Some Experiments with Evaluation of Legendre Polynomials

Richard Fateman
Computer Science
University of California, Berkeley

December 4, 2007

Abstract

Common practice is to recommend evaluation of polynomials by Horner's rule. Here's an example where it is fast but doesn't work nearly as accurately as another fairly easy method. Can a method for Legendre polynomials be both fast and accurate? [1]

1 Legendre Polynomials

A substantial literature has grown up around the useful notion of orthonormal polynomials and one prime example is that of Legendre polynomials (also known as Legendre Functions of the First Kind and usually written as P_n) which we encountered most recently in looking at formulas for Gaussian quadrature. In this case we wanted to evaluate them at particular points and we have a choice of how to do so. The polynomials can be defined in various ways, but one popular method uses the recurrence (for integer $n \geq 0$):

$$P_n(x) \ := \ \frac{(2n-1)xP_{n-1} - (n-1)P_{n-2}}{n}, \quad P_0 = 1, \quad P_1 = x$$

Another method is to expand this expression as a polynomial in x, extract the coefficients, and use Horner's rule.

Let us try as an example, $P_5(x)$ which is

$$\frac{63\,x^5 - 70\,x^3 + 15\,x}{8}$$

Using Horner's Rule it can be expressed as

$$\frac{x\,\left(x^2\,\left(63\,x^2 - 70\right) + 15\right)}{8}$$

or by performing the indicated division:

$$x\,\left(x^2\,\left(7.875\,x^2 - 8.75\right) + 1.875\right)$$

The recurrence, on the other hand, requires following a program. Here it is expressed in Macsyma:

[1] We have previously observed that Chebyshev polynomials (T_n)can be calculated using a recurrence that computes T_{n+m} from T_n and T_m. For this, see papers by Fateman and by Koepf [1, 3].

```
h(x):=block([p0,p1,p2,p3,p4],
   p0:1,
   p1:x,
   p2:(3*x*p1-p0)/2,
   p3:(5*x*p2-2*p1)/3,
   p4:(7*x*p3-3*p2)/4,
      (9*x*p4-4*p3)/5);
```

Normally one would not write this out but express this as a loop (see Appendix) or perhaps recursively. Even concealing the computation of coefficients (e.g. $(2n-1)$ and $n-1$), this seems to take considerably more multiplications than Horner's rule: 11 vs 4, and twice as many additions. Horner's rule is often recommended as an efficient and usually numerically accurate way of arranging computations.

2 Do we get the same answers though?

Consider the 20th Legendre polynomial,

$(34461632205\, x^{20} - 167890003050\, x^{18} + 347123925225\, x^{16} + \cdots)/262144$

Alternatively, we can express the 20th Legendre polynomial as a list in terms of its coefficients, and use Horner's rule. The coefficients look like this (we have converted them to double-float precision):

$\{131460.694137573, -640449.535552979, 1324172.68838501, \cdots\}$

Exact computation tells us that

$$P_{20}(1/2) = -\frac{13292650571}{274877906944}$$

This is about - 0.04835838106374.

Using Horner's Rule with exact rational coefficients and a double-float $x = 1/2$ gives - 0.048358*39600128* where we have indicated the inaccurate digits as italics. This computation converts the exact rational coefficients to the type of the argument x, as needed, so all the computation was done in double-float.

Putting the computation over a common denominator, using integer coefficients in the numerator, and then evaluating numerator using Horner's rule, we get - 0.0483583811*2922*

Just running the recurrence in double-floats gives - 0.04835838106*736*, which has **two more correct digits**.

Hypothesis: Don't evaluate Legendre polynomials by Horner's Rule, unless you are not particularly concerned with accuracy.

We searched for more extreme examples.

Consider P_{20} evaluated exactly at $x = 99/100$ to

$$\frac{-1181643375269313506361069299324346732887555 59}{52428800000000000000000000000000000000000000}$$

which is about - 0.22538058762918729903432260 5.

Horner's Rule evaluates this polynomial of degree 10 in x^2 using 10 adds and 10 multiplies, plus one squaring. Call that a total cost of 21. Unfortunately, the answer obtained in this way at $x = 0.99$ is - 0.2253*5407978638*, where the digits in italics are incorrect.

Computing the same value by running the recurrence for degree 20 uses, for each of the 18 iterations, 4 floating-point multiplies, one divide, one add, and also a few integer operations. Ignoring the integer operations, call the cost about 108 operations.

The answer is - 0.22538058762918*8*, correct, to almost all digits.(the final 8 should be rounded up to a 9).

If we run the iteration using 100-digit bigfloat arithmetic, the answer has 99 correct digits.

Experiments with graphing rapidly show that the worst behavior for Horner's Rule is when $|x|$ is just less than 1, where the zeros of P_n for different values of n are quite close. Indeed, the value at 1, computed exactly, is 1, but using Horner's rule is 1.29.

A question of some interest to us is whether we can evaluate Legendre Polynomials using some other scheme which takes no more arithmetic than Horner's rule, yet maintains the same numerical accuracy as the recurrence. For our motivating application, it is particularly important to have accurate values of Legendre polynomials near their zeros, so that these zeros can be accurately computed as a component of generating Gaussian quadrature formulas of various orders.

One possibility is to shift the Legendre polynomial, essentially re-expressing it as a Taylor series centered at 1. In this case the Horner's rule expansion is computed relative to $y = 1 - x$ (or by anti-symmetry at the other end of the unit interval), and accuracy is very high at (say) $x = 99/100$ or $y = 1/100$. This rule does not have the symmetry of expansions about zero, and in particular the Horner's rule at order 20 has 21 non-zero coefficients, not just 10, requiring twice as much arithmetic. This is less than the recurrence, but with similar accuracy to the recurrence in a limited area. If it were really as accurate, a reasonable tradeoff might be to use the expansion around 1 for numbers with absolute value in the range 0.5 to 1. Programs using this technique are also indicated in the appendix. Unfortunately the polynomial evaluation techniques, computed using any standard fixed-precision floating-point arithmetic, just do not appear as smooth functions, monotonic in appropriate intervals. As such they probably cannot be used reliably for (say) zero-finding. Their unfortunate behavior can easily be confirmed using graphics software: A close look at a plot shows the computation by recurrence producing a smooth curve, a consequence of its accuracy, but any of several Horner's rule computations produces a jagged graph at some region between 0 and 1. [2].

References

[1] R. Fateman, Lookup tables, recurrences and complexity. Proc. of ISSAC 89, ACM Press, New York, 1989, 68–73.

[2] Course Notes and Solutions for Math 128, February 2004. `http://www.cs.berkeley.edu/~wkahan/Math128/M128Bsoln09Feb04.pdf`.

[3] W. Koepf. Efficient Computation of Chebyshev Polynomials in Computer Algebra, `http://www.mathematik.uni-kassel.de/~koepf/cheby.pdf`.

3 Appendix: Programs

These experiments were done with the Macsyma / Maxima, computer algebra system.

```
(
/* define a recurrence for Legendre_p polynomials*/

lp(q,x):= block([p0:1,p1:x,pn:x],  /*fast and 100 percent accurate if x is rat(z), say.*/
if (q=0) then 1 else if (q=1) then x else
  (   for n:2 thru q do
  (pn: 1/n*(x*(2*n-1)*p1-(n-1)*p0),
   p0:p1,
   p1:pn),
pn)),
```

```
/* Make a Horner's rule version of a legendre polynomial*/

list22horner(L,var,ans):=  /*evaluate a list as a polynomial using Horner's Rule */
   if L=[] then ans else list22horner(rest(L),var,var*ans+first(L)),
kill(lglistz),
/* keep a list of the non-zero coefficients, memoized */
lglistz[n]:=block([r:[]], for i in poly2list(lp(n,'x),'x) do
                   if i#0 then r:cons(i, r), reverse(r)),

/*Compute nth legendre_p at x  using Horner's rule and the coefficients
in lglistz. */
clg(n,x) :=block([y:x^2],
   (if oddp(n) then x else 1) *  list22horner(lglistz[n],y,0)),

/*taylor series */
tay1[n](y):=''horner(subst(-y,x-1,taylor(lp(n,'x),'x,1,n)),y),
/* value of a legendre polynomial near 1 and minus 1.*/
lpnear1(n,x):=tay1[n](1-x) )$
lpnearm1(n,x):=-tay1[n](x-1) )$
lpnear0(n,x):=clg(n,x)$

/* Other programs */
(g[0](x):=1, g[1](x):=x, g[n](x):= (1/n)*(x*(2*n-1)*g[n-1](x)-(n-1)*g[n-2](x)))$
sp[L](x):= sum((binomial(L,k)*binomial(-L-1,k)/2^k*(1-x)^k,k,0,L)$

/* make a list of expansion coefficients around 1-x, and also the denominator */
ex1(n):= block([h:poly2list(ratnumer(sp[n](1-'y)),'y)],   [h, last(h)])$

/* similar, around 0 */
ex0(n):= block([s:rat(sp[n]('y))], [poly2list(ratnumer(s),'y),ratdenom(s)])$
```