Evaluation of Function

Numerical Methods for IT

Polynomials

Given a polynomial $P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$
and $v \in \mathbb{R}$.
Evaluation $P_n(v)$?

We have
$P_n(v) = a_0 + a_1v + a_2v^2 + \cdots + a_{n-1}v^{n-1} + a_nv^n$
$= a_0 + v(a_1 + a_2v + \cdots + a_{n-1}v^{n-2} + a_nv^{n-1})$
$= a_0 + v * P_{n-1}(v)$, where
$P_{n-1}(v) = b_0 + b_1v + \cdots + b_{n-2}v^{n-2} + b_{n-1}v^{n-1}, b_i = a_{i+1}, i = 0, \dots, n-2$

Let $P(n) = a[0 \dots n]$ is a 1-dimensional array that represents the computer representation of the polynomial $P_n(x)$. Given a value v. To evaluate $P_n(v)$, we can represent the recursive form as follows:

$$P(i, v) = \begin{cases} a[0], & i = 0 \\ a[0] + v * P(i, v), & i \geq 1 \end{cases}.$$

Rational functions

To evaluate a rational function like:

$$R(x) = \frac{P_\mu(x)}{Q_v(x)} = \frac{p_0+p_1x+\cdots+p_\mu x^\mu}{q_0+q_1x+\cdots+q_v x^v}.$$

Given w, to evaluate $R(w)$, we can use the polynomial evaluation routine to:

(1) $num \leftarrow P(\mu, w)$

(2) $dom \leftarrow Q(v, w)$

(3) Return num/dom

Evaluation of Continued Fractions

Continued fractions are often powerful ways of evaluating functions that occur in scientific applications.

A continued fraction looks like:

$$f(x) = b_0 + \frac{a_1}{b_1 + \frac{a_2}{b_2 + \frac{a_3}{b_3 + \cdots}}} \equiv b_0 + \frac{a_1}{b_1 +} \frac{a_2}{b_2 +} \frac{a_3}{b_3 +} \ldots.$$

For example, the continued fraction representation of tangent function is: $\tan x = \frac{x}{1-} \frac{x^2}{3-} \frac{x^2}{5-} \frac{x^2}{7-} \ldots$

Let f_n denote the result of $f(x)$ with coefficients through a_n and b_n. Then

$f_n = \frac{A_n}{B_n}$,

where A_n, B_n are given by the following recurrence:

$A_{-1} \equiv 1\,; B_{-1} \equiv 0$

$A_0 \equiv b_0\,; B_0 \equiv 1$

$A_j = b_j A_{j-1} + a_j A_{j-2}\,; B_j = b_j B_{j-1} + a_j B_{j-2}\,;$

$j = 1,2, \ldots, n.$

Steed's method.

. Set $f_0 = b_0. If\ b_0 = 0, set\ f_0 = \epsilon.$

. Set $C_0 = f_0, D_0 = 0.$

. For $j = 1,2, \ldots$:

. Set $D_j = b_j + a_j D_{j-1}$

. If $D_j = 0, set\ D_j = \epsilon$

. Set $C_j = b_j + a_j / C_{j-1}.$

. If $C_j = 0, set\ C_j = \epsilon.$

. Set $D_j = 1/D_j, \Delta = C_j D_j, f_j = f_{j-1}\Delta_j.$

. If $|\Delta_j - 1| < \epsilon_0, then\ exit.$

Chebyshev Approximation

The Chebyshev polynomial of degree n is denoted $T_n(x) = \cos(n\ arcos\ x)$.

It can be combined with trigonometric identities to yield explicit expression for $T_n(x)$:
$T_0(x) = 1\,; T_1(x) = x\,; T_2(x) = 2x^2 - 1\,; T_3(x) = 4x^3 - 3x^2\,; T_4(x) = 8x^4 - 8x^2 + 1 \ldots$
$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \forall n \geq 1$ (*).
Suppose N is so large. We consider the truncated approximation:
$f(x) \approx \left[\Sigma_{k=0}^{m-1} c_k T_k(x)\right] - \frac{1}{2} c_0$ (**) where $c_j = \frac{2}{N} \Sigma_{k=0}^{N-1} f\left[\cos\left(\frac{\pi\left(k+\frac{1}{2}\right)}{N}\right)\right] \cos\left(\frac{\pi j\left(k+\frac{1}{2}\right)}{N}\right)$.

Now that we have the Chebyshev coefficients, how to evaluate the approximation?
We can use the above recurrence relation (*) to generate values for $T_k(x)$ from $T_0 = 1, T_1 = x$, while also accumulating the sum of (**).
It is better to use Clenshaw's recurrence formula:
$d_{m+1} \equiv d_m \equiv 0$
$d_j = 2xd_{j+1} - d_{j+2} + c_j, j = m-1, m-2, \ldots, 1$
$f(x) \equiv d_0 = xd_1 - d_2 + \frac{1}{2} c_0$.

Polynomial Approximation from Cheb. Coeff.

Convert the c_k's into actual polynomial coefficients in the original variable x and have an approximation of the following form:

$$f(x) \approx \Sigma_{k=0}^{m-1} g_k x^k, a \leq x \leq b.$$

Given a coefficients c[0..n-1], this routine returns d[0..n-1]: $\Sigma_{k=0}^{n-1} d_k y^k = \Sigma_{k=0}^{n-1} c_k T_k(y) - c_0/2$.

```
Int k,j;
Doub sv;
VecDoub d(m), dd(m);
For (j=0;j<m;j++): d[0]=c[m-1];
For (j=m-2;j>0;j--):
        for (k=m-2;k>0;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=m-2;j>0;j--): d[j]=d[j-1]-dd[j];
d[0]=-dd[0]+0.5*c[0];
Return d
```

Given a coefficients c[0..n-1], this routine returns g[0..n-1]: $\Sigma_{k=0}^{n-1} d_k y^k = \Sigma_{k=0}^{n-1} g_k x^k - c_0/2$. The interval -1<y<1 is mapped to the interval a<x<b.

```
Int k,j,n=d.size();
Doub cnst=2.0/(b-a), fac=cnst;
VecDoub d(m), dd(m);
For (j=0;j<n;j++):
        d[j] *= fac
        fac *= cnst
cnst = 0.5*(a+b)
For (j=0; j< n-2; j++):
        for (k=n-2; k>=j; k--):
                d[k] -= cnst*d[k+1]
```