Verifying the accuracy of polynomial approximations in HOL

John Harrison*

University of Cambridge Computer Laboratory
New Museums Site, Pembroke Street
Cambridge CB2 3QG, England

Abstract. Many modern algorithms for the transcendental functions rely on a large table of precomputed values together with a low-order polynomial to interpolate between them. In verifying such an algorithm, one is faced with the problem of bounding the error in this polynomial approximation. The most straightforward methods are based on numerical approximations, and are not prima facie reducible to a formal HOL proof. We discuss a technique for proving such results formally in HOL, via the formalization of a number of results in polynomial theory, e.g. squarefree decomposition and Sturm's theorem, and the use of a computer algebra system to compute results that are then checked in HOL. We demonstrate our method by tackling an example from the literature.

1 Introduction

Many algorithms for the transcendental functions such as exp, sin and ln in floating point arithmetic are based on table lookup. Suppose that a transcendental function $f(x)$ is to be calculated. Values of $f(a_i)$ are prestored for some approximately equally-spaced family of values a_i. When $f(x)$ is required, there are enough a_i that x will lie very close to one of them. By the use of some sort of interpolation formula, whose exact nature depends on f, one can reduce the problem of calculating $f(x)$ to calculating some similar function $g(x')$ for a small argument $x' = x - a_k$. For example, when calculating e^x, we can evaluate $e^{x'}$ and then multiply it by the prestored constant e^{a_k}, since $e^x = e^{a_k + x'} = e^{a_k} e^{x'}$. Because x' is so small, the appropriate interpolating function $g(x')$ can be approximated adequately by a fairly low-order polynomial. As part of a verification effort for such an algorithm that will be reported elsewhere (Harrison 1997), we need to prove some mathematical results about the accuracy of such polynomial approximations.

Suppose we are interested in approximating a function $f(x)$ over a closed interval $[a, b]$, i.e. $\{x \mid a \leq x \leq b\}$, by a polynomial of degree n. The natural choice is the polynomial $p(x)$ that, out of all polynomials of degree n, has the best *minimax* error behaviour, i.e. minimizes the maximum magnitude of the error:

* Work supported by the EPSRC grant 'Floating Point Verification'

$$||f(x) - p(x)||_\infty = sup_{a \le x \le b}|f(x) - p(x)|$$

A fundamental theorem of Chebyshev states that such a polynomial always exists and has an interesting 'equal ripple' property: the maximum value of $|f(x) - p(x)|$ is attained at some $n + 2$ points in the interval such that the sign of $f(x) - p(x)$ alternates between successive points. However the theorem does not yield an analytical expression for the coefficients, or the maximum error. Certainly the truncated Taylor expansion is normally not the best approximation in this sense. The truncated expansion in terms of *Chebyshev polynomials* is usually close. In fact there are special cases, typically for rational functions, where the coefficients in such a series are those for a best approximation, or can be converted into them analytically — see Rivlin (1962) for example. But these are no good for the functions we are interested in, and the only general methods known are iterative numerical ones that rely on numerically approximating extrema as part of the intermediate steps. The standard algorithm is usually attributed to Remes (1934), and also often referred to as the *exchange algorithm*.

2 Our approach

In summary, then, it seems difficult to calculate the maximum error in the best approximation by analytical methods. Besides, any value so found would need slight but messy modifications when the coefficients are stored as approximate floating point numbers; typically the required coefficients will not be exactly representable. Instead, we allow the approximation to be calculated outside the theorem prover by arbitrary means, with coefficients truncated to the form actually used. Then we prove a theorem characterizing the maximum error in this polynomial approximation. Our procedure has the added advantage that we can take from the literature the polynomial approximations actually considered by other workers, without worrying about whether they coincide with the best approximations as we would calculate them.

The function $f(x)$ that we are interested in approximating is typically fairly well-behaved, and therefore so is the error $e(x) = f(x) - p(x)$ for a polynomial $p(x)$. In particular, we can normally assume that $e(x)$ is differentiable at least once throughout the interval. Moreover, it is usually easy to arrive at a (pessimistic) bound B for the absolute value of the derivative $e'(x)$. This isn't the case for pathological functions such as $x\ sin(1/x)$ near zero, but for the basic transcendentals, it is easy over the intervals typically considered. By the mean value theorem (proved in HOL), this ensures that B is a fixed bound for uniform continuity, meaning that:

$$\forall x, x' \in [a, b].\ |e(x) - e(x')| \le B|x - x'|$$

Therefore it is easy in principle to approximate the extremal values of $e(x)$ to an accuracy ϵ simply by dividing up the interval into a set of values a_i at most ϵ/B apart and finding the maximum of the $e(a_i)$. In fact, this method works for a weaker notion of uniform continuity, normally assumed in the definition of 'computable real function', and for functions of more than one variable (Pour-El and Richards 1980).

Nevertheless, the above is too expensive in practice, since it involves a large number of calculations. This is all the more true in HOL, where a single evaluation of a transcendental function to moderate accuracy can take many seconds (Harrison 1996). A much more refined approach is to use the fact that if a function is differentiable in an interval, then its maximum and minimum are each attained either at the endpoints of the interval or at one of the points of zero derivative. This is easy to prove in HOL using the existing real analysis theory. So if we can approximate the points of zero derivative to within ϵ/B, we can get the maximum and minimum to accuracy ϵ by evaluating $e(x)$ only at these points.

However, we are now faced with the problem of locating the points of zero derivative. It's generally pretty easy in practice to approximate all the derivative's roots x_i numerically, e.g. using a Newton iteration. Provided they are simple roots, we can then find (rational) numbers α_i and β_i such that $\alpha_i < x_i < \beta_i$ and the signs of $e'(\alpha_i)$ and $e'(\beta_i)$ differ. Now the intermediate value theorem (already proved in HOL) assures us that there is a root between α_i and β_i. If we also ensure that $\beta_i - \alpha_i \leq \epsilon/B$ this approximation serves for the later calculations. However there are two difficulties. First, if x_i is a double root, or in general a root of even order, we will no longer have different signs for $e'(\alpha_i)$ and $e'(\beta_i)$ and it will be more complicated to prove in HOL that there is a root between those points. A more serious problem is that we have no HOL proof that we have located *all* the roots of $e'(x)$, and without that we have no way of proving our final result.

We are not aware of any simple general theory that can prove exactly how many roots an equation involving transcendental functions has. Therefore the above approach seems very difficult. Instead we modify it as follows.[2] We approximate $f(x)$ by a truncated Taylor series $t(x)$, choosing the degree so large that truncation incurs an error well below the tightness of the bound we are trying to achieve, say $\epsilon/2$. (An alternative would be to truncate the Chebyshev expansion, but although that might lead to a lower-degree polynomial, the analytical details are messier.) Now instead of working with $f(x)$ above we work with $t(x)$, and we then need only locate all the zeros of a *polynomial* $e'(x)$. Moreover this polynomial has rational coefficients, since floating point numbers are rational and the coefficients in all the typical Taylor series are rational. This problem is tractable. It is not hard to prove (see later) that a polynomial of degree n can have at most n roots. If we find exactly n, our task is necessarily complete. This favourable situation can't be relied on, but in general we can calculate the exact number of roots using a classical result due to Sturm. The main difficulties are that we must prove this theorem in HOL and then apply it. In principle, it would be possible to use the general quantifier elimination procedure for the reals that has already been developed by Harrison (1996), but in practice it is far too slow.

[2] Thanks to David Wheeler for this suggestion.

3 Polynomials in HOL

The ring $\mathbb{R}[X]$ of univariate polynomials over the reals may be characterized abstractly via a universal property. The most natural concrete definition is that it is the space of functions $c : \mathbb{N} \to \mathbb{R}$ such that for all sufficiently large n we have $c_n = 0$.[3] We define the degree of c, written $\partial(c)$ or $deg(c)$, to be the least d such that $\forall i > d. c_i = 0$. The function c is thought of as $\Sigma_{i=0}^{deg(c)} c_i x^i$, and this motivates the definition of addition and negation componentwise, e.g. $(b + c)_i = b_i + c_i$, and multiplication by:

$$(bc)_n = \Sigma_{i=0}^{n} b_i c_{n-i}$$

Rather than define a type of polynomials in this way, we deal directly with the *functions* $\mathbb{R} \to \mathbb{R}$ determined by polynomials. For a polynomial c, this is the function that takes x to $\Sigma_{i=0}^{deg(c)} c_i x^i$. Actually, in the case of the reals, nothing is lost by this, since every polynomial function determines a unique polynomial, and vice versa.[4] We define a function `poly` that maps a list of coefficients $[c_0; \ldots; c_n]$ into the corresponding polynomial function $\lambda x.\ \Sigma_{i=0}^{n} c_i x^i$:[5]

```
|- (poly [] x = &0) ∧ (poly (CONS h t) x = h + x * poly t x)
```

Note that `poly` is not injective, since lists with trailing zeros give the same function, and so the salient notion of equality for polynomial functions arising from two lists l_1 and l_2 is that `poly` $l_1 =$ `poly` l_2, or equivalently $\forall x.$`poly` $l_1\ x =$ `poly` $l_2\ x$, rather than simply $l_1 = l_2$. If desired, we could introduce an equivalence relation between lists corresponding to this notion. We can now define arithmetic operations on lists by primitive recursion, including addition (`++`):

```
|- ([] ++ l2 = l2) ∧
   (CONS h t ++ l2 = if (l2 = []) then CONS h t
                     else CONS (h + (HD l2) (t ++ (TL l2))))
```

multiplication by a constant (`##`):

```
|- (c ## [] = []) ∧
   (c ## CONS h t = CONS (c * h) (c ## t))
```

and multiplication of two polynomials (`**`):

```
|- ([] ** l2 = []) ∧
   (CONS h t ** l2 =
      if t = [] then h ## l2 else h ## l2 ++ CONS (&0) (t ** l2))
```

[3] We write c_n rather than $c(n)$, since it is probably more natural.

[4] This depends on the fact that the underlying ring is infinite. For example in a 2-element ring the polynomials x and x^2 are distinct even though they both determine the same function.

[5] The symbol `&` is the injection $\mathbb{N} \to \mathbb{R}$, and hence appears in real numeral constants.

From these, we can define negation and exponentiation in an obvious way:

```
|- neg p = --(&1) ## p

|- (p exp 0 = [&1]) ∧
   (p exp (SUC n) = p ** p exp n)
```

We also need to define differentiation of polynomials. This can be expressed purely as a formal manipulation of the list of coefficients, conveniently done via an auxiliary function:

```
|- (diff_aux n [] = []) ∧
   (diff_aux n (CONS h t) = CONS (&n * h) (diff_aux (SUC n) t))

|- diff l = if l = [] then [] else diff_aux 1 (TL l)
```

We then prove that all these operations do indeed work as expected on the corresponding polynomial functions. These are all straightforward single or double structural inductions on lists:

```
|- ∀p1 p2 x. poly (p1 ++ p2) x = poly p1 x + poly p2 x

|- ∀p c x. poly (c ## p) x = c * poly p x

|- ∀p x. poly (neg p) x = --(poly p x)

|- ∀x p1 p2. poly (p1 ** p2) x = poly p1 x * poly p2 x

|- ∀p n x. poly (p exp n) x = (poly p x) pow n

|- ∀l x. ((poly l) diffl (poly (diff l) x)) x
```

The last of these uses a notion from the HOL real analysis theory: it says that `poly` l is locally differentiable at x with derivative `poly` (`diff` l) x there. From the above results, it is straightforward to verify all the basic properties that one would expect of polynomial functions, e.g. that addition and multiplication are commutative, and that all polynomial functions are infinitely continuously differentiable, and that the product rule holds for derivatives of polynomial products. Note a subtle point however: not all identities hold at the level of *lists*. For example we have:

```
|- ([] ** [&1; &2] = []) ∧
   ([&1; &2] ** [] = [&0])
```

We do however define a function that deletes trailing zeros, and prove that it does not affect the polynomial function. Proving the opposite, that if l_1 and l_2 give rise to the same polynomial function then their normalized forms are equal, is harder, and it's convenient to wait till we've proved the key property that a nontrivial polynomial only has finitely many roots.

```
|- (normalize [] = []) ∧
   (normalize (CONS h t) = if normalize t = [] then
                             if h = &0 then [] else [h]
                           else CONS h (normalize t))

|- ∀p. poly (normalize p) = poly p
```

If desired, we could redefine the addition operation to use this at the end, and then all operations would return normalized results for normalized inputs. However we do not do that, merely using it to define the notion of degree:

```
|- ∀p. degree p = PRE (LENGTH (normalize p))
```

Although we don't really want to rely on representation details, it's very convenient to get us started to prove one key result at the level of lists, rather than polynomial functions. This is that if a polynomial derived from a nonempty list is divided by a linear polynomial, there is a constant remainder:

```
|- ∀t h. ∃q r. CONS h t = [r] ++ [-- a; &1] ** q
```

(Note that `[-- a; &1]` is thought of as '$x - a$'.) From this, we find that if a is a root of a nontrivial polynomial, then the polynomial is divisible by $(x - a)$ *at the level of lists*:

```
|- ∀a p. (poly p a = &0) = (p = []) ∨ (∃q. p = [-- a; &1] ** q)
```

Now since we also have:

```
|- ∀q. LENGTH ([-- a; &1] ** q) = SUC (LENGTH q)
```

it is a straightforward induction on the length of a generating list to show that a nonzero polynomial can only have finitely many roots, bounded, in fact, by the length of the list, and therefore by the degree of a normalized polynomial:

```
|- ∀p. ¬(poly p = poly [])
       ⟹ (∃i. ∀x. (poly p x = &0)
                  ⟹ (∃n. n <= (LENGTH p) ∧ (x = i n)))
```

or more abstractly, using a notion from the HOL set theory:

```
|- ∀p. ¬(poly p = poly []) ⟹ FINITE {x | poly p x = &0}
```

Several key results flow immediately from this fact. The polynomial ring has no zerodivisors, i.e. if the product of two polynomials is trivial, so is one of the factors. Hence cancellation holds. We can also get a more constructive definition of what it means for a polynomial to be zero.

```
|- ∀p q. (poly (p ** q) = poly []) =
         (poly p = poly []) ∨ (poly q = poly [])

|- ∀p q r. (poly (p ** q) = poly (p ** r)) =
           (poly p = poly []) ∨ (poly q = poly r)

|- ∀p. (poly p = poly []) = FORALL (λc. c = &0) p
```

where here, and later, we use the logical operations on lists defined by:

```
|- (FORALL P [] = T) ∧
   (FORALL P (CONS h t) = P h ∧ FORALL P t)

|- (EX P [] = F) ∧
   (EX P (CONS h t) = P h ∨ EX P t)
```

After this, there are no real mathematical difficulties left in getting the results we want; it is merely necessary to accumulate a number of lemmas. Many of these are concerned with divisibility of polynomials, which we define, no longer at the list level, by:

```
|- p1 divides p2 = ∃q. poly p2 = poly (p1 ** q)
```

There are various obvious properties collected, e.g:

```
|- ∀p. p divides p

|- ∀p q r. p divides q ∧ q divides r ⟹ p divides r

|- ∀p q m n. p exp n divides q ∧ m <= n ⟹ p exp m divides q

|- ∀p q r. p divides q ∧ p divides r ⟹ p divides q ++ r

|- ∀p q r. p divides q ∧ p divides q ++ r ⟹ p divides r

|- ∀p q. (poly p = poly []) ⟹ q divides p
```

A slightly less trivial property is that the linear polynomials are prime elements in the polynomial ring:

```
|- ∀a p q. [a; &1] divides p ** q =
           [a; &1] divides p ∨ [a; &1] divides q
```

A major result in what follows is that for each nonzero polynomial p and real number a, we have a welldefined 'order' n for a such that $p(x)$ is divisible by $(x-a)^n$ but not by $(x-a)^{n+1}$. Hence a is a root of p precisely if its order is nonzero. The definition as a choice term is trivial:

```
|- order a p =
       εn. [-- a; &1] exp n divides p ∧
           ¬([-- a; &1] exp SUC n divides p)
```

Some tedious but straightforward proofs are required to show that it has the required properties for any nontrivial polynomial:

```
|- ∀p a. ¬(poly p = poly [])
         ⟹ ([-- a; &1] exp (order a p)) divides p ∧
             ¬(([-- a; &1] exp (SUC(order a p))) divides p)

|- ∀p a. (poly p a = &0) = (poly p = poly []) ∨ ¬(order a p = 0)

|- ∀p a n. [-- a; &1] exp n divides p =
           (poly p = poly []) ∨ n <= order a p

|- ∀p a. ¬(poly p = poly [])
         ⟹ (∃q. (poly p = poly ([-- a; &1] exp (order a p) ** q)) ∧
                 ¬([-- a; &1] divides q))
```

4 Squarefree decomposition

Given a polynomial $p(x)$, we are interested in locating its roots. As we have already mentioned, given an isolating interval $[\alpha, \beta]$ for a root such that $p(\alpha)$ and $p(\beta)$ have opposite signs, we can straightforwardly prove in HOL from the intermediate value theorem and the known continuity of polynomials that there must be at least one root in the interval. However, this doesn't work for roots of even order (in the precise sense of 'order' we have defined above), since in that case the function has the same sign at either side of the root. This is one reason why we would prefer the polynomial to have no multiple real roots. Another reason is that Sturm's theorem is easier to prove for polynomials without multiple real roots — this is actually the only form we have proved in HOL.

Therefore, it's convenient to start off by finding the so-called 'squarefree decomposition' of p. To get this, we divide p by $gcd(p, p')$ where p' is the derivative polynomial. It is easy to see that the resulting polynomial has the same roots but that they are all of order 1. If a is a root of $p(x)$ of order $n+1$, then we have $p(x) = (x-a)^{n+1}q(x)$ for $q(a) \neq 0$. Differentiating, we have $p'(x) = (n+1)(x-a)^n q(x) + (x-a)^{n+1}q'(x)$. This is obviously divisible by $(x-a)^n$ but not by $(x-a)^{n+1}$, for that would, by the primality of linear factors, imply that $(x-a)$ divides $q(x)$, a contradiction since we assumed $q(a) \neq 0$. Hence a has order $min(n, n+1) = n$ in $gcd(p, p')$, and so order 1 in $p/gcd(p, p')$.

We start by defining what it means for a polynomial to be 'squarefree', i.e. to have no quadratic factors. Actually, since we only consider real factors, we prepend the letter 'r':

```
|- ∀p. rsquarefree p =
       ¬(poly p = poly []) ∧ (∀a. (order a p = 0) ∨ (order a p = 1))
```

We prove without much trouble that this is equivalent to the fact that p and p' have no common factors, and deduce a simple decomposition theorem for squarefree polynomials:

```
|- ∀p. rsquarefree p =
       (∀a. ¬((poly p a = &0) ∧ (poly (diff p) a = &0)))

|- ∀p a.
      rsquarefree p ∧ (poly p a = &0)
      ⟹ (∃q. (poly p = poly ([-- a; &1] ** q)) ∧ ¬(poly q a = &0))
```

Now we need to show that dividing by $gcd(p, p')$ always converts a polynomial p into a squarefree one with the same roots. Rather than defining $gcd(p, p')$ explicitly in HOL, we will simply assume some d such that the following hold for some polynomials r and s, q and e:

$$
\begin{aligned}
p &= qd \\
p' &= ed \\
d &= rp + sp'
\end{aligned}
$$

It is clear that this constrains d to be a gcd of p and p', since any other common divisor of p and p' must divide d. We can calculate the gcd externally together with the coefficients r and s, using the `gcdex` function of the Maple computer algebra system, and get q and e via its division function. These are then plugged into the following HOL theorem for checking.

```
|- ∀p q d e r s.
      ¬(poly (diff p) = poly []) ∧
      (poly p = poly (q ** d)) ∧
      (poly (diff p) = poly (e ** d)) ∧
      (poly d = poly (r ** p ++ s ** diff p))
      ⟹ rsquarefree q ∧
          (∀a. (poly q a = &0) = (poly p a = &0))
```

The proof of this is not too difficult, by deriving a couple of additional facts about how orders of roots interact with operations:

```
|- ∀a p q.
      ¬(poly (p ** q) = poly [])
      ⟹ (order a (p ** q) = order a p + order a q)

|- ∀p a.
      ¬(poly (diff p) = poly []) ∧ ¬(order a p = 0)
      ⟹ (order a p = SUC(order a (diff p)))
```

Now fix a and consider the orders of a in the various polynomials we consider above. We have, assuming p and p' are nontrivial (the latter obviously implies the former):

$$
\begin{aligned}
order_a(p) &= order_a(q) + order_a(d) \\
order_a(p') &= order_a(e) + order_a(d) \\
order_a(d) &\geq min(order_a(p'), order_a(p))
\end{aligned}
$$

These together imply (over the natural numbers) that $order_a(q) = 0$ if $order_a(p) = 0$ and $order_a(q) = 1$ otherwise. The proof is done automatically by HOL's linear natural number arithmetic package, after case-splitting over whether $order_a(p)$ is zero and throwing in the fact that $order_a(p) = order_a(p') + 1$ in the latter case.

5 Sturm's theorem

Sturm's theorem (Benedetti and Risler 1990) gives a precise figure for the number of (distinct) real roots a polynomial has in an interval. Assuming the polynomial has rational coefficients and the endpoints of the interval are rational, it requires only rational arithmetic. The key concept is a *Sturm sequence* for a polynomial p. This is a finite sequence of polynomials, with p as the first element that has certain important properties. Rather than deal abstractly in terms of these properties, we use the 'standard' Sturm sequence directly. This starts with $p_0 = p$ and $p_1 = p'$ and thereafter proceeds by division, with a change of sign, so that $p_i = q_i p_{i+1} - p_{i+2}$ for some quotient q_i, and $deg(p_{i+2}) < deg(p_{i+1})$. Actually we can relax this slightly by rescaling the polynomials to keep the coefficients as simple as possible, provided we do not change their signs. We thus define the property of being a Sturm sequence in HOL as follows.

```
|- (STURM p p' [] = p' divides p) ∧
   (STURM p p' (CONS g gs) = (∃k. &0 < k ∧ p' divides (p ++ k ## g)) ∧
                             degree g < degree p' ∧
                             STURM p' g gs)
```

Note that we separate out the first two elements of the list, to give simpler manipulation, and do not yet assume that p' is the derivative of p. We next define the number of variations in sign of a finite sequence of numbers (or a list in our formalization), not counting zeros:

```
|- (varrec prev [] = 0) ∧
   (varrec prev (CONS h t) =
      if prev * h < &0 then SUC (varrec h t)
      else if h = &0 then varrec prev t else varrec h t)

|- variation l = varrec (&0) l
```

Sturm's theorem involves calculating the number of variations in sign of the polynomials in a Sturm sequence, evaluated at a point. It asserts that the number of roots of p between a and b is the difference in this variation when calculated at a and b respectively. The proof proceeds by analyzing how this variation changes across roots of polynomials in the Sturm sequence. We can break the problem down so that we only need to consider one root at a time. It is easy to prove by induction on the definition of finiteness that any finite set of reals can be laid out in an ordered linear sequence $i_0, \ldots, i_{N-1}$:

```
|- ∀s. FINITE s
       ⟹ (∃i N.
              (∀x. x IN s = (∃k. k < N ∧ (x = i k))) ∧
              (∀k. i k < i (SUC k)))
```

The set of zeros of a list of nontrivial polynomials is finite (optionally in any interval, since any subset of a finite set is finite), by list induction, so we can find such an enumeration of the points that are roots of any of the polynomials in a sequence. Now by considering the mid-points between adjacent i_k, we can split up an interval $[a, b]$ into intervals so that there is at most one root in each (though it may be a root of more than one of the polynomials at once). This is actually quite tedious to prove, since we need to take care at the endpoints.

Now we can confine ourselves to intervals containing at most one root, and such that if the root is one of the endpoints, it is not a root of the starting polynomial p. (This is ruled out for the two endpoints a and b by hypothesis; for internal endpoints this holds for the other polynomials too.) We just need to prove that if such an interval contains no root of p, the variation is the same at both ends, while if there is a root, it decreases by 1 in passing from left to right.

The first is pretty easy, under fairly weak hypotheses. Consider an interval $[a, b]$ and suppose that $p_i(c) \neq 0$ but $p_{i+1}(c) = 0$. Since we have $p_i = q_i p_{i+1} - k p_{i+2}$ for positive k, this means that $p_i(c)$ and $p_{i+2}(c)$ have opposite signs. Since c is the only possible root, neither p_i nor p_{i+2} changes sign throughout the interval, so they have opposite signs at both ends. Now the signs of $p_{i+1}(a)$ and $p_{i+1}(b)$ make no difference to the overall variation between p_i and p_{i+2}, which is one in each case. This argument is easily formalized in HOL using induction, though not pure structural induction. Depending on whether the head element of a list is zero, we either move to the consideration of a list one element or two elements shorter, so we need to perform wellfounded induction on the length of the list. The final result is:

```
|- ∀l f f' c.
       STURM f f' l ∧ a <= c ∧ c <= b ∧
       (∀x. a <= x ∧ x <= b ∧
            EX (λp. poly p x = &0) (CONS f (CONS f' l))
            ⟹ (x = c)) ∧
       ¬(poly f c = &0)
       ⟹ (varrec (poly f a) (MAP (λp. poly p a) (CONS f' l)) =
            varrec (poly f b) (MAP (λp. poly p b) (CONS f' l)))
```

We just need to prove that if the starting polynomial does have a root in the interval, then the variation changes by 1. Now we use an assumption that the starting polynomial is squarefree. Therefore the derivative has no zero in the interval, and so by the above lemma we get no change in variation from the tail of the sequence. We need only consider the change in sign from $p(x)$ to $p'(x)$. In fact there must be exactly one change of sign, because p crosses the axis precisely once, the root being simple. This is easily formalized using the mean value theorem: the derivative does not change sign over the interval, so either the derivative is positive everywhere and the function goes from negative to positive, or vice versa. In either case the result follows. Plugging this together with the lemma for the rest of the sequence, we get the result for a single interval, and so by summing over the intervals, we get the final result:

```
|- ∀f a b l.
      a <= b ∧ ¬(poly f a = &0) ∧ ¬(poly f b = &0) ∧
      rsquarefree f ∧ STURM f (diff f) l
      ⟹ {x | a <= x ∧ x <= b ∧ (poly f x = &0)} HAS_SIZE
          (variation (MAP (λp. poly p a) (CONS f (CONS (diff f) l))) -
           variation (MAP (λp. poly p b) (CONS f (CONS (diff f) l))))
```

where, because the HOL cardinality function is total and hence does not encode finiteness, we use:

```
|- s HAS_SIZE n = FINITE s ∧ (CARD s = n)
```

The main result also holds without the restriction that the starting polynomial be squarefree, but it is more complicated to prove (using the above squarefree case as a lemma) and we do not need it. However, note that the Sturm sequence is up to rescaling a Euclidean division sequence for the polynomial and its derivative. Hence we expect the last term of the Sturm sequence to be a constant polynomial, and this itself implies that the original polynomial must have been squarefree without a separate check. Thus we can sharpen the above slightly, if we rule out a few degenerate cases. The form we actually use, as proved in HOL, is:

```
|- ∀f a b l d.
       a <= b ∧
       ¬(poly f a = &0) ∧
       ¬(poly f b = &0) ∧
       ¬(poly (diff f) = poly []) ∧
       STURM f (diff f) l ∧
       ¬(l = []) ∧
       (LAST l = [d]) ∧
       ¬(d = &0)
       ⟹ {x | a <= x ∧ x <= b ∧ (poly f x = &0)} HAS_SIZE
           variation (MAP (λp. poly p a) (CONS f (CONS (diff f) l))) -
           variation (MAP (λp. poly p b) (CONS f (CONS (diff f) l)))
```

6 Applications

We will consider an example taken from a paper by Tang (1989) giving algorithms for the exponential function in single and double IEEE standard arithmetic. They use polynomials to approximate $e^x - 1$ over the range $[-\frac{ln(2)}{64}, \frac{ln(2)}{64}]$ for the respective precisions. The paper gives the coefficients as IEEE numbers in hexadecimal form. Translated into rational numbers, the approximating polynomial for single precision is:

$$p_{single}(x) = x + \frac{8388676}{2^{24}}x^2 + \frac{11184876}{2^{26}}x^3$$

Tang asserts an error of approximately $2^{-33.2}$ for the approximation, obtained by 'locating numerically all the extreme points of $e^t - 1 - p(t)$ in the interval $[-0.010831, 0.010831]$'. It is this result that we will formalize in HOL. Consider the error in truncated Taylor expansions. There are several versions of Taylor's theorem proved in the HOL real analysis theory, and the most convenient one for our purposes is the one for infinitely everywhere differentiable functions:

```
|- ∀f diff.
       (diff 0 = f) ∧ (∀m x. (diff m diffl diff (SUC m) x) x)
       ⟹ (∀x n.
              ∃t. abs t <= abs x ∧
                  (f x =
                   Sum (0,n) (λm. diff m (&0) / &(FACT m) * x pow m) +
                   diff n t / &(FACT n) * x pow n))
```

Note that `Sum (0,n) f` means $\Sigma_{i=0}^{n-1} f_i$, so the above says that for some t with $|t| \leq |x|$ we have

$$f(x) = (\Sigma_{i=0}^{n-1} \frac{f^{(i)}(0)}{i!} x^i) + \frac{f^{(n)}(t)}{n!} x^n$$

We can sharpen the above to $|t| < |x|$ when $x \neq 0$ and $n \neq 0$, but that is unnecessary here. Instantiating to the exponential function and using the known derivative, we get:

```
|- ∀x n. ∃t. abs t <= abs x ∧
            (exp x = Sum (0,n) (λm. x pow m / &(FACT m)) +
                     exp t / &(FACT n) * x pow n)
```

that is, $e^x = (\Sigma_{i=0}^{n-1} x^i/i!) + e^t x^n/n!$. We need only consider x with $|x| \leq ln(2)/64$. In this case, e^t for $|t| \leq |x|$ is almost 1, so the maximum error from truncating the series before the n^{th} term is approximately $\epsilon_n = (ln(2)/64)^n/n!$. We can see that an additional 3 terms gives an error of about $\epsilon_7 = 2^{-58}$, easily small enough for our purposes. [6]

[6] Note that the $n = 4$ value indicates that the minimax approximation we are concerned with is 5 times as accurate as the truncated Taylor series of the same order, a difference that suffices to make a significant change to the overall error in the algorithm.

$$t(x) = x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6$$

Hence the error polynomial $e(x) = t(x) - p_{single}(x)$ is:

$$e(x) = -\frac{17}{4194304}x^2 - \frac{49}{50331648}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6$$

Differentiating with respect to x yields:

$$e'(x) = -\frac{17}{2097152}x - \frac{49}{16777216}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5$$

This is already squarefree, so the squarefree decomposition is trivial. It has fewer real roots than its full complement of 5, just 3 of them, all in the interval of interest. Hence we need to use Sturm's theorem. The Sturm sequence given by Maple, after rescaling the original derivative and all the members of the sequence to make all the coefficients integers, can easily be checked in HOL by expanding the definitions and doing arithmetic. We can then deduce from Sturm's theorem that there are exactly 3 real roots in the interval concerned. First, we want to isolate them to a reasonable accuracy. For the accuracy of the polynomial bound to be well above the tightness of the error bound required, we choose $\epsilon = 2^{-48}$. It is straightforward to give a crude upper bound on the derivative using the following theorem:

```
|- ∀x k p. abs x <= k ⟹ abs (poly p x) <= poly (MAP abs p) k
```

and we duly find that $B = 2^{-21}$ suffices. Therefore we want to isolate the roots to within $\epsilon/B = 2^{-27}$. Maple offers the following isolating intervals when given the required accuracy: $[0, 0]$, $[\frac{936399}{2^{27}}, \frac{936400}{2^{27}}]$ and $[\frac{-935680}{2^{27}}, \frac{-935679}{2^{27}}]$.

We can now prove that an ordered list of these isolating intervals does indeed include all the roots, using the following general lemma:

```
|- ∀l a b.
     {x | a <= x ∧ x <= b ∧ (poly p x = &0)} HAS_SIZE (LENGTH l) ∧
     recordered a l b ∧
     FORALL (λ(u,v). poly p(u) * poly p(v) <= &0) l
     ⟹ ∀x. a <= x ∧ x <= b ∧ (poly p(x) = &0)
            ⟹ EX (λ(u,v). u <= x ∧ x <= v) l
```

where:

```
|- (recordered a [] b = a <= b) ∧
   (recordered a (CONS h t) b =
       a < FST h ∧ FST h <= SND h ∧ recordered (SND h) t b)
```

As we said, there is already a theorem in HOL asserting that we can maximize a differentiable function in an interval by considering values only at the endpoints and points of zero derivative:

```
|- ∀f f' a b K.
      (∀x. a <= x ∧ x <= b ⟹ (f diffl (f' x)) x) ∧
      abs(f a) <= K ∧ abs(f b) <= K ∧
      (∀x. a <= x ∧ x <= b ∧ (f'(x) = &0) ⟹ abs(f x) <= K)
      ⟹ (∀x. a <= x ∧ x <= b ⟹ abs(f x) <= K)
```

This can be modified to take account of our approximate knowledge of the points of zero derivative:

```
|- ∀f f' l a b.
    (∀x. a <= x ∧ x <= b ⟹ (f diffl f'(x)) x) ∧
    (∀x. a <= x ∧ x <= b ⟹ abs(f'(x)) <= B) ∧
    abs(f a) <= K + B * e ∧ abs(f b) <= K + B * e ∧
    (∀x. a <= x ∧ x <= b ∧ (f'(x) = &0)
        ⟹ EX (λ(u,v). u <= x ∧ x <= v) l) ∧
    FORALL (λ(u,v). a <= u ∧ v <= b ∧
                    abs(u - v) <= e ∧ abs(f(u)) <= K) l
    ⟹ ∀x. a <= x ∧ x <= b ⟹ abs(f(x)) <= K + B * e
```

We now have all the ingredients required by this theorem, so we get the final result for the error in approximating the higher degree Taylor series. We now include the error from the Taylor truncation, using the following theorem to give a crude bound on the error term:

```
|- ∀x. &0 <= x ∧ x <= inv(&2) ⟹ exp(x) <= &1 + &2 * x
```

Thus we get the final result:

```
|- ∀x. --(&10831) / &1000000 <= x ∧ x <= &10831 / &1000000
       ⟹ abs((exp(x) - &1) - (x + (&8388676 / &2 pow 24) * x pow 2 +
                                 &11184876 / &2 pow 26 * x pow 3))
          <= (&23 / &27) * inv(&2 pow 33)
```

7 Conclusion and future work

This paper illustrates how we can incorporate an important form of numerical reasoning into a formal HOL proof. This is appealing since it allows us to conduct the whole verification in a single system, without relying on the correctness of external tools or stepping outside the usual logic. At the same time, the proof is nontrivial, with the development of the material described here taking several weeks' work. In addition, the eventual runtimes are large (over an hour on a fast machine) owing to the extensive need for numerical calculation, which is rather slow when done by pure inference. The difficulty in both these senses is all the more striking when one considers that an informal error analysis of this numerical approximation occupies about two lines of the source paper.

One of the motivations behind ACL2, the successor to NQTHM (Boyer and Moore 1979), is that calculation is an important part of proof in verifications, and deserves to be a key consideration in the design of theorem provers. In one sense, our work suggests that an ordinary theorem prover may be adequate for the task, but it would obviously be preferable to make the calculations here much faster. This could certainly be done in ACL2, which can perform rational arithmetic in proofs at almost the same speed as in the host machine. Nevertheless, ACL2 would not be particularly convenient for the whole proof, since it does not support real numbers. It would be impossible to use many of the proofs here in ACL2, or even to *state* our final theorem, without artificial paraphrases.

A key objective of future work is to automate this class of proofs. We have already automated most of the internal manipulations such as multiplying polynomials, and it would not be difficult to package up the polynomial-bounding as an automatic HOL derived rule. Then we could use it in the future for say ln or sin with minimal effort. The very final part, using the error in the Taylor series, could also be automated, but only with more work. There are optimizations that can be made on a case-by-case basis, which would also be tricky to automate. For example, since the example derivative above only has 3 real roots *in total*, we could evaluate the variation at ± 1 and save some rational arithmetic.

The extension to rational functions would be fairly straightforward since the zeros of these can be located in much the same way. We may consider this if we ever need to deal with rational functions in our future verifications.

References

Benedetti, R. and Risler, J.-J. (1990) *Real algebraic and semi-algebraic sets.* Hermann, Paris.

Boyer, R. S. and Moore, J S. (1979) *A Computational Logic.* ACM Monograph Series. Academic Press.

Harrison, J. (1997) Floating point verification in HOL Light: The exponential function. Unpublished draft, to appear.

Harrison, J. (1996) Theorem proving with the real numbers. Technical Report 408, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK. Author's PhD thesis.

Pour-El, M. B. and Richards, J. I. (1980) *Computability in Analysis and Physics.* Perspectives in Mathematical Logic. Springer-Verlag.

Remes, M. E. (1934) Sur le calcul effectif des polynomes d'approximation de Tchebichef. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, **199**, 337–340.

Rivlin, T. J. (1962) Polynomials of best uniform approximation to certain rational functions. *Numerische Mathematik*, **4**, 345–349.

Tang, P. T. P. (1989) Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, **15**, 144–157.

This article was typeset using the LaTeX macro package with the LLNCS2E class.