Numerical Methods I – Problem Set 8

Fall Semester 2005

Due November 14

In *trigonometric interpolation* we use a truncated Fourier series to interpolate a periodic function u on the interval $[0, 2\pi]$. Let N be an even number of equidistant nodes $x_j = jh$, where $h = 2\pi/N$ and $j = 0, \ldots, N-1$.

The trigonometric interpolant of u is the function

$$v(x) = \sum_{k=-N/2}^{N/2-1} c_k \, \mathrm{e}^{\mathrm{i}kx} \, . \tag{1}$$

To determine the coefficients c_k, we impose the interpolation condition $v(x_j) = u(x_j)$ for $j = 0, \ldots, N-1$, i.e.

$$\sum_{k=-N/2}^{N/2-1} c_k \, \mathrm{e}^{\mathrm{i}kx_j} = u(x_j) \, . \tag{2}$$

We multiply this equation by $\mathrm{e}^{-\mathrm{i}mx_j}$ and sum over all j, so that

$$\sum_{j=0}^{N-1} \mathrm{e}^{-\mathrm{i}mx_j} \sum_{k=-N/2}^{N/2-1} c_k \, \mathrm{e}^{\mathrm{i}kx_j} = \sum_{j=0}^{N-1} \mathrm{e}^{-\mathrm{i}mx_j} \, u(x_j) \, . \tag{3}$$

Notice that

$$\begin{aligned}
\sum_{j=0}^{N-1} \mathrm{e}^{-\mathrm{i}mx_j} \sum_{k=-N/2}^{N/2-1} c_k \, \mathrm{e}^{\mathrm{i}kx_j} &= \sum_{k=-N/2}^{N/2-1} c_k \sum_{j=0}^{N-1} \mathrm{e}^{\mathrm{i}(k-m)x_j} \\
&= \sum_{k=-N/2}^{N/2-1} c_k \sum_{j=0}^{N-1} \mathrm{e}^{\mathrm{i}(k-m)hj} \\
&= \sum_{k=-N/2}^{N/2-1} c_k \cdot \begin{cases} N & \text{if } k = m \\ \dfrac{1 - \left(\mathrm{e}^{\mathrm{i}(k-m)2\pi/N}\right)^N}{1 - \mathrm{e}^{\mathrm{i}(k-m)2\pi/N}} = 0 & \text{if } k \neq m \end{cases} \\
&= N \, c_m \, .
\end{aligned} \tag{4}$$

We therefore obtain that

$$c_m = \frac{1}{N} \sum_{j=0}^{N-1} \mathrm{e}^{-\mathrm{i}mx_j} \, u(x_j) \, . \tag{5}$$

Remark: The normalization convention is arbitrary. Most library routines, including those in Octave, have the factor $1/N$ in the inverse transform (1) rather than in the forward transform (5). The convention above has the advantage that the *discrete Fourier transform* (5) can be seen as the Riemann sum approximation of the continuous Fourier transform. If both the forward and the inverse transform get a factor of $1/\sqrt{N}$, the transform is unitary, which has certain theoretical advantages, but is usually avoided in actual code.

1. **Project:** Write an Octave program which computes the coefficients c_m for $m = -\frac{N}{2}, \ldots, \frac{N}{2} - 1$ for a given function u, e.g.

   $$u(x) = x\,(x - 2\,\pi)\,\mathrm{e}^{-x}\,.$$

   This operation can be done very efficiently via the *Fast Fourier Transform*, available in *Octave* as the function `fft`. Compare your result with that produced by `fft` to find out how the coefficients c_m are laid out in Octave's data structure.

2. **Project:** Write an Octave function `tpolyval(c,x)`, which, in analogy with the built-in function `polyval`, evaluates the trigonometric interpolant v via equation (1). Use the coefficient layout as employed by `fft`.

   (a) Plot u and v for the given example.

   (b) What happens when $N = 10$ and you interpolate $u(x) = \sin(6x)$?