype<double>();
```

Share Follow

answered Oct 12 '20 at 17:22
Sayan Dey
618 ● 1 ● 9

If you want to use multidimensional array(like numpy) for image processing or neural network, you can use `OpenCV` `cv::Mat` along vectors with real or complex values, other matrices etc.

A Mat contains the following information: `width, height, type, channels, data, flags, datastart, dataend` and so on.

It has several methods for matrix manipulation. Bonus you can create then on CUDA cores as well as `cv::cuda::GpuMat`.

Consider I want to create a matrix with 10 rows, 20 columns, type CV_32FC3:

```
int R = 10, C = 20;
Mat m1;
m1.create(R, C, CV_32FC3); //creates empty matrix

Mat m2(cv::Size(R, C), CV_32FC3); // creates a matrix with R rows, C columns
with data type T where R and C are integers,

Mat m3(R, C, CV_32FC3); // same as m2
```

BONUS:

Compile tiny and compact opencv library for just matrix operations. One of the ways is like as mentioned in this article.

OR

compile opencv source code using following cmake command:

```
$ git clone https://github.com/opencv/opencv.git
$ cd opencv
$ git checkout <version you want to checkout>
$ mkdir build
$ cd build
$ cmake -D WITH_CUDA=OFF -D WITH_MATLAB=OFF -D BUILD_ANDROID_EXAMPLES=OFF -D
BUILD_DOCS=OFF -D BUILD_PERF_TESTS=OFF -D BUILD_TESTS=OFF -
DANDROID_STL=c++_shared -DBUILD_SHARED_LIBS=ON -D BUILD_opencv_objdetect=OFF -D
BUILD_opencv_video=OFF -D BUILD_opencv_videoio=OFF -D
BUILD_opencv_features2d=OFF -D BUILD_opencv_flann=OFF -D
BUILD_opencv_highgui=OFF -D BUILD_opencv_ml=OFF -D BUILD_opencv_photo=OFF -D
```

Try this example:

```
#include "opencv2/core.hpp"
#include<iostream>

int main()
{
    std::cout << "OpenCV Version " << CV_VERSION << std::endl;

    int R = 2, C = 4;
    cv::Mat m1;
    m1.create(R, C, CV_32FC1); //creates empty matrix

    std::cout << "My Mat : \n" << m1 << std::endl;
}
```

Compile the code with following command:

```
$ g++ -std=c++11 opencv_mat.cc -o opencv_mat `pkg-config --libs opencv` `pkg-config --cflags opencv`
```

Run the executable:

```
$ ./opencv_mat

OpenCV Version 3.4.2
My Mat :
[0, 0, 0, 0;
 0, 0, 0, 0]
```

Share Follow

edited Jan 12 at 15:00

answered Jun 18 '20 at 17:11

Milind Deore

2

basic entrywise operations and tensor contractions. Development seems to have slowed down quite some time ago, but perhaps that's just because the library does what it does and not many changes need to be made.

Share Follow

answered May 4 '13 at 13:07

Dan Stahlke

1,349 ● 14 ● 20

2

xtensor is good, but I ended up writing a mini-library myself as a toy project with c++20, while trying to keep the interface as simple as possible. Here it is: https://github.com/gbalduzz/NDArray

Example code:

```
using namespace nd;
NDArray<int, 2> m(3, 3); // 3x3 matrix
m = 2; // assign 2 to all
m(-1, all) = 1; // assign 1 to the last row.

auto tile = m(range{1, end}, range{1, end}); // 2x2 tile
std::sort(tile.begin(), tile.end());

std::cout << m; // prints [[2, 2, 2], [2, 1, 1], [1, 2, 2]]
```

~~It does not provide fancy arithmetic operators collapsing multiple operations together, yet, but~~ you can broadcast arbitrary lambdas to a set of tensors with the same shape, or use lazily evaluated arithmetic operators.

Let me know what do you think about the interface and how it compares with the other options, and if this has any hope, what sort of operations you would like to see implemented.

Free license and no dependency!

Giovanni Balduzzi
21 ● 3

1

VIGRA contains a good N-dimensional array implementation:

http://ukoethe.github.io/vigra/doc/vigra/Tutorial.html

I use it extensively, and find it very simple and effective. It's also header only, so very easy to integrate into your development environment. It's the closest thing I've come across to using NumPy in terms of it's API.

The main downside is that it isn't so widely used as the others, so you won't find much help online. That, and it's awkwardly named (try searching for it!)

Share Follow

answered Aug 5 '15 at 14:27
Martin
11 ● 1

1

Use LibTorch (PyTorch frontend for C++) and be happy.

Share Follow

answered Aug 2 '19 at 10:28
Артем Ященко
21 ● 1

0

Eigen is a template library for linear algebra (matrices, vectors…). It is header only and free to use (LGPL).

Share Follow

answered Jun 23 '12 at 12:21
Claudio
1,570 ● 9 ● 17

0

Share Follow

answered Jun 23 '12 at 12:25

Matt Phillips

8,691 ● 8 ● 41 ● 72

-1

While GLM is designed to mesh easily with OpenGL and GLSL, it is a fully functional header only math library for C++ with a very intuitive set of interfaces.

It declares vector & matrix types as well as various operations on them.

Multiplying two matrices is a simple as (M1 * M2). Subtracting two vectors (V1- V2).

Accessing values contained in vectors or matrices is equally simple. After declaring a vec3 vector for example, one can access its first element with vector.x. Check it out.

Share Follow

answered Sep 18 '15 at 19:01

user5344731