MCS 507 Project One : **the Fast Fourier Transform**

The Fourier transform takes a signal from the time into the frequency domain. One application of the Fourier transform is that we can recover the amplitudes and frequencies of a sampled signal.

We will use the package `numpy.fft`. The underlying code for these functions is an f2c-translated and modified version of the FFTPACK routines. FFTPACK [1] is a package of Fortran subprograms for the fast Fourier transform of periodic and other symmetric sequences. It includes complex, real, sine, cosine, and quarter-wave transforms.

For example, consider the signal $2 \cdot \cos(4 \cdot 2\pi t) + 5 \cdot \sin(10 \cdot 2\pi t)$ composed of a cosine with amplitude 2, frequency 4, and a sine with amplitude 5 and frequency 10. Using `rfft` of the package `numpy.fft`, the script below computes the discrete Fourier transform on the real array of samples via the efficient Fast Fourier Transform algorithm. We recover the amplitudes and corresponding frequencies of the components of our signal. With `matplotlib` we plot the spectrum.

```
import numpy as np
f = lambda t: 2*np.cos(4*2*np.pi*t) + 5*np.sin(10*2*np.pi*t)
n = 256
r = np.arange(0.0,1.0,1.0/n)
s = f(r)
F = np.fft.rfft(s)
m = n/2
p = lambda z: (abs(np.real(z))/m,abs(np.imag(z))/m)
t = p(F)
tol = 1.0e-8
for i in range(0,len(t[0])):
   if(t[0][i] > tol):
      print str(t[0][i]) + '*cos(' + str(i) + '*2*pi*t)'
   if(t[1][i] > tol):
      print str(t[1][i]) + '*sin(' + str(i) + '*2*pi*t)'
import matplotlib.pyplot as plt
plt.plot(abs(F)/m)
plt.show()
```

The script prints

```
2.0*cos(4*2*pi*t)
5.0*sin(10*2*pi*t)
```

and a window pops up showing the spectrum.

Assignment One: measuring the CPU time

The cost of the Fast Fourier Transform algorithm on a data set of dimension n is proportional to $n \log_2(n)$. The purpose of this assignment is to experimentally determine whether the running time of the implementation of the FFT algorithm in the package `numpy.fft` is indeed $O(n \log_2(n))$.

To measure the elapsed CPU time in Python programs, we can use the `time` module as follows:

```
import time
start_time = time.clock()
# insert code to time here
stop_time = time.clock()
cpu_time = stop_time - start_time
print 'cpu time :', cpu_time, 'seconds'
```

Write a Python script to run the FFT algorithm on random data of increasing size n (doubling the value for n each time), taking into account the size of the random access memory on your computer. Eventually you may have to execute the same call multiple times in a loop to get times that are sufficiently large to notice. Report the observed execution times in a table. Do you see the $n \log_2(n)$ formula in the observed execution times?

Assignment Two: denoising of signals

One application of the FFT is to remove low amplitude noise from signals. In this assignment, you will write a Python script to simulate the denoising with the FFT. The steps in the script are as follows:

1. take samples of an exact signal;
2. to the sampled signal, add small numbers;
3. apply the Fourier transform and remove those components that have low amplitudes;
4. apply the inverse Fourier transform after removing low amplitude components;
5. compare the result with the original exact signal.

You can try your script on any random data. For a more realistic experiment, you can use the `sound` module of `scitools` and compare the original, the noisy, and the reconstructed signal by listening.

Assignment Three: fast convolution

The Fourier transform turns a convolution into a componentwise product. Interpreting the arrays as the coefficient vectors of two polynomials (the ith entry is the coefficient of x^i), the convolution of the coefficient vectors gives the coefficient vector of the product of the two polynomials. Using the fast FFT, the $O(n^2)$ operation becomes $O(n \log_2(n))$. The script below illustrates the convolution of two arrays padded with enough zeroes:

```
import numpy as np
from numpy.fft import rfft, irfft
a = np.array([1,2,3,4,0,0,0,0])
b = np.array([3,4,9,8,0,0,0,0])
print np.convolve(a,b)
A = rfft(a); B = rfft(b)
C = A*B
print irfft(C)
```

what is printed is

```
[ 3 10 26 50 59 60 32  0  0  0  0  0  0  0  0]
[  3.  10.  26.  50.  59.  60.  32.   0.]
```

Write a python script to demonstrate the benefit of performing a convolution using the FFT, using timings on sufficiently large arrays. Note that `signal` package of `scipy` contains the function `fftconvolve`.

The deadline is Monday 24 September at 10AM

On the deadline, bring to class the answers to the assignments on paper. In addition, also send me by email the scripts you wrote. You may work individually or in pairs on this project.

If you have questions or difficulties with the assignments, feel free to come to my office for help.

References

[1] P.N. Swarztrauber. Vectorizing the FFTs. In *Parallel Computations*, edited by G. Rodrigue, pages 51–83, Academic Press, 1982.