**PES University, Bangalore**


# GENERIC PROGRAMMING PROJECT

## Numpy for C++ - NumCpp

**Submitted To:**
Mr. N S Kumar

**Submitted By:**
Punit P Koujalgi
Sec I, 6th Sem
CS Dept, PESU

# CONTENTS



1. Introduction ---------- 1
2. Codes ---------- 2
3. Explanation ---------- 4
4. Test Cases ---------- 5
5. Results ---------- 8
6. Future Work ---------- 9
7. References ---------- 10

# INTRODUCTION 1

---

NumCpp is a C++ library that aims to emulate the Numpy library of Python. This library has functions to create 1-dimensional and 2-dimensional arrays that can be manipulated the way Numpy arrays can be. Generic approach is used to write the code with the help of C++ templates so that these arrays work with any data type(including user defined classes).Slicing functionality has been provided with the help of reference_wrapper object. Also Iterators are written for both types of arrays to easily access the objects.

To Use the NumCpp library :

- Download the NumCpp code from github
- Include “NumCpp.hpp” header file in your code
- Now you have access Array1D and Array2D to use, but it’s important to know that these are template functions
- Some code-use snippets:

```
Array1D<int> array2{1,3,5,7,9,11};

Array1D<float> array1(10);

for (int i{ 0 }; i<10; ++i)

       array1[i] = i+1.5;

Array2D<int> array2={

        {9,8,7},

        {6,5},

        {3,2,1}};

    std::cout<<array2<<"\n";
```

Please read the full report to view many more examples, understand how to use the library in more complex setting and to understand the implementation.

# CODES



---

## Constructors for 1D-Arrays

```
class Array1D
{
    private:
        int m_length{};
        T* m_data{};
    public:
        Array1D()=default;

        Array1D(int length):m_length{length}
        {
            assert(length>=0);
            if(length>0)
                m_data = new T[length]{};
        }
        Array1D(std::initializer_list<T> list):
Array1D(static_cast<int>(list.size()))
        {
            int count{0};
            for(auto element : list)
            {
                m_data[count] = element;
                ++count;
            }
        }
        Array1D(const Array1D<T> &array): Array1D<T>(array.m_length)
        {     for(int i{0};i<m_length;i++)
                m_data[i]=array.m_data[i];
        }
```

---

## Constructors for 2D-Arrays

```
class Array2D
{     private:
        int m_rows{};
        int m_cols{};
        Array1D<T>* m_data{};
      public:
        Array2D()=default;
        Array2D(int rows,int cols): m_rows(rows),m_cols(cols)
        {   assert(rows>=0 && cols>=0);
            m_data=new Array1D<T>[rows]{};
            for(int i=0;i<rows;i++)
                m_data[i]=Array1D<T>(cols);
        }
        Array2D(std::initializer_list<std::initializer_list<T>> list):
                            m_rows{static_cast<int>(list.size())},m_cols{0}
        {     for(auto &x : list)
                if(static_cast<int>(x.size()) > m_cols)
                    m_cols=static_cast<int>(x.size());
            m_data=new Array1D<T>[m_rows]{};
            auto it=list.begin();
            for(int i=0;i<m_rows;i++)
            {   m_data[i]=Array1D<T>(m_cols);//try resize
                int j=0;
                for(auto &x : *it)
                {   m_data[i][j]=x;
                    j++;
                }
                ++it;
            }
        }
```

# EXPLANATION



---

The Array1D class uses a pointer to dynamically allocated memory to store the objects. It keeps track of length by member m_length. Now Objects can be initialized with just length in which case m_data points to default initialized array. Objects can also be initialized with more intuitive initializer lists. Almost every operator is overloaded to provide functionality that is provided by Numy, for example the arithmetic operators, logical operators etc. Also I/O operators are overloaded to print the array in a pretty way.

The Array2D class also uses a pointer of the type Array1D so that it can point to an array of Array1D( that act as columns). It also has m_rows and m_cols as members to track the shape. Objects can be initialized with number of rows and columns in which case objects are default initialized. However they can also be initialized with an initializer list of initializer lists. Again many operators like arithmetic and logical operators are overloaded along with output operator.

In both the classes, Deep copying has been given importance otherwise if data members of two arrays point to the same dynamically allocated array and life of one ends,it's memory is deallocated and the data member of another one becomes dangling. Also destructors have been written with utmost care to avoid memory leakage.

Slices have been implemented with the concept of class specialization. Both Array1D and Array2D classes have been specialized with reference_wrapper as type. So all we have to do is create an array with the intended type wrapped inside reference_wrapper and assign it any slice of the main array. Also type converter for reference classes are written to convert an array of reference_wrapper to an array of just that type. This copy by value so any changes to this new array doesn't affect the original array.

# TEST CASES



---

```cpp
#include<iostream>
#include "../header/NumCpp.hpp"

int main()
{
    //test 1
    Array1D<int> array1{2,4,6,8};
    Array1D<int> array2{1,3,5,7,9,11};
    Array1D<int> temp;

    std::cout<<"Before Swapping :\n";
    std:: cout << "array1 is: ";;
    for (int i{ 0 }; i<array1.getLength(); ++i)
        std::cout << array1[i] << ' ';
    std::cout << '\n';

    std:: cout << "array2 is: ";;
    for (int i{ 0 }; i<array2.getLength(); ++i)
        std::cout << array2[i] << ' ';
    std::cout << '\n';

    std::cout<<"\n\nAfter Swapping :\n";
    temp=array1;
    array1=array2;
    array2=temp;

    std:: cout << "array1 is: "<<array1<<"\n";
    std:: cout << "array2 is: "<<array2<<"\n";

    system("pause");
}
```

```
E:\Database\sem_6\GP\project2\test>g++ -std=c++17 main1.cpp -o out

E:\Database\sem_6\GP\project2\test>out.exe
Before Swapping :
array1 is: 2 4 6 8
array2 is: 1 3 5 7 9 11


After Swapping :
array1 is: [ 1 , 3 , 5 , 7 , 9 , 11 ]
array2 is: [ 2 , 4 , 6 , 8 ]
Press any key to continue . . .

E:\Database\sem_6\GP\project2\test>
```

```cpp
    std::cout<<"\n\nDifferent Functions:\n";

    Array1D<int> array3{2,4,6,8};
    array3 = { 1, 3, 5, 7, 9, 11 };
    std::cout<<array3<<"\n";

    // Resize the array to 8 elements
    array3.resize(8);
    std::cout<<array3<<"\n";

    // Insert the number 20 before element with index 5
    array3.insertBefore(20, 5);
    std::cout<<array3<<"\n";

    // Remove the element with index 3
    array3.remove(3);
    std::cout<<array3<<"\n";

    // Add 30 and 40 to the end and beginning
    array3.insertAtEnd(30);
    std::cout<<array3<<"\n";

    array3.insertAtBeginning(40);
    std::cout<<array3<<"\n";

    std::cout<<array3.sum()<<"\n";
    std::cout<<array3.min()<<"\n";
    std::cout<<array3.max()<<"\n";
    array3.sort();
    std::cout<<array3<<"\n";
```

```
E:\Database\sem_6\GP\project2\test>out.exe


Different Functions:
[ 1 , 3 , 5 , 7 , 9 , 11 ]
[ 1 , 3 , 5 , 7 , 9 , 11 , 0 , 0 ]
[ 1 , 3 , 5 , 7 , 9 , 20 , 11 , 0 , 0 ]
[ 1 , 3 , 5 , 9 , 20 , 11 , 0 , 0 ]
[ 1 , 3 , 5 , 9 , 20 , 11 , 0 , 0 , 30 ]
[ 40 , 1 , 3 , 5 , 9 , 20 , 11 , 0 , 0 , 30 ]
119
0
40
[ 0 , 0 , 1 , 3 , 5 , 9 , 11 , 20 , 30 , 40 ]


Press any key to continue . . .
```

---

```cpp
#include<iostream>
#include "../header/NumCpp.hpp"

int main()
{
    std::cout<<"Different ways to iterate:\n";

    Array1D<int> array{2,4,6,8,10,12,14,16};

    for (int i=0; i<array.getLength(); ++i)
        std::cout << array[i] << ' ';
    std::cout << '\n';

    for(Array1D<int>::Iterator it= array.begin(); //can
            it != array.end() ; it++)
    {
        std::cout<<*it<<" ";
    }
    std::cout<<"\n";

    for(auto &x: array)
        std::cout<<x<<" ";
    std::cout<<"\n";

    std::cout<<array<<"\n";

    std::cout<<"\n\n\n";
    system("pause");

}
```

```
E:\Database\sem_6\GP\project2\test>g++ -std=c++17 main3.cpp -o out

E:\Database\sem_6\GP\project2\test>out.exe
Different ways to iterate:
2 4 6 8 10 12 14 16
2 4 6 8 10 12 14 16
2 4 6 8 10 12 14 16
[ 2 , 4 , 6 , 8 , 10 , 12 , 14 , 16 ]


Press any key to continue . . .
```

test > main5.cpp > main()

```cpp
#include<iostream>
#include "../header/NumCpp.hpp"

int main()
{
    //Array slicing
    Array1D<int> arr{0,1,2,3,4,5,6,7,8,9};
    Array1D<std::reference_wrapper<int>> slice1=arr(3,8)
    for(int i=0; i<slice1.getLength(); i++)
        std::cout<<slice1[i]<<" ";
    std::cout<<"\n";

    Array1D<std::reference_wrapper<int>> slice3=arr(0,ar
    std::cout<<slice3<<"\n";

    Array1D<std::reference_wrapper<int>> slice2=arr(3,6)
    std::cout<<slice2<<"\n";
    slice2 = { 0, 0, 0};
    std::cout<<slice2<<"\n";
    std::cout<<arr<<"\n";


    //casting
    Array1D<int> arr1{1,2,3,4,5,6,7,8,9};
    Array1D<int> res1(arr1(0,8,2));
    Array1D<std::reference_wrapper<int>> res2(arr1(1,8,2
    Array1D<int> res3= res1 + res2;
    std::cout<<res1<<"\n";
    std::cout<<res2<<"\n";
    std::cout<<res3<<"\n";

}
```

```
E:\Database\sem_6\GP\project2\test>g++ -std=c++17 main5.cpp -o out

E:\Database\sem_6\GP\project2\test>out.exe
3 4 5 6 7
[ 0 , 2 , 4 , 6 , 8 ]
[ 3 , 4 , 5 ]
[ 0 , 0 , 0 ]
[ 0 , 1 , 2 , 0 , 0 , 0 , 6 , 7 , 8 , 9 ]
[ 1 , 3 , 5 , 7 ]
[ 2 , 4 , 6 , 8 ]
[ 3 , 7 , 11 , 15 ]
[ 2 , 4 , 6 , 8 ]
[ 0 , 4 , 6 , 8 ]
[ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ]


Press any key to continue . . .
```

---

test > main1_2D.cpp > main()

```cpp
#include<iostream>
#include "../header/NumCpp.hpp"

int main()
{
    Array2D<int> array1(3,3);

    for(int i=0;i<3;i++)
        for(int j=0;j<3;j++)
            array1[i][j]=1;

    for(int i=0;i<array1.getRows();i++)
    {
        for(int j=0;j<array1.getCols();j++)
            std::cout<<array1[i][j]<<" ";
        std::cout<<"\n";
    }

    Array2D<int> array2={
        {9,8,7},
        {6,5},
        {3,2,1}
    };
    std::cout<<array2<<"\n";

    Array2D<int> array3;
    array3=array2;
    std::cout<<array3<<"\n";
    array3={{1,2,3},
            {4,5,6},
            {7,8,9}};
    std::cout<<array3<<"\n";
```

```
E:\Database\sem_6\GP\project2\test>g++ -std=c++17 main1_2D.cpp -o out

E:\Database\sem_6\GP\project2\test>out.exe
1 1 1
1 1 1
1 1 1
[[ 9 , 8 , 7 ],
 [ 6 , 5 , 0 ],
 [ 3 , 2 , 1 ]]
[[ 9 , 8 , 7 ],
 [ 6 , 5 , 0 ],
 [ 3 , 2 , 1 ]]
[[ 1 , 2 , 3 ],
 [ 4 , 5 , 6 ],
 [ 7 , 8 , 9 ]]


Press any key to continue . . .
```

```cpp
#include<iostream>
#include "../header/NumCpp.hpp"

int main()
{
    std::cout<<"Different ways to iterate:\n";
    Array2D<int> array1={
        {9,8,7},
        {6,5,4},
        {3,2,1}
    };

    for(int i=0;i<array1.getRows();i++)
    {
        for(int j=0;j<array1.getCols();j++)
            std::cout<<array1[i][j]<<" ";
        std::cout<<"\n";
    }

    for(Array2D<int>::Iterator it= array1.begin();
            it != array1.end() ; ++it)
    {
        std::cout<<*it<<" ";
    }
    std::cout<<"\n";

    for(auto &x: array1)
        std::cout<<x<<" ";
    std::cout<<"\n";

    std::cout<<array1<<"\n";
```

```
E:\Database\sem_6\GP\project2\test>g++ -std=c++17 main3_2D.cpp -o out

E:\Database\sem_6\GP\project2\test>out.exe
Different ways to iterate:
9 8 7
6 5 4
3 2 1
9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
[[ 9 , 8 , 7 ],
 [ 6 , 5 , 4 ],
 [ 3 , 2 , 1 ]]


Press any key to continue . . .
```

---

```cpp
int main()
{
    //Array slicing
    Array1D<int> arr{0,1,2,3,4,5,6,7,8,9};
    Array1D<std::reference_wrapper<int>> slice1=arr(3,8);
    for(int i=0; i<slice1.getLength(); i++)
        std::cout<<slice1[i]<<" ";
    std::cout<<"\n";

    Array1D<std::reference_wrapper<int>> slice3=arr(0,arr.getLength(),2);
    std::cout<<slice3<<"\n";

    Array1D<std::reference_wrapper<int>> slice2=arr(3,6);
    std::cout<<slice2<<"\n";
    slice2 = { 0, 0, 0};
    std::cout<<slice2<<"\n";
    std::cout<<arr<<"\n";


    //casting
    Array1D<int> arr1{1,2,3,4,5,6,7,8,9};
    Array1D<int> res1(arr1(0,8,2));
    Array1D<std::reference_wrapper<int>> res2(arr1(1,8,2));
    Array1D<int> res3= res1 + res2;
    std::cout<<res1<<"\n";
    std::cout<<res2<<"\n";
    std::cout<<res3<<"\n";

    Array1D<int> res4(static_cast<Array1D<int>>(res2)); //Doesn't affect o
    std::cout<<res4<<"\n";
    res4[0]=0;
    std::cout<<res4<<"\n";
```

```
E:\Database\sem_6\GP\project2\test>out.exe
[[ 1 , 1 , 1 , 1 ],
 [ 2 , 2 , 2 , 2 ],
 [ 3 , 3 , 3 , 3 ],
 [ 4 , 4 , 4 , 4 ]]
[[ 1 , 1 ],
 [ 2 , 2 ]]
[[ 2 , 2 ],
 [ 3 , 3 ]]
[[ 1 , 1 , 1 , 1 ],
 [ 2 , 0 , 0 , 2 ],
 [ 3 , 0 , 0 , 3 ],
 [ 4 , 4 , 4 , 4 ]]
[[ 2 , 2 , 2 , 2 ],
 [ 4 , 4 , 4 , 4 ]]
[[ 1 , 1 ],
 [ 2 , 2 ],
 [ 3 , 3 ],
 [ 4 , 4 ]]
[[ 1 , 0 , 1 , 0 ],
 [ 2 , 0 , 2 , 0 ],
 [ 3 , 0 , 3 , 0 ],
 [ 4 , 0 , 4 , 0 ]]
[[ 1 , 1 , 1 , 1 ],
 [ 2 , 2 , 2 , 2 ],
 [ 3 , 3 , 3 , 3 ],
 [ 4 , 4 , 4 , 4 ]]
[[ 3 , 3 ],
 [ 5 , 5 ]]


Press any key to continue . . .
```

# RESULTS

All the above test cases were run on a PC with Windows Operating system, 4GB of RAM and in the VS Code environment. The programs were compiled with gcc with C++17 standards. Some of the functionality may break if old compilers are used, so it’s recommended to use C++17 or higher to optimal use.

As we can see from the test cases above, the library provides a clean and intuitive interface to work with arrays. The code runs very fast, that's the magic of C++ and there is no overhead of interpreting line by line as in Python. Almost all everyday use of numpy arrays can be emulated with the help of NumCpp with having to learn a new programming language and with added speed advantage.

# FUTURE WORK



---

This library represents the bare minimum for numpy functionality and therefore has to be greatly improved to compete with numpy. With Numpy we can create N-dimensional arrays, whereas NumCpp is limited to two dimensions. For scientific use, functionality like probability distributions has to be added. Numpy also includes tons of other functionality along the likes of Calculus, fancy indexing and so on. Also a Pandas prototype can be built on top of NumCpp providing minimum functionality.

# REFERENCES

---


1. **https://www.learncpp.com/**

   **This website has been a great resource to learn complex topics in C++**
2. **https://github.com/jakevdp/PythonDataScienceHandbook/blob/8a34a4f653bdbdc01415a94dc20d4e9b97438965/notebooks/02.00-Introduction-to-NumPy.ipynb**

   **Repository that explains concepts in Numpy with the help of examples.**
3. **https://www.youtube.com/playlist?list=PLIrATfBNZ98dudnM48yfGUldqGD0S4FFb**

   **This explains the use of iterators and how we can implement iterators for our own Data structures.**