5.8 Chebyshev Approximation

The Chebyshev polynomial of degree n is denoted $T_n(x)$, and is given by the explicit formula

$$T_n(x) = \cos(n \text{ arccos } x) \tag{5.8.1}$$

This may look trigonometric at first glance (and there is in fact a close relation between the Chebyshev polynomials and the discrete Fourier transform); however (5.8.1) can be combined with trigonometric identities to yield explicit expressions for $T_n(x)$ (see Figure 5.8.1),

$$\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
&\cdots \\
T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \quad n \geq 1.
\end{aligned} \tag{5.8.2}$$

(There also exist inverse formulas for the powers of x in terms of the T_n's — see equations 5.11.2-5.11.3.)

The Chebyshev polynomials are orthogonal in the interval $[-1, 1]$ over a weight $(1 - x^2)^{-1/2}$. In particular,

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0 & i \neq j \\ \pi/2 & i = j \neq 0 \\ \pi & i = j = 0 \end{cases} \tag{5.8.3}$$

The polynomial $T_n(x)$ has n zeros in the interval $[-1, 1]$, and they are located at the points

$$x = \cos\left(\frac{\pi(k - \frac{1}{2})}{n}\right) \qquad k = 1, 2, \ldots, n \tag{5.8.4}$$

In this same interval there are $n + 1$ extrema (maxima and minima), located at

$$x = \cos\left(\frac{\pi k}{n}\right) \qquad k = 0, 1, \ldots, n \tag{5.8.5}$$

At all of the maxima $T_n(x) = 1$, while at all of the minima $T_n(x) = -1$; it is precisely this property that makes the Chebyshev polynomials so useful in polynomial approximation of functions.

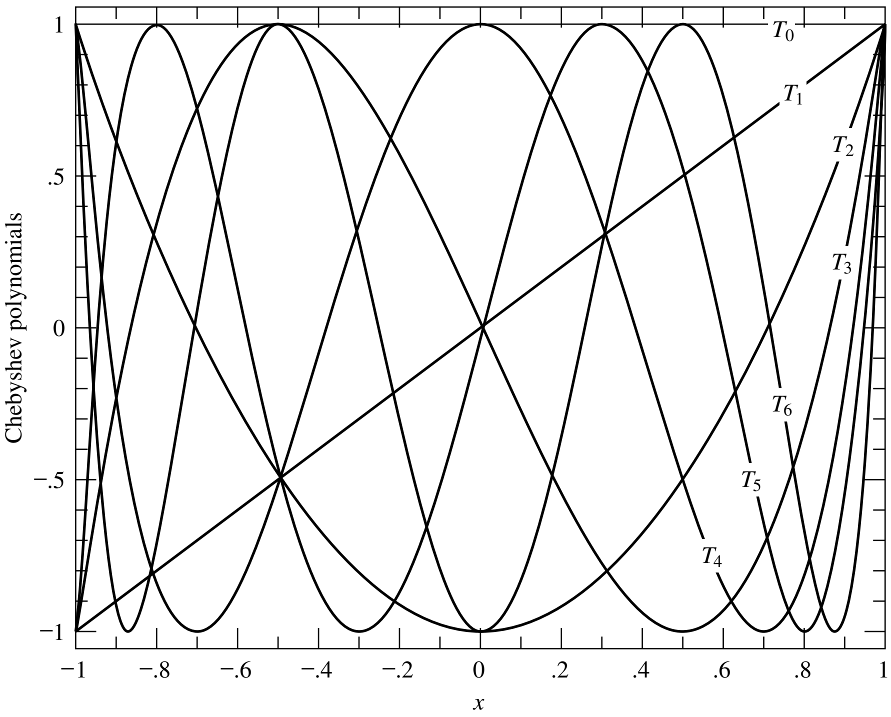

Figure 5.8.1. Chebyshev polynomials $T_0(x)$ through $T_6(x)$. Note that T_j has j roots in the interval $(-1,1)$ and that all the polynomials are bounded between ± 1.

The Chebyshev polynomials satisfy a discrete orthogonality relation as well as the continuous one (5.8.3): If x_k $(k = 1,\dots,m)$ are the m zeros of $T_m(x)$ given by (5.8.4), and if $i,j < m$, then

$$\sum_{k=1}^{m} T_i(x_k)T_j(x_k) = \begin{cases} 0 & i \neq j \\ m/2 & i = j \neq 0 \\ m & i = j = 0 \end{cases} \tag{5.8.6}$$

It is not too difficult to combine equations (5.8.1), (5.8.4), and (5.8.6) to prove the following theorem: If $f(x)$ is an arbitrary function in the interval $[-1,1]$, and if N coefficients $c_j, j = 0,\dots,N-1$, are defined by

$$\begin{aligned} c_j &= \frac{2}{N}\sum_{k=1}^{N} f(x_k)T_j(x_k) \\ &= \frac{2}{N}\sum_{k=1}^{N} f\left[\cos\left(\frac{\pi(k-\frac{1}{2})}{N}\right)\right]\cos\left(\frac{\pi j(k-\frac{1}{2})}{N}\right) \end{aligned} \tag{5.8.7}$$

then the approximation formula

$$f(x) \approx \left[\sum_{k=0}^{N-1} c_k T_k(x)\right] - \frac{1}{2}c_0 \tag{5.8.8}$$

is *exact* for x equal to all of the N zeros of $T_N(x)$.

For a fixed N, equation (5.8.8) is a polynomial in x which approximates the function $f(x)$ in the interval $[-1, 1]$ (where all the zeros of $T_N(x)$ are located). Why is this particular approximating polynomial better than any other one, exact on some other set of N points? The answer is *not* that (5.8.8) is necessarily more accurate than some other approximating polynomial of the same order N (for some specified definition of "accurate"), but rather that (5.8.8) can be truncated to a polynomial of *lower* degree $m \ll N$ in a very graceful way, one that *does* yield the "most accurate" approximation of degree m (in a sense that can be made precise). Suppose N is so large that (5.8.8) is virtually a perfect approximation of $f(x)$. Now consider the truncated approximation

$$f(x) \approx \left[\sum_{k=0}^{m-1} c_k T_k(x)\right] - \frac{1}{2}c_0 \tag{5.8.9}$$

with the same c_j's, computed from (5.8.7). Since the $T_k(x)$'s are all bounded between ± 1, the difference between (5.8.9) and (5.8.8) can be no larger than the sum of the neglected c_k's ($k = m, \ldots, N-1$). In fact, if the c_k's are rapidly decreasing (which is the typical case), then the error is dominated by $c_m T_m(x)$, an oscillatory function with $m+1$ equal extrema distributed smoothly over the interval $[-1, 1]$. This smooth spreading out of the error is a very important property: The Chebyshev approximation (5.8.9) is very nearly the same polynomial as that holy grail of approximating polynomials the *minimax polynomial*, which (among all polynomials of the same degree) has the smallest maximum deviation from the true function $f(x)$. The minimax polynomial is very difficult to find; the Chebyshev approximating polynomial is almost identical and is very easy to compute!

So, given some (perhaps difficult) means of computing the function $f(x)$, we now need algorithms for implementing (5.8.7) and (after inspection of the resulting c_k's and choice of a truncating value m) evaluating (5.8.9). The latter equation then becomes an easy way of computing $f(x)$ for all subsequent time.

The first of these tasks is straightforward. A generalization of equation (5.8.7) that is here implemented is to allow the range of approximation to be between two arbitrary limits a and b, instead of just -1 to 1. This is effected by a change of variable

$$y \equiv \frac{x - \frac{1}{2}(b+a)}{\frac{1}{2}(b-a)} \tag{5.8.10}$$

and by the approximation of $f(x)$ by a Chebyshev polynomial in y.

```
#include <math.h>
#include "nrutil.h"
#define PI 3.141592653589793

void chebft(float a, float b, float c[], int n, float (*func)(float))
```

Chebyshev fit: Given a function `func`, lower and upper limits of the interval [a,b], and a maximum degree `n`, this routine computes the `n` coefficients `c[0..n-1]` such that $\text{func}(x) \approx [\sum_{k=0}^{n-1} c_k T_k(y)] - c_0/2$, where y and x are related by (5.8.10). This routine is to be used with moderately large `n` (e.g., 30 or 50), the array of `c`'s subsequently to be truncated at the smaller value m such that c_m and subsequent elements are negligible.

```
{
    int k,j;
    float fac,bpa,bma,*f;
```

```
    f=vector(0,n-1);
    bma=0.5*(b-a);
    bpa=0.5*(b+a);
    for (k=0;k<n;k++) {                  We evaluate the function at the n points required
        float y=cos(PI*(k+0.5)/n);          by (5.8.7).
        f[k]=(*func)(y*bma+bpa);
    }
    fac=2.0/n;
    for (j=0;j<n;j++) {
        double sum=0.0;                  We will accumulate the sum in double precision,
        for (k=0;k<n;k++)                    a nicety that you can ignore.
            sum += f[k]*cos(PI*j*(k+0.5)/n);
        c[j]=fac*sum;
    }
    free_vector(f,0,n-1);
}
```

(If you find that the execution time of `chebft` is dominated by the calculation of N^2 cosines, rather than by the N evaluations of your function, then you should look ahead to §12.3, especially equation 12.3.22, which shows how fast cosine transform methods can be used to evaluate equation 5.8.7.)

Now that we have the Chebyshev coefficients, how do we evaluate the approximation? One could use the recurrence relation of equation (5.8.2) to generate values for $T_k(x)$ from $T_0 = 1, T_1 = x$, while also accumulating the sum of (5.8.9). It is better to use Clenshaw's recurrence formula (§5.5), effecting the two processes simultaneously. Applied to the Chebyshev series (5.8.9), the recurrence is

$$
\begin{aligned}
d_{m+1} &\equiv d_m \equiv 0 \\
d_j &= 2xd_{j+1} - d_{j+2} + c_j \qquad j = m-1, m-2, \dots, 1 \\
f(x) \equiv d_0 &= xd_1 - d_2 + \frac{1}{2}c_0
\end{aligned}
\tag{5.8.11}
$$

```
float chebev(float a, float b, float c[], int m, float x)
```

Chebyshev evaluation: All arguments are input. `c[0..m-1]` is an array of Chebyshev coefficients, the first `m` elements of `c` output from `chebft` (which must have been called with the same `a` and `b`). The Chebyshev polynomial $\sum_{k=0}^{\mathtt{m}-1} \mathtt{c}_k T_k(y) - \mathtt{c}_0/2$ is evaluated at a point $\mathtt{y} = [\mathtt{x} - (\mathtt{b}+\mathtt{a})/2]/[(\mathtt{b}-\mathtt{a})/2]$, and the result is returned as the function value.

```
{
    void nrerror(char error_text[]);
    float d=0.0,dd=0.0,sv,y,y2;
    int j;

    if ((x-a)*(x-b) > 0.0) nrerror("x not in range in routine chebev");
    y2=2.0*(y=(2.0*x-a-b)/(b-a));         Change of variable.
    for (j=m-1;j>=1;j--) {                Clenshaw's recurrence.
        sv=d;
        d=y2*d-dd+c[j];
        dd=sv;
    }
    return y*d-dd+0.5*c[0];               Last step is different.
}
```

If we are approximating an *even* function on the interval $[-1, 1]$, its expansion will involve only even Chebyshev polynomials. It is wasteful to call `chebev` with all the odd coefficients zero [1]. Instead, using the half-angle identity for the cosine in equation (5.8.1), we get the relation

$$T_{2n}(x) = T_n(2x^2 - 1) \tag{5.8.12}$$

Thus we can evaluate a series of even Chebyshev polynomials by calling `chebev` with the even coefficients stored consecutively in the array `c`, but with the argument x replaced by $2x^2 - 1$.

An odd function will have an expansion involving only odd Chebysev polynomials. It is best to rewrite it as an expansion for the function $f(x)/x$, which involves only even Chebyshev polynomials. This will give accurate values for $f(x)/x$ near $x = 0$. The coefficients c'_n for $f(x)/x$ can be found from those for $f(x)$ by recurrence:

$$\begin{aligned} c'_{N+1} &= 0 \\ c'_{n-1} &= 2c_n - c'_{n+1}, \qquad n = N, N-2, \dots \end{aligned} \tag{5.8.13}$$

Equation (5.8.13) follows from the recurrence relation in equation (5.8.2).

If you insist on evaluating an odd Chebyshev series, the efficient way is to once again use `chebev` with x replaced by $y = 2x^2 - 1$, and with the odd coefficients stored consecutively in the array `c`. Now, however, you must also change the last formula in equation (5.8.11) to be

$$f(x) = x[(2y - 1)d_2 - d_3 + c_1] \tag{5.8.14}$$

and change the corresponding line in `chebev`.

CITED REFERENCES AND FURTHER READING:

Clenshaw, C.W. 1962, *Mathematical Tables*, vol. 5, National Physical Laboratory, (London: H.M. Stationery Office). [1]

Goodwin, E.T. (ed.) 1961, *Modern Computing Methods*, 2nd ed. (New York: Philosophical Library), Chapter 8.

Dahlquist, G., and Bjorck, A. 1974, *Numerical Methods* (Englewood Cliffs, NJ: Prentice-Hall), §4.4.1, p. 104.

Johnson, L.W., and Riess, R.D. 1982, *Numerical Analysis*, 2nd ed. (Reading, MA: Addison-Wesley), §6.5.2, p. 334.

Carnahan, B., Luther, H.A., and Wilkes, J.O. 1969, *Applied Numerical Methods* (New York: Wiley), §1.10, p. 39.

5.9 Derivatives or Integrals of a Chebyshev-approximated Function

If you have obtained the Chebyshev coefficients that approximate a function in a certain range (e.g., from chebft in §5.8), then it is a simple matter to transform them to Chebyshev coefficients corresponding to the derivative or integral of the function. Having done this, you can evaluate the derivative or integral just as if it were a function that you had Chebyshev-fitted *ab initio*.

The relevant formulas are these: If $c_i,\ i = 0, \dots, m-1$ are the coefficients that approximate a function f in equation (5.8.9), C_i are the coefficients that approximate the indefinite integral of f, and c'_i are the coefficients that approximate the derivative of f, then

$$C_i = \frac{c_{i-1} - c_{i+1}}{2(i-1)} \qquad (i > 1) \tag{5.9.1}$$

$$c'_{i-1} = c'_{i+1} + 2(i-1)c_i \qquad (i = m-1, m-2, \dots, 2) \tag{5.9.2}$$

Equation (5.9.1) is augmented by an arbitrary choice of C_0, corresponding to an arbitrary constant of integration. Equation (5.9.2), which is a recurrence, is started with the values $c'_m = c'_{m-1} = 0$, corresponding to no information about the $m+1$st Chebyshev coefficient of the original function f.

Here are routines for implementing equations (5.9.1) and (5.9.2).

```
void chder(float a, float b, float c[], float cder[], int n)
Given a,b,c[0..n-1], as output from routine chebft §5.8, and given n, the desired degree
of approximation (length of c to be used), this routine returns the array cder[0..n-1], the
Chebyshev coefficients of the derivative of the function whose coefficients are c.
{
    int j;
    float con;

    cder[n-1]=0.0;                              n-1 and n-2 are special cases.
    cder[n-2]=2*(n-1)*c[n-1];
    for (j=n-3;j>=0;j--)
        cder[j]=cder[j+2]+2*(j+1)*c[j+1];       Equation (5.9.2).
    con=2.0/(b-a);
    for (j=0;j<n;j++)                           Normalize to the interval b-a.
        cder[j] *= con;
}
```

```
void chint(float a, float b, float c[], float cint[], int n)
Given a,b,c[0..n-1], as output from routine chebft §5.8, and given n, the desired degree
of approximation (length of c to be used), this routine returns the array cint[0..n-1], the
Chebyshev coefficients of the integral of the function whose coefficients are c. The constant of
integration is set so that the integral vanishes at a.
{
    int j;
    float sum=0.0,fac=1.0,con;

    con=0.25*(b-a);                          Factor that normalizes to the interval b-a.
    for (j=1;j<=n-2;j++) {
        cint[j]=con*(c[j-1]-c[j+1])/j;       Equation (5.9.1).
```