Numerical Methods

Math 3338 – Spring 2022

Worksheet 3

numpy

1 Reading

CP	2.2.4, 2.2.5, 2.3, 2.5
NMEP	1.2, 1.3, 1.4

Table 1: Sections Covered

2 What is it and Why?

Numpy is a collection of methods for using arrays. An array is like a list except that all the entries of the array must have the same type. This may seem like a limitation, but it turns out to be incredible important. This limitation allows to *vectorize* computations which is an extremely fast operation (you'll see how fast in the homework).

Numpy is used widely in industry for numerical computation. It's arguably the most important Python library. Here is a good quickstart, that's more indepth than this worksheet `https://numpy.org/devdocs/user/quickstart.html`.

3 The Basics

Load numpy using the following,

```
import numpy as np
```

We can define arrays fairly simply,

```
a = np.array([1,2,3,4,5])
b = np.array([10.0,9,8,7,6])
```

We didn't specify a type in these, so Python just kind figures them out. We can, of course, check.

```
a.dtype
b.dtype
```

Notice that a has type "int32" and b has type "float64", 10.0 is a float so it captured that information. You can specify an explicit type `np.array(list, dtype = type)`, so we could have specified 'a' as 'dtype = np.float64' if we wanted it to be floats instead of 'int32'. This usually isn't a huge issue, but it's good to know.

What can you do with arrays? Point-wise arithmetic of course. The two columns here are equivalent (except the right allows for some extra functionality, like ignoring 0 for division)

`a+b`	`np.add(a,b)`
`a-b`	`np.subtract(a,b)`
`a*b`	`np.multiply(a,b)`
`a**b`	`np.power(a,b)`
`a/b`	`np.divide(a,b)`

There are many rules for when this works, the arrays need the same shape[1], the types need to "match"

[1] Well... not really. We might discuss broadcasting

(numbers to numbers). In general, these aren't huge concerns.

You can also have multidimensional arrays (like matrices),

```
A = np.array([[1,2,3],[4,5,6]])
B = np.array([[-1,-2,-3],[-4,-5,-6]])
```

These are 2×3 matrices, which you can tell from `A.shape`. Just remember, $A * B$ is point-wise multiplication NOT MATRIX MULTIPLICATION. You won't be the first nor last to get confused by this.

Here is a long list of functions to create template arrays. Try to figure out what they do.

- `np.zeros(shape)`
- `np.ones(shape)`
- `np.arange(first,last,step,type)`
- `np.linspace(first,last,number)` – We will use this often
- `np.random.rand(shape)`

4 Slicing

You can access elements of an array as you would expect, `a[0]` is 1 (the first element of `a`). You can also slice, the first 3 elements of `a` are `a[:3]`. You can get the first row of a matrix with the following `A[0,:]`, this takes the zeroth row and all the columns.

You can also apply boolean masks to arrays. Try the following.

```
mask = np.array([True,True,False,False,True])
a[mask]
```

You should get the array `[1,2,5]`, or the values where the mask was True. Why is this useful? Let's say I wanted all the values in `a` less than 4.

```
a[a<4]
```

The `a<4` creates a mask and then we apply it. This will turn out to be incredibly useful and you'll probably use it all the time.

5 Vectorization

This is what you've all been waiting for, you just may not have realized it. I need two variables, one normal list and one array.

```
L = [1,2,3,4,5]
a = np.array(L)
```

Find the square root of each element of each.

```
sqrtL = [math.sqrt(elm) for elm in L]
sqrta = np.sqrt(a)
```

For the list, I used a comprehension (the correct way to do this) and for the array I called a numpy function, sqrt. Both of these will have the same result, but the numpy version will be an incredible amount faster[2]. That's because numpy is able to vectorize it's operations.

Vectorization requires knowing some information about the elements of the array. Because arrays require all elements to be the same type, we know the size of each element[3] which allows numpy to quickly iterate over the array. Lists, on the other hand, can have various types and may not be sequential in memory.

[2] At least for large lists.

[3] Size like how much memory it takes to store

So for each element you need to find it in memory, ask how large it is, take the squareroot, ask where the next element is and repeat.

A good analogy is a deck of cards vs a box of paper slips. It's way easier to flip through a deck of cards than it is to pull random pieces of paper from a box.

You should always, when possible, write operations that are or can be vectorized. We'll slowly build when you can and can't do this. As a good rule of thumb always use the numpy version of math operations. Like, `np.sqrt`, `np.sin`, `np.exp`, etc. Also avoid loops at all costs.

6 Arrays

The best explanation of Numpy arrays I've read is this chapter of this book. This chapter is quite long, but it's a fascinating read. It details exactly what an array is, why they are faster, how Python treats variables and lists. You should really consider reading this chapter.

In particular, This section discusses slicing and other array manipulation techniques. In particular you should understand how to build an array, and how array slicing works.

Numerical Methods

Math 3338 – Spring 2022

Homework 3 (Due: Tuesday, January 25)

For this homework set you'll be submitting a PDF which you'll create with LaTeX. If you don't know how to use LaTeX, well you're about to learn real fast. Organize the tables so it all looks good.

Problem 1 (1 pt) The code `np.random.rand(shape)` returns an array of shape with random values between 0 and 1. Describe how to take this output and modify it so that the values are between a and b for numbers a and b. Your answer should have math in it, which LaTeX is great at.

Problem 2 (1 pt) Let's analyze the speed of sorting a list vs an array. Create a list containing 10,000,000 random integers. Copy this to an array. Time how long it takes to sort the array and how long it takes to sort the list. Repeat this 10 times.

```
np.sort(a)
L.sort()
```

Create a table showing your results (you should probably utilize the previous problem). Discuss the difference in timing.

Repeat everything, except with random numbers from 0 to 1. What do you notice is different?

Discuss your findings using words and sentences. Don't write a novel, be concise.

Problem 3 (1 pt) Let's analyze the speed of applying a function to a list vs an array. Create a list containing 10,000,000 random integers. Copy this to an array. Time how long it takes to apply the function the array and how long it takes to apply the function the list. Repeat this 10 times.

For the function, let's use $f(x) = \sin(x)$. You'll need to use a comprehension on the list and the numpy function on the array. You can use np.sin in the comprehension.

Create a table showing your results (you should probably utilize the previous problem). Discuss the difference in timing.

Repeat everything, except with random numbers from 0 to 1. What do you notice is different?

Discuss your findings.