Assignment_4_Notebook

October 5, 2017

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import math
        import time
        import cmath as cmath
        from scipy import fftpack
```

We will be testing both the discrete fourier and fast fourier transform and investigating how the behavior of both scale with the size of vectors we are fourier transforming. Below will be the function defintions which implement our algorithms.

```
In [2]: def discrete(y):
            N = len(y)
            c = np.zeros(N, complex)
            for k in range (0, N):
                for n in range(0, N):
                    c[k] += y[n]*cmath.exp(-2*(math.pi)*1j*k*n/N)
            return c

        def fast(y):
            N = len(y)
            c_even = np.zeros(int(N/2), complex)
            c_odd = np.zeros(int(N/2), complex)
            for k in range(0,int(N/2)):
                for n in range(0,int(N/2)):
                    c_even[k] = y[2*n]*cmath.exp(-2*(math.pi)*1j*k*n/(N/2))
                    c_odd[k] = y[2*n + 1]*cmath.exp(-2*(math.pi)*1j*k*n/(N/2))

            np.insert(c_odd, np.arange(len(c_even)), c_even)
```

We'll define an array of vector sizes called, lengths, which will be the sizes of the vectors we will fourier transform. Similarly we will define an array of times which will be filled with the time it took each method to fourier transform a vector of a given size. The sizes of our vectors will be

$$2^m$$

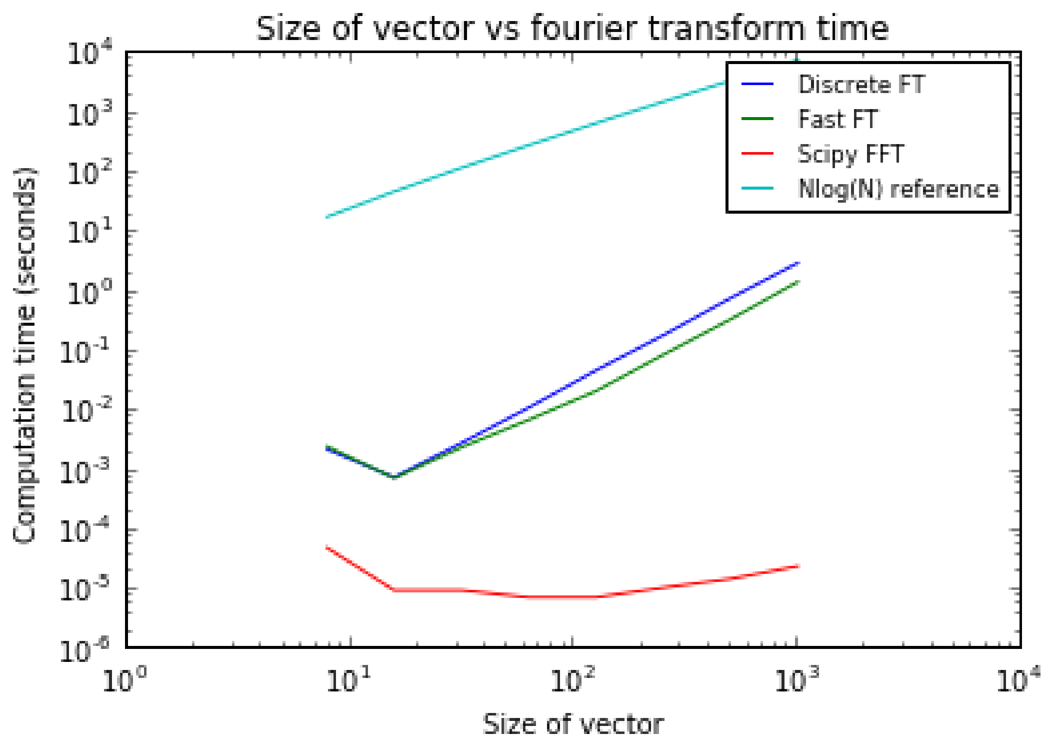
for integer m.

```
In [3]: lengths = np.zeros(8, int)

        for i in range(0,len(lengths)):
            lengths[i] = 2**(i+3)
        times_discrete = np.zeros(len(lengths))
        times_fast = np.zeros(len(lengths))
        times_fastest = np.zeros(len(lengths))

        def fourier(method):
            for j in range(len(lengths)):
                y=np.random.random(lengths[j])

                start_time = time.time()
                if method == 'discrete':
                    c = discrete(y)
                    end_time = time.time()
                    times_discrete[j] = end_time - start_time
                if method == 'fast':
                    c = fast(y)
                    end_time = time.time()
                    times_fast[j] = end_time - start_time
                if method == 'fastest':
                    c = fftpack.fft(y)
                    end_time = time.time()
                    times_fastest[j] = end_time - start_time
```

We then define a method, fourier( ), which takes our algorithm. It initializes a vector of size lengths[j] with random entries and performs the fourier transform. It records the time it takes to do the fourier transform of the given size lengths[j]

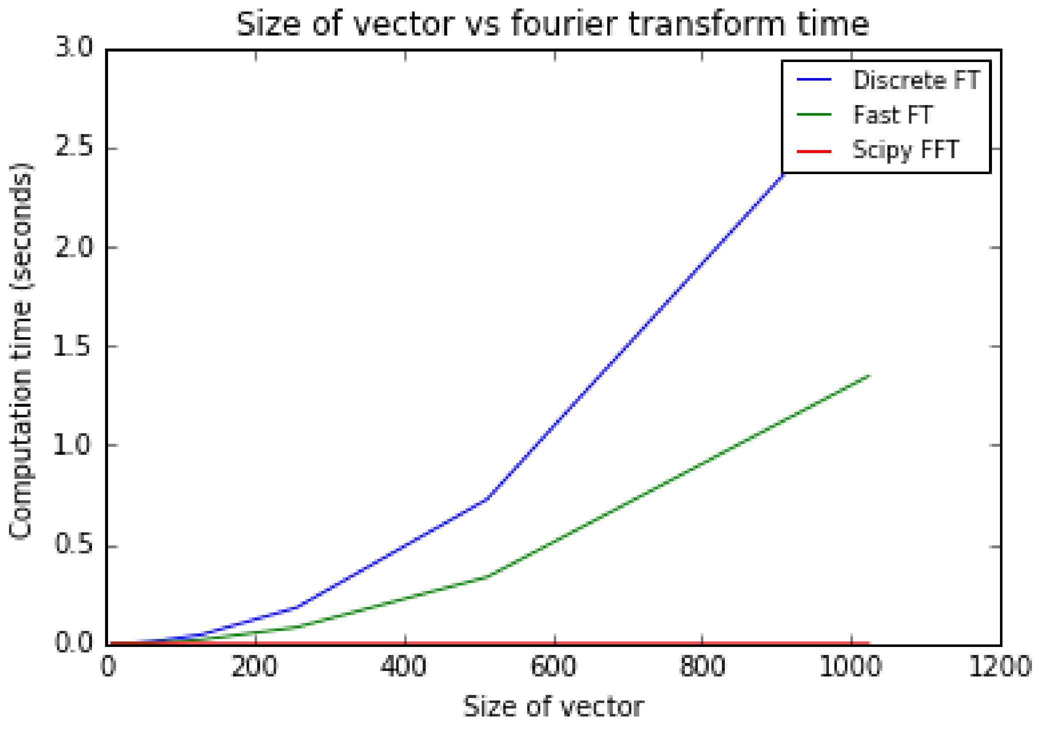
```
In [4]: fourier('discrete')
        fourier('fast')
        fourier('fastest')

        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)

        test = np.zeros(len(lengths))
        for i in range(0, len(lengths)):
                test[i] = lengths[i]*np.log(lengths[i])

        ax.plot(lengths, times_discrete, label='Discrete FT')
        ax.plot(lengths, times_fast, label='Fast FT')
        ax.plot(lengths, times_fastest, label='Scipy FFT')
        ax.plot(lengths, test, label='Nlog(N) reference')

        legend = ax.legend(loc='upper right', fontsize='small')
```

```
ax.set_yscale('log')
ax.set_xscale('log')
ax.set_title('Size of vector vs fourier transform time')
ax.set_xlabel('Size of vector')
ax.set_ylabel('Computation time (seconds)')

plt.show()
```



We plot the three methods in a linear, linear plot make more apparent the difference between the discrete fourier and fast fourier transform

```
In [5]: fourier('discrete')
        fourier('fast')
        fourier('fastest')

        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)

        test = np.zeros(len(lengths))
        for i in range(0, len(lengths)):
            test[i] = lengths[i]*np.log(lengths[i])

        ax.plot(lengths, times_discrete, label='Discrete FT')
        ax.plot(lengths, times_fast, label='Fast FT')
        ax.plot(lengths, times_fastest, label='Scipy FFT')

        legend = ax.legend(loc='upper right', fontsize='small')

        ax.set_title('Size of vector vs fourier transform time')
        ax.set_xlabel('Size of vector')
        ax.set_ylabel('Computation time (seconds)')

        plt.show()
```



As a final test we see what is the largest size vector, of the form

$$2^m$$

, that each method can fourier transform in around a second.

```
In [9]: #Discrete fourier transform
        m = 9
        N = 2**m

        test=np.random.random(N)
        start_time = time.time()
        discrete(test)
        end_time = time.time()

        print('Method: Discrete , Vector size: ', N, ', time: ', end_time - start_time)
```

```
Method: Discrete , Vector size:  512 , time:  0.7232100963592529
```

```
In [10]: #Fast fourier transform
         m = 10
         N = 2**m

         test=np.random.random(N)
         start_time = time.time()
         fast(test)
         end_time = time.time()

         print('Method: Fast , Vector size: ', N, ', time: ', end_time - start_time)
```

```
Method: Fast , Vector size:  1024 , time:  1.3843870162963867
```

```
In [11]: #Scipy fourier transform
         m = 24
         N = 2**m

         test=np.random.random(N)
         start_time = time.time()
         fftpack.fft(test)
         end_time = time.time()

         print('Method: Scipy , Vector size: ', N, ', time: ', end_time - start_time)
```

```
Method: Scipy , Vector size:  16777216 , time:  0.7411789894104004
```

It's easy to see how remarkable the scipy fast fourier transform is. It can transform a vector with almost 17 million elements in one second!

```
In [ ]:
```