

Math 128a, Section 3 — Solutions to Problem Set 4

The goal of this problem set is to build and analyze a “nonequidistant Fast Fourier Transform” which evaluates a periodic trigonometric interpolant

$$t(x) = \sum_{l=0}^{N-1} c_l e^{ilx}$$

at  $M$  arbitrarily given points  $x_j$  in the interval  $[0, 2\pi]$  at optimal cost  $O((N \log N + M) \log \epsilon)$  where  $\epsilon$  is the desired accuracy.

(1) Show that the error in degree  $2k - 1$  centered equidistant polynomial interpolation of  $t(x)$  from a mesh of points  $x_j = jh$ , where  $h = 2\pi/qN$ , is bounded by

$$|t(x) - p(x)| \leq C \left(\frac{\pi}{q}\right)^{2k}$$

for  $x$  in the center interpolation interval  $kh \leq x \leq (k+1)h$ , where

$$C = \sum_{l=0}^{N-1} |c_l|.$$

(Hint:  $1^2 \cdot 3^2 \cdot 5^2 \cdots (2k-1)^2 < 1 \cdot 2 \cdot 3 \cdot 4 \cdots 2k = (2k)!$ )

**Solution:** We are interpolating the function  $t(x)$  on the center interval of  $2k$  points evenly spaced at distance  $h$  one from another. We have an error bound for such approximations:

$$|t(x) - p(x)| \leq \frac{M_{2k}}{(2k)!} (x - x_1)(x - x_2) \cdots (x - x_{2k}),$$

where  $M_{2k}$  is an upper bound on  $t^{(2k)}(x)$  and  $x_1, \dots, x_{2k}$  are the points at which we are interpolating. The linear factors of  $p(x)$  may be grouped in pairs as follows:

$$\begin{aligned} (x - x_k)(x - x_{k+1}) &\leq \left(\frac{h}{2}\right)^2 \\ (x - x_{k-1})(x - x_{k+2}) &\leq \left(\frac{3h}{2}\right)^2 \\ &\vdots \\ (x - x_1)(x - x_{2k}) &\leq \left(\frac{(2k-1)h}{2}\right)^2 \end{aligned}$$

giving us

$$\begin{aligned} |t(x) - p(x)| &\leq \frac{M_{2k}}{(2k)!} \left(\frac{h}{2}\right)^{2k} (1)^2 (3)^2 \cdots (2k-1)^2 \\ &\leq \frac{M_{2k}}{(2k)!} \left(\frac{h}{2}\right)^{2k} (1)(2)(3)(4) \cdots (2k-1)(2k) \\ &= M_{2k} \left(\frac{h}{2}\right)^{2k}. \end{aligned}$$

Now we give bounds on the derivatives of  $t(x)$ :

$$\frac{d^{2k}}{dx^{2k}} \left( \sum_{l=0}^{N-1} c_l e^{ilx} \right) = \sum_{l=0}^{N-1} c_l (il)^{2k} e^{ilx}$$

$$|t^{(2k)}(x)| \leq \sum_{l=0}^{N-1} |c_l| \cdot l^{2k} \cdot 1 \leq \sum_{l=0}^{N-1} |c_l| \cdot N^{2k} = N^{2k} C.$$

Plugging in  $h = 2\pi/qN$ , we arrive at

$$|t(x) - p(x)| \leq N^{2k} C \left( \frac{2\pi}{2qN} \right)^{2k} = C \left( \frac{\pi}{q} \right)^{2k}.$$

(2) Write a matlab script `t = neqfft( n, c, m, x, k, q )` that (a) uses matlab's built-in FFT to evaluate  $t(x)$  on an equidistant mesh  $x = jh$  with  $0 \leq j \leq qN$ , (b) wraps the resulting values periodically around to cover the range  $-k \leq j \leq qN + k$ , and (c) interpolates (use the Newton form) from the nearest  $2k$  equidistant mesh values to each of  $M$  arbitrarily given points  $x_j$  in the interval  $[0, 2\pi]$ . Compare the speed (in flops or CPU time) and accuracy of your script to direct evaluation of  $t(x)$  at each  $x_j$ , for  $N = 16, 32, 64, 128, 256$  random  $c_k$ 's and  $M = 10, 20, 40, 80, 160$  random points  $x_j$  in  $[0, 2\pi]$ . Experimentally determine  $k$  and  $q$  to approximately minimize flops or CPU time while guaranteeing that errors are bounded by  $10^{-5}C$ .

**Solution:** See the sample scripts and output below. Most of you used subroutines with a lot of overhead and thus found that `neqfft` was actually slower than direct evaluation in the tested range, but you should still have been able to notice a linear trend in execution time as a function of  $M$ , and a nearly linear trend in terms of  $N$ .

<http://www.math.berkeley.edu/~hthall/128A/matlab/neqfft.m>

```
function [ t, s, errbd, err, tcpu, scpu ] = neqfft( n, c, m, x, k, q )
% Evaluate nonequidistant FFT of n coeffs c at m arbitrary points x:
% t(x_j) = sum_{k=1}^n c(k) e^{i(k-1)x_j} for j = 1, ..., m.
% First FFT onto an q-oversampled mesh, then use centered polynomial interpolation
% of degree 2k-1. Error bounded by (pi/q)^{2k}.
% Alternatively could use Aitken or Neville formula if m is small,
% Alternatively could evaluate derivatives t'(x), t''(x), ... and
% use Hermite interpolation to decrease stencil size and thereby slightly decrease error.
%
tic;
errbd = ( pi / q )^(2*k) * sum( abs( c ) );
nq = q * n;
hq = 2*pi/nq;
nk = nq + 2*k+1;
d = zeros( 2*k, nk );
% Take standard FFT (of automatically zero-padded coeffs c) to get nq mesh values d.
% d( j, 1 ) = sum_{k=1}^n c(k) exp( -i 2 pi (j-1) (k-1) / nq ) for j = 1, ..., nq
d( 1, 1:nq ) = fft( c, nq ) ;
```

```

%
% Add 2k+1 periodic values at right end---shift x if near left end.
d( 1, 1:nk ) = [ d( 1, 1:nq ) d( 1, 1:2*k+1 ) ];
% Compute divided differences of d --- this is efficient if m is large.
for j = 1 : 2*k-1
    % Skip dividing by hq as we won't be including it in our Newton formula.
    d( j+1, 1:nk-j ) = ( d( j, 2:nk-j+1 ) - d( j, 1:nk-j ) ) / j;
end
%
% Locate each x_j in mesh (shift at left end) and interpolate by Newton formula.
t = zeros( 1, m );
for j = 1:m
    % Index of nearest grid point x_k = (kc-1)*hq left of x(j)=(kc-1+tc)*hq.
    kc = 1 + floor( x(j) / hq );
    % Index of left endpoint of stencil.
    kl = kc - k + 1;
    % Integer multiple plus fraction of mesh spacing from kl to x(j).
    tl = ( x(j) - ( kl-1 ) * hq ) / hq;
    % Wrap around if kl runs off left end of periodic array.
    if kl < 1
        kl = kl + nq;
    end
    % Horner's rule for Newton interpolation from kl to t.
    t(j) = d( 2*k, kl );
    for s = 2*k-2:-1:0
        t(j) = d( s+1, kl ) + ( tl-s ) * t(j);
    end
end
end
tcpu = toc;
clear d
% Check against direct evaluation
tic;
s = zeros( 1, m );
for j = 1:m
    sum = 0.0;
    for k = 1:n
        sum = sum + c(k) * exp( -i * (k-1) * x(j) );
    end
    s(j) = sum;
end
end
err = norm( s - t );
scpu = toc;

```

<http://www.math.berkeley.edu/~hthall/128A/matlab/testneqfft.m>

```

% Test neqfft.m on random data.
% Determine optimal parameters for given accuracy.

```



```
plot( 20*2.^(0.5*p), tcpu );  
plot( 20*2.^(0.5*p), scpu );  
p  
scpu  
tcpu  
eps  
kmin  
qmin
```

