06_Array_Operations_with_NumPy

January 9, 2016

 Thanks: Gilbert Forsyth for help writing the notebooks. NSF for support via CAREER award 1149784. [@LorenaABarba](https://twitter.com/LorenaABarba)

1 12 steps to Navier-Stokes

This lesson complements the first interactive module of the online CFD Python class, by Prof. Lorena A. Barba, called **12 Steps to Navier-Stokes.** It was written with BU graduate student Gilbert Forsyth.

1.1 Array Operations with NumPy

For more computationally intensive programs, the use of built-in Numpy functions can provide an increase in execution speed many-times over. As a simple example, consider the following equation:

$$u_i^{n+1} = u_i^n - u_{i-1}^n$$

Now, given a vector $u^n = [0, 1, 2, 3, 4, 5]$ we can calculate the values of u^{n+1} by iterating over the values of u^n with a for loop.

```
In [1]: import numpy as np
```

```
In [2]: u = np.array((0, 1, 2, 3, 4, 5))

        for i in range(1,len(u)):
            print  u[i]-u[i-1]
```

```
1
1
1
1
1
```

This is the expected result and the execution time was nearly instantaneous. If we perform the same operation as an array operation, then rather than calculate $u_i^n - u_{i-1}^n$ 5 separate times, we can slice the u array and calculate each operation with one command:

```
In [3]: u[1:]-u[0:-1]
```

```
Out[3]: array([1, 1, 1, 1, 1])
```

What this command says is subtract the 0th, 1st, 2nd, 3rd, 4th and 5th elements of u from the 1st, 2nd, 3rd, 4th, 5th and 6th elements of u.

1.1.1 Speed Increases

For a 6 element array, the benefits of array operations are pretty slim. There will be no appreciable difference in execution time because there are so few operations taking place. But if we revisit 2D linear convection, we can see some substantial speed increases.

In [4]:
```
nx = 81
ny = 81
nt = 100
c = 1
dx = 2.0/(nx-1)
dy = 2.0/(ny-1)
sigma = .2
dt = sigma*dx

x = np.linspace(0,2,nx)
y = np.linspace(0,2,ny)

u = np.ones((ny,nx)) ##create a 1xn vector of 1's
un = np.ones((ny,nx)) ##

###Assign initial conditions

u[.5/dy:1/dy+1,.5/dx:1/dx+1]=2
```

With our initial conditions all set up, let's first try running our original nested loop code, making use of the iPython "magic" function **%%timeit**, which will help us evaluate the performance of our code.

Note: The **%%timeit** magic function will run the code several times and then give an average execution time as a result. If you have any figures being plotted within a cell where you run **%%timeit**, it will plot those figures repeatedly which can be a bit messy.

In [5]:
```
%%timeit
u = np.ones((ny,nx))
u[.5/dy:1/dy+1,.5/dx:1/dx+1]=2

for n in range(nt+1): ##loop across number of time steps
    un = u.copy()
    row, col = u.shape
    for j in range(1, row):
        for i in range(1, col):
            u[j,i] = un[j, i] - (c*dt/dx*(un[j,i] - un[j,i-1]))-(c*dt/dy*(un[j,i]-un[j-1,i]))
            u[0,:] = 1
            u[-1,:] = 1
            u[:,0] = 1
            u[:,-1] = 1
```

```
1 loops, best of 3: 3.49 s per loop
```

With the "raw" Python code above, the best execution time achieved was 3.49 seconds. Keep in mind that with these three nested loops, that the statements inside the **j** loop are being evaluated more than 650,000 times. Let's compare that with the performance of the same code implemented with array operations:

In [6]:
```
%%timeit
u = np.ones((ny,nx))
u[.5/dy:1/dy+1,.5/dx:1/dx+1]=2
```

```
        for n in range(nt+1): ##loop across number of time steps
            un = u.copy()
            u[1:,1:]=un[1:,1:]-(c*dt/dx*(un[1:,1:]-un[1:, 0:-1]))-(c*dt/dy*(un[1:,1:]-un[0:-1,1:]))
            u[0,:] = 1
            u[-1,:] = 1
            u[:,0] = 1
            u[:,-1] = 1
```

```
100 loops, best of 3: 7.92 ms per loop
```

As you can see, the speed increase is substantial. The same calculation goes from 3.49 seconds to 7.92 milliseconds. 3 seconds isn't a huge amount of time to wait, but these speed gains will increase exponentially with the size and complexity of the problem being evaluated.

In [7]:
```
from IPython.core.display import HTML
def css_styling():
    styles = open("../styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[7]:
```
<IPython.core.display.HTML at 0x7f3d753eb590>
```