About Fourier Interpolation and Differentiation

Consider an interval [-A,A] and a 2A-periodic function f to be approximated over the interval by a trigonometric polynomial of degree N

$$p(x) = \sum_{k=-N/2}^{N/2-1} p_k v_k(x), v_k(x) = e^{i\frac{k\pi}{A}x}, k = -N/2, -N/2+1, ..., 0, 1, ..., N/2-1$$

$\{v_k\}$ is an orthogonal, linearly independent set of functions:

$$\frac{1}{2A}\int_{-A}^{A} v_k(x)\bar{v}_m(x)dx = \delta_{km}$$

(note the complex conjugate). They are *also* an orthogonal basis for the gridpoint values

$$f(x_m), m = -N/2, ..., N/2-1, x_m = \frac{m}{N}2A$$

$$\frac{1}{N}\sum_{j=-N/2}^{N/2-1} v_k(x_j)\bar{v}_m(x_j) = \delta_{km}$$

so the gridpoint values of any 2A-periodic function f admit a representation

$$f(x_m) = p_N(x_m) = \sum_{k=-N/2}^{N/2-1} \hat{f}_k e^{i\frac{k\pi}{A}x_m} = \sum_{k=-N/2}^{N/2-1} \hat{f}_k e^{i\frac{km\pi}{N}}, \hat{f}_k = \frac{1}{N}\sum_{m=-N/2}^{N/2-1} f(x_m)e^{-i\frac{km\pi}{N}}$$

p_N can be evaluated for any x (just replace x_m by x) and is the interpolating trigonometric polynomial. The coefficients can be evaluated by the Discrete Fourier Transform.

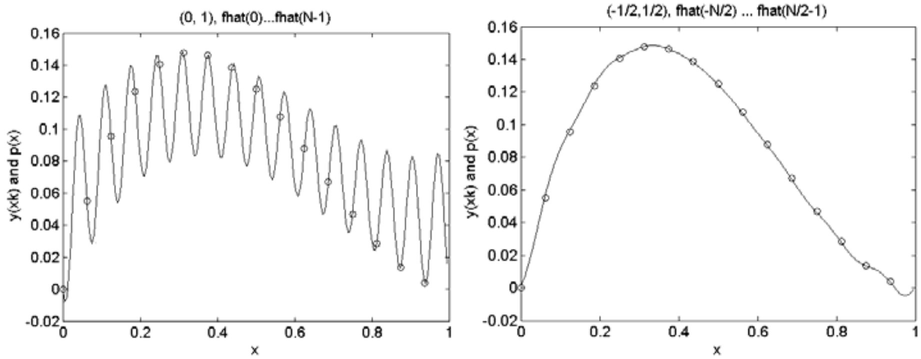
Matlab's FFT , according to the documentation,

- Computes N times the fhat coefficients numbered 0 to N-1 (i.e., 1 to N with Matlab indexing which starts at 1),
- IFFT divides by N
- the x-values are 0,h,2h,…,(N-1)h, h=1/N.

```
For length N input vector x, the DFT is a length N vector X,
with elements
                     N
   X(k) =           sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.
                    n=1
The inverse DFT (computed by IFFT) is given by
                     N
   x(n) = (1/N) sum  X(k)*exp( j*2*pi*(k-1)*(n-1)/N), 1 <= n <= N.
                    k=1
```

Clearly, it is necessary to experiment here and understand how to use Matlabs FFT for differentiation. Choose $f(x) = x(1-x)^2$ so that f(0) = f(1) = 0, but f' and f'' do not match.

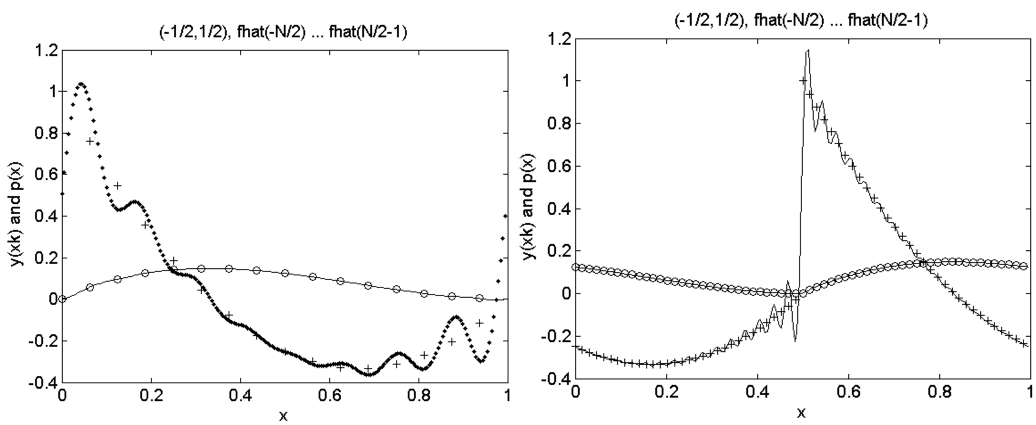
First, use the Matlab coefficients and basis functions $v_k = e^{i2\pi kx}$, k = 0,1,…,N-1,the interval [0,1) and data points $x_m = m/N, m = 0,1,2,..., N-1$. The result is in Fig. 1, left.
To the right is the result with v_k, k = -N/2,…,N/2-1, interval [-1/2,1/2), data points $x_m = (m - N/2)/N, m = 0,1,2,..., N-1$



The two trigonometric polynomials behave VERY differently between grid-points. But in fact, they differ only by a multiple of the lowest degree exp. function which vanishes at *all* gridpoints, $c_N \sin(\frac{x-a}{b-a} N\pi)$

Exercise: Do the math and find c_N !

`Clearly`, the symmetric variant looks better, so we use that for differentiation:



The crosses are the exact values of the derivative. The approximation is good in the central parts of the interval, but poor at the end-points. The reason is that *f* has discontinuous derivatives of order 1 and 2 at $x = 0$ (and 1): The Gibbs phenomenon. To the right is a plot with 64 data points, and the data shifted 1/2 period so the jump is at $x = 0.5$. Of course, the interpolation error is also large at x = 0.5, but the discrepancy is better seen in the derivative which is discontinuous.

Here is an approximation to spectral computation of the *n*:th derivative, using the symmetric variant:

```
function yd = ffd(y,a,b,n)
% Compute yd = n:th derivative of y(x)
% defined by its values y at (a,a+h,...,a+(N-1)h), a+Nh = b
N    = length(y);
yhat = fftshift(fft(y));
d    = 2*pi*1i*(-N/2:N/2-1)/(b-a);
yd   = ifft(fftshift(d.^n.*yhat));
```

`Exercise` Explain the **`d-`**vector used as differentiation operator in wavenumber space.

Note: For the tasks in HW4 it may be more convenient to use the ingredients of **`ffd`** than the whole code itself.