Some Experiments with Evaluation of Legendre Polynomials

Richard Fateman
Computer Science
University of California, Berkeley

July 2, 2011

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 uses for orthonormal polynomials. Here we look at the example of Legendre polynomials (in particular, Legendre Functions of the First Kind) usually written as P_n, which we encountered most recently in looking at formulas for Gaussian quadrature. In this case we need to evaluate them at particular points, and must choose a sequence of operations for the computation. The polynomials can be defined in various algebraically equivalent ways, but with distinct numerical-roundoff behaviors. One popular method directly uses the well-known defining 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 which might occur to someone with a computer algebra system handy and interested in minimizing the operation count 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 by a constant:

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

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

```
h(x):=block([p0,p1,p2,p3,p4],
   p0:1,
   p1:x,
```

[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].

```
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. Furthermore, Horner's rule is often recommended as an efficient and usually numerically accurate way of arranging computations[2].

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):

[131460.6941413879, 0, -640449.5355606079, 0, 1324172.688388824, 0, -1513340.215301514, 0, 1043287.572669983, 0, -444238.5793304443, 0, 114889.2877578735, 0, -17020.63522338867, 0, 1276.54764175415, 0, -37.00138092041016, 0, 0.176197052001953][3]

Exact computation tells us that

$$P_{20}(1023/1024) =$$

$$\frac{339170423999949688117475113219809282601234425556486659961924007629}{42124916667422874679167211073468172927558038160219644501724391014 4}$$

This is about 0.805153934612399. The number we chose, $f = 1023/1024$ is easily and exactly represented as a binary floating point number, and so it does not really matter if we use it as an exact rational or as 0.9990234375000000..0000.

Using Horner's Rule with exact rational coefficients and f as a rational number and performing rational arithmetic gives the huge exact rational number shown above. Performing double-float arithmetic gives

0.805153934*63615* where we have typeset the incorrect digits in italics.

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

But where? Actually, evaluating Legendre polynomials outside of [-1,1] is probably irrelevant in applications; clearly as we approach 1 and -1, something bad happens in terms of cancellations in the Horner's rule formulation. For some randomly chosen point in the interval [-1,1], nothing much bad happens, but when the result is expected to be close to zero, the Horner's rule cancellations just don't quite work. Let us choose an approximation to a zero of P_{20}, $z = 0.636053680726474$

Consider $P_{20}(z)$. We can evaluate this using the recursive definition, in double precision, and get

2.1898038937706589e-13

We can extend the (binary float) z by 250 additional zero bits and evaluate again.

We get the same 2.1895736219...e-13. So our answer was good to about 4 decimal digits. Now let us try the Horner's Rule version. Using z we get a number that has the wrong sign: -2.300937218535637*e-14. If we extend z again and stuff it into Horner's Rule, we get many more correct digits but not as many as the recursive version. (Instead of 100 decimal digits, about 83 digits).

Horner's Rule, carefully implemented, can evaluate P_{20}, a polynomial of degree 10 in x^2 using 10 adds and 10 multiplies, plus one squaring. Call that a total cost of 21.

[2]There is a subtlety in expressing a numeric computation in a computer algebra system. It is, in some respects, like a highly-optimizing compiler. That is, writing x+x might be changed to 2*x, and (1.0+x)-1 changed to x. Thus instruction counts and even answers might be, in some cases, at variance with expectations. In a sequence of expressions in *a program*, such simplifications are usually *not* done, and the order of operations is more-or-less the same as a conventional program

[3]An earlier version of this paper unfortunately used *single-precision floats* and were inaccurate after about 8 digits, This was an error by the author in mistakenly assuming that all implementations of Macsyma and Maxima were using double-precision throughout. This blunder resulted in some claims of inaccuracy that were false. In fact there are problems in some regions, but not all that were previously claimed.

Computing the same value by running the recurrence for degree 20 uses, for each of the 18 iterations, 3 floating-point multiplies, one divide, one add, and also a few integer operations. Ignoring the integer operations, call the cost about 90 operations.

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 a zero or 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].

I am grateful to Axel Vogt for directing me to a technical report [4] with a solution for accurate evaluation of orthogonal polynomials especially in the neighborhood of points where the recurrence is sensitive to rounding errors. While Legendre polynomials using the recurrence above are not problematical, the recurrence for Chebyshev polynomials is less stable near $|x| = 1$ and for Laguerre polynomials, the rounding errors accumulate for small x. For the record, the modified recursion for Legendre polynomials looks like this:

```
(d[0](x):=x-1, p[1](x):=x,
 d[k](x):= (2*k+1)/(k+1)*(x-1)*p[k](x)+ k/(k+1)*d[k-1](x),
 p[k](x):=d[k-1](x)+p[k-1](x))
```

or rewritten as an iterative program:

```
lp2(q,x):= block([pk:x, dk:x-1],
if (q=0) then 1 else
  (for k:1 thru q-1 do
    (dk: 1/(k+1) *((2*k+1)*(x-1)*pk +k*dk),
     pk:dk+pk),
  pk))
```

which boils down to just about the same computation. As long as we are tinkering with this formulation, we make explicit the possibility of precomputing the rational constants. Here's a rearranged program that defines the needed constants (in Maxima, these are stored when first computed):

```
(h[k]:=(2*k+1)/(k+1), /*precompute constants, store in arrays */
 g[k]:=k/(k+1),

lp3(q,x):= block([pk:x, dk, xm1:x-1],
  if (q=0) then 1 else
    (dk:xm1,
     for k:1 thru q-1 do
      (dk: h[k]*xm1*pk +g[k]*dk, /*each iteration is 3 mults, 2 adds, */
```

```
  pk:dk+pk),                /* and 2 array refs */
pk)))
```

As indicated, the technique provides a better payoff for accuracy for other orthogonal polynomials [4].

3 Conclusion

Horner's rule may be faster, but is certainly not as accurate as a recurrence-based algorithm for computing Legendre polynomials. The recurrence is remarkably stable, and is recommended especially near 1 or zeros of the polynomial.

4 Acknowledgment

Thanks to Dragan Stoikovitch for spotting a significant error in an earlier version of this paper, as noted in the footnote concerning single/double precision.

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.

[4] P. Levire and R. Piessens,"A note on the evaluation of orthogonal polynomials using recurrence relations", Report TW 74, Dept. Computer Science, Katholieke Universiteit Leuven, September, 1985, 9 pages. http://www2.cs.kuleuven.be/~nalag/papers/paul/report74/.

5 Appendix: Programs

These experiments were done with the 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.
Also quite accuracy for floats. */
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*/

l2h(L,var):= /*evaluate a list as a polynomial using Horner's Rule */
if L=[] then 0 else first(L)+ var*l2h(rest(L),var)$


kill(lglistz)$
/* lglistz[n] is  a list of all the coefficients of P[n] memoized */
```

```
/* utility program to make a list of coefficients in any polynomial h, in variable v */

poly2list(h,v):=
 block([keepfloat:true,p:ratexpand(h),ans:[]],
 for i:hipow(p,v) step -1 thru 1 do
   (q:bothcoef(p,v^i),
    ans:cons(part(q,1),ans),
    p:part(q,2)),
 cons(p, ans))$

lglistz[n]:=poly2list(lp(n,'x),'x)$


/*Compute nth legendre_p at x using Horner's rule and the coefficients
in lglistz. */

clg(n,x):= l2h(lglistz[n],x)$

/*trials */

/* large errors near x=1, 10^-10 */

plot2d( '(clg(20,x)-lp(20,x)),[x,0,1]);

/*near a root of p[20], differences more than 10^-12 */

g:0.636053680726 47$
plot2d( '(clg(20,x+g)-lp(20,x+g)),[x,-10^-8,10^-8])$

/* not especially near a root, differences below about 10^-13 */

plot2d( '(clg(20,x+0.5)-lp(20,x+0.5)),[x,-10^-8,10^-8])$

/*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)])$
```