Rational Functions

You evaluate a rational function like

$$R(x) = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu} \tag{5.3.4}$$

in the obvious way, namely as two separate polynomials followed by a divide. As a matter of convention one usually chooses $q_0 = 1$, obtained by dividing numerator and denominator by any other q_0. It is often convenient to have both sets of coefficients stored in a single array, and to have a standard function available for doing the evaluation:

```
double ratval(double x, double cof[], int mm, int kk)
```
Given mm, kk, and cof[0..mm+kk], evaluate and return the rational function (cof[0] + cof[1]x + ⋯ + cof[mm]x^mm)/(1 + cof[mm+1]x + ⋯ + cof[mm+kk]x^kk).
```
{
    int j;
    double sumd,sumn;            Note precision! Change to float if desired.

    for (sumn=cof[mm],j=mm-1;j>=0;j--) sumn=sumn*x+cof[j];
    for (sumd=0.0,j=mm+kk;j>=mm+1;j--) sumd=(sumd+cof[j])*x;
    return sumn/(1.0+sumd);
}
```

CITED REFERENCES AND FURTHER READING:

Acton, F.S. 1970, *Numerical Methods That Work*; 1990, corrected edition (Washington: Mathematical Association of America), pp. 183, 190. [1]

Mathews, J., and Walker, R.L. 1970, *Mathematical Methods of Physics*, 2nd ed. (Reading, MA: W.A. Benjamin/Addison-Wesley), pp. 361–363. [2]

Knuth, D.E. 1981, *Seminumerical Algorithms*, 2nd ed., vol. 2 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley), §4.6. [3]

Fike, C.T. 1968, *Computer Evaluation of Mathematical Functions* (Englewood Cliffs, NJ: Prentice-Hall), Chapter 4.

Winograd, S. 1970, *Communications on Pure and Applied Mathematics*, vol. 23, pp. 165–179. [4]

Kronsjö, L. 1987, *Algorithms: Their Complexity and Efficiency*, 2nd ed. (New York: Wiley). [5]

5.4 Complex Arithmetic

As we mentioned in §1.2, the lack of built-in complex arithmetic in C is a nuisance for numerical work. Even in languages like FORTRAN that have complex data types, it is disconcertingly common to encounter complex operations that produce overflows or underflows when both the complex operands and the complex result are perfectly representable. This occurs, we think, because software companies assign inexperienced programmers to what they believe to be the perfectly trivial task of implementing complex arithmetic.

Actually, complex arithmetic is not *quite* trivial. Addition and subtraction are done in the obvious way, performing the operation separately on the real and imaginary parts of the operands. Multiplication can also be done in the obvious way, with 4 multiplications, one addition, and one subtraction,

$$(a + ib)(c + id) = (ac - bd) + i(bc + ad) \tag{5.4.1}$$

(the addition before the i doesn't count; it just separates the real and imaginary parts notationally). But it is sometimes faster to multiply via

$$(a + ib)(c + id) = (ac - bd) + i[(a + b)(c + d) - ac - bd] \tag{5.4.2}$$

which has only three multiplications (ac, bd, $(a + b)(c + d)$), plus two additions and three subtractions. The total operations count is higher by two, but multiplication is a slow operation on some machines.

While it is true that intermediate results in equations (5.4.1) and (5.4.2) can overflow even when the final result is representable, this happens only when the final answer is on the edge of representability. Not so for the complex modulus, if you are misguided enough to compute it as

$$|a + ib| = \sqrt{a^2 + b^2} \qquad \text{(bad!)} \tag{5.4.3}$$

whose intermediate result will overflow if either a or b is as large as the square root of the largest representable number (e.g., 10^{19} as compared to 10^{38}). The right way to do the calculation is

$$|a + ib| = \begin{cases} |a|\sqrt{1 + (b/a)^2} & |a| \geq |b| \\ |b|\sqrt{1 + (a/b)^2} & |a| < |b| \end{cases} \tag{5.4.4}$$

Complex division should use a similar trick to prevent avoidable overflows, underflow, or loss of precision,

$$\frac{a + ib}{c + id} = \begin{cases} \dfrac{[a + b(d/c)] + i[b - a(d/c)]}{c + d(d/c)} & |c| \geq |d| \\[2ex] \dfrac{[a(c/d) + b] + i[b(c/d) - a]}{c(c/d) + d} & |c| < |d| \end{cases} \tag{5.4.5}$$

Of course you should calculate repeated subexpressions, like c/d or d/c, only once.

Complex square root is even more complicated, since we must both guard intermediate results, and also enforce a chosen branch cut (here taken to be the negative real axis). To take the square root of $c + id$, first compute

$$w \equiv \begin{cases} 0 & c = d = 0 \\[1ex] \sqrt{|c|}\sqrt{\dfrac{1 + \sqrt{1 + (d/c)^2}}{2}} & |c| \geq |d| \\[2ex] \sqrt{|d|}\sqrt{\dfrac{|c/d| + \sqrt{1 + (c/d)^2}}{2}} & |c| < |d| \end{cases} \tag{5.4.6}$$

Then the answer is

$$\sqrt{c+id} = \begin{cases} 0 & w = 0 \\ w + i\left(\dfrac{d}{2w}\right) & w \neq 0, c \geq 0 \\ \dfrac{|d|}{2w} + iw & w \neq 0, c < 0, d \geq 0 \\ \dfrac{|d|}{2w} - iw & w \neq 0, c < 0, d < 0 \end{cases} \tag{5.4.7}$$

Routines implementing these algorithms are listed in Appendix C.

CITED REFERENCES AND FURTHER READING:

Midy, P., and Yakovlev, Y. 1991, *Mathematics and Computers in Simulation*, vol. 33, pp. 33–49.

Knuth, D.E. 1981, *Seminumerical Algorithms*, 2nd ed., vol. 2 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley) [see solutions to exercises 4.2.1.16 and 4.6.4.41].

5.5 Recurrence Relations and Clenshaw's Recurrence Formula

Many useful functions satisfy recurrence relations, e.g.,

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x) \tag{5.5.1}$$

$$J_{n+1}(x) = \frac{2n}{x}J_n(x) - J_{n-1}(x) \tag{5.5.2}$$

$$nE_{n+1}(x) = e^{-x} - xE_n(x) \tag{5.5.3}$$

$$\cos n\theta = 2\cos\theta\cos(n-1)\theta - \cos(n-2)\theta \tag{5.5.4}$$

$$\sin n\theta = 2\cos\theta\sin(n-1)\theta - \sin(n-2)\theta \tag{5.5.5}$$

where the first three functions are Legendre polynomials, Bessel functions of the first kind, and exponential integrals, respectively. (For notation see [1].) These relations are useful for extending computational methods from two successive values of n to other values, either larger or smaller.

Equations (5.5.4) and (5.5.5) motivate us to say a few words about trigonometric functions. If your program's running time is dominated by evaluating trigonometric functions, you are probably doing something wrong. Trig functions whose arguments form a linear sequence $\theta = \theta_0 + n\delta$, $n = 0, 1, 2, \ldots$, are efficiently calculated by the following recurrence,

$$\begin{aligned} \cos(\theta+\delta) &= \cos\theta - [\alpha\cos\theta + \beta\sin\theta] \\ \sin(\theta+\delta) &= \sin\theta - [\alpha\sin\theta - \beta\cos\theta] \end{aligned} \tag{5.5.6}$$