Assignment 6

Corrections: [none yet]

1. This is an exercise in Python coding with modules and in how numerical codes are validated. Write a Python module `DirectDFT.py` with function `DFT(f)` that takes a one dimensional real `numpy` array, `f` and returns a one dimensional complex numpy array of the same length that is the DFT of f. It should compute the discrete Fourier transform (DFT) sums directly

$$\widehat{f}_j = \frac{1}{n} \sum_{k=0}^{n-1} e^{-2\pi i jk/n} f_k \; .$$

Write a test program that imports the `DirectDFT` module and checks that the following things are done correctly. One of the first lines of your test program should be: `import DirectDFT`, or: `import DirectDFT as SFTW` (slowest Fourier transform in the west) or with some other name.

(a) The correct $\widehat{f}_k$ if $f_j = \delta_{km}$. This is the array with $f_m = 1$ and $f_j = 0$ if $j \neq m$. You don't have to do this for every m, but you should do more than one. Don't print the sequence $\widehat{f}_k$, just print a number saying how accurate it was.

(b) The Parseval relation (you need to find C for this convention of DFT)

$$\sum_j f_j^2 = C \sum_k \left| \widehat{f}_k \right|^2 \; .$$

(c) The left circular shift $g = Lf$ is given in components by $g_j = f_{j-1}$, $g_0 = f_{n-1}$. Find the DFT of g in terms of the DFT of f and test that your code does this relation correctly. Choose a non-trivial f and print only a measure of accuracy, not the sequences.

(d) The *inverse DFT* formulas are almost the same as the *direct DFT* ones. If the direct DFT is written abstractly as a linear operator (matrix) $\mathcal{F}$, so $\widehat{f} = \mathcal{F} f$, the inverse $f = \mathcal{F}^{-1} \widehat{f}$ is almost the same as $\mathcal{F}^{-1} = \mathcal{F}^*$. Add an inverse DFT `iDFT` function to your `DirectDFT.py` module and check that your functions do this

$$f \xrightarrow{\texttt{DFT}} \widehat{f} \xrightarrow{\texttt{iDFT}} f \; ,$$

to near machine precision but not exactly.

(e) Check that your `DFT` function computes $\mathcal{F}^4 = CI$ to near machine precision. on the right side I is the identity matrix and C is a constant (a power of n). That is, check that

$$f \xrightarrow{\texttt{DFT}} *** \xrightarrow{\texttt{DFT}} *** \xrightarrow{\texttt{DFT}} *** \xrightarrow{\texttt{DFT}} f \; .$$

2. This is an exercise in looking at the DFT and Fourier series of specific functions. For this and the rest of this assignment, use the `numpy` functions `numpy.fft.fft` and `numpy.fft.ifft`. Let $f(x)$ be a periodic function with period L, so $f(x+L) = f(x)$. Choose a symmetric interval around zero, which is $-\frac{L}{2} \leq x \leq \frac{L}{2}$. Sample f at n uniformly spaced points in this interval $x_k = k\Delta x$, with $\Delta x = L/n$. The samples f_k form a vector in $\mathbb{R}^n$. It is important to include only one of the endpoints, because $f(-L/2) = f(L/2)$. Consider two functions

$$\begin{aligned} g(x) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \\ h(x) &= \frac{1}{2} e^{-|x|} \end{aligned}$$

These functions are not periodic, but if L is large enough they "act like" periodic functions. You could take the periodic version

$$g_{\mathrm{P}}(x) = \sum_{n=-\infty}^{\infty} g(x+n) \; .$$

But if L is large enough ($L = 6$ is probably large enough), then g and g_{P} are indistinguishable for $x \in \left[-\frac{L}{2}, \frac{L}{2}\right]$.

Compute the DFT of g and h, and plot $|\widehat{g}_k|$ and $\left|\widehat{h}_k\right|$ in a centered range (approximately) $-\frac{n}{2} \leq k \leq \frac{n}{2}$. These should be the same for $k = 0$ and both decay as k grows. Comment on which set of DFT coefficients converge to zero faster.

3. A function $p(x)$ is a *trigonometric polynomial* with period L and n terms if

$$p(x) = \sum_{\left|j \leq \frac{n}{2}\right|} a_j e^{2\pi i j x/L} \; .$$

If n is even, then the range of j in the sum must be altered a little to insure that there n terms in the sum. A trigonometric polynomial *interpolates* a function $f(x)$ at points x_k if $f(x_k) = p(x_k)$. Suppose the sample points x_k are evenly spaced as before, find an algorithm and write a program to evaluate the coefficients a_j using a size n DFT. Find a DFT algorithm that evaluates p at $M \ll n$ evenly spaced points with spacing $h = L/M$. Take M to be large (maybe $M = 1000$) but experiment with small n. Plot p and f on the same plot to see the difference. Apply this to the functions g and h above. Show that g is well approximated with a small n but h requires larger n to be represented accurately.

4. The *spectral derivative* of a function is the derivative of the interpolating trigonometric polynomial. Write a DFT based program to evaluate the spectral derivative of f at n evenly spaced interpolation points.

   (a) Plot the error $f'(x_k) - p'(x_k)$ for functions g and h. Comment on the accuracy. Note that the spectral derivative of h is inaccurate even in regions where h itself is smooth. This is called *pollution*: errors created at one point (the singularity of h at $x = 0$) and then spread to larger parts of the computation.

   (b) Compute for a sequence of Δx the accuracy measure

   $$D_k = \max_k |f'(x_k) - p'(x_k)| \ .$$

   This should show that the spectral derivative of g is *exponentially* accurate, decreasing like an exponential involving Δx, while the spectral derivative of h is quite poor.