5.10 Polynomial Approximation from Chebyshev Coefficients

You may well ask after reading the preceding two sections, "Must I store and evaluate my Chebyshev approximation as an array of Chebyshev coefficients for a transformed variable y? Can't I convert the c_k's into actual polynomial coefficients in the original variable x and have an approximation of the following form?"

$$f(x) \approx \sum_{k=0}^{m-1} g_k x^k \tag{5.10.1}$$

Yes, you can do this (and we will give you the algorithm to do it), but we caution you against it: Evaluating equation (5.10.1), where the coefficient g's reflect an underlying Chebyshev approximation, usually requires more significant figures than evaluation of the Chebyshev sum directly (as by `chebev`). This is because the Chebyshev polynomials themselves exhibit a rather delicate cancellation: The leading coefficient of $T_n(x)$, for example, is 2^{n-1}; other coefficients of $T_n(x)$ are even bigger; yet they all manage to combine into a polynomial that lies between ± 1. *Only* when m is no larger than 7 or 8 should you contemplate writing a Chebyshev fit as a direct polynomial, and even in those cases you should be willing to tolerate two or so significant figures less accuracy than the roundoff limit of your machine.

You get the g's in equation (5.10.1) from the c's output from `chebft` (suitably truncated at a modest value of m) by calling in sequence the following two procedures:

```
#include "nrutil.h"

void chebpc(float c[], float d[], int n)
```
Chebyshev polynomial coefficients. Given a coefficient array `c[0..n-1]`, this routine generates a coefficient array `d[0..n-1]` such that $\sum_{k=0}^{n-1} d_k y^k = \sum_{k=0}^{n-1} c_k T_k(y) - c_0/2$. The method is Clenshaw's recurrence (5.8.11), but now applied algebraically rather than arithmetically.
```
{
    int k,j;
    float sv,*dd;

    dd=vector(0,n-1);
    for (j=0;j<n;j++) d[j]=dd[j]=0.0;
    d[0]=c[n-1];
    for (j=n-2;j>=1;j--) {
        for (k=n-j;k>=1;k--) {
            sv=d[k];
            d[k]=2.0*d[k-1]-dd[k];
            dd[k]=sv;
        }
        sv=d[0];
        d[0] = -dd[0]+c[j];
        dd[0]=sv;
    }
    for (j=n-1;j>=1;j--)
        d[j]=d[j-1]-dd[j];
    d[0] = -dd[0]+0.5*c[0];
    free_vector(dd,0,n-1);
}
```

```
void pcshft(float a, float b, float d[], int n)
```
Polynomial coefficient shift. Given a coefficient array d[0..n-1], this routine generates a coefficient array g[0..n-1] such that $\sum_{k=0}^{\mathtt{n-1}} \mathrm{d}_k y^k = \sum_{k=0}^{\mathtt{n-1}} g_k x^k$, where x and y are related by (5.8.10), i.e., the interval $-1 < y < 1$ is mapped to the interval $\mathrm{a} < x < \mathrm{b}$. The array g is returned in d.
```
{
    int k,j;
    float fac,cnst;

    cnst=2.0/(b-a);
    fac=cnst;
    for (j=1;j<n;j++) {             First we rescale by the factor const...
        d[j] *= fac;
        fac *= cnst;
    }
    cnst=0.5*(a+b);                 ...which is then redefined as the desired shift.
    for (j=0;j<=n-2;j++)            We accomplish the shift by synthetic division. Synthetic
        for (k=n-2;k>=j;k--)            division is a miracle of high-school algebra. If you
            d[k] -= cnst*d[k+1];        never learned it, go do so. You won't be sorry.
}
```

CITED REFERENCES AND FURTHER READING:

Acton, F.S. 1970, *Numerical Methods That Work*; 1990, corrected edition (Washington: Mathematical Association of America), pp. 59, 182–183 [synthetic division].

5.11 Economization of Power Series

One particular application of Chebyshev methods, the *economization of power series*, is an occasionally useful technique, with a flavor of getting something for nothing.

Suppose that you are already computing a function by the use of a convergent power series, for example

$$f(x) \equiv 1 - \frac{x}{3!} + \frac{x^2}{5!} - \frac{x^3}{7!} + \cdots \tag{5.11.1}$$

(This function is actually $\sin(\sqrt{x})/\sqrt{x}$, but pretend you don't know that.) You might be doing a problem that requires evaluating the series many times in some particular interval, say $[0, (2\pi)^2]$. Everything is fine, except that the series requires a large number of terms before its error (approximated by the first neglected term, say) is tolerable. In our example, with $x = (2\pi)^2$, the first term smaller than 10^{-7} is $x^{13}/(27!)$. This then approximates the error of the finite series whose last term is $x^{12}/(25!)$.

Notice that because of the large exponent in x^{13}, the error is *much smaller* than 10^{-7} everywhere in the interval except at the very largest values of x. This is the feature that allows "economization": if we are willing to let the error elsewhere in the interval rise to about the same value that the first neglected term has at the extreme end of the interval, then we can replace the 13-term series by one that is significantly shorter.

Here are the steps for doing so:

1. Change variables from x to y, as in equation (5.8.10), to map the x interval into $-1 \le y \le 1$.
2. Find the coefficients of the Chebyshev sum (like equation 5.8.8) that exactly equals your truncated power series (the one with enough terms for accuracy).
3. Truncate this Chebyshev series to a smaller number of terms, using the coefficient of the first neglected Chebyshev polynomial as an estimate of the error.