CONTS

Pedagogical Uses for Symbolic Algebra in a Numerical Analysis Course

Dr. Dennis Pence
Western Michigan University
Department of Mathematics
Kalamazoo, Michigan USA 49008
dennis.pence@wmich.edu

Abstract: It is very easy to think that a numerical analysis course will use a computer or calculator for only numerical computations. Certainly that is a major emphasis of the course. However there are many opportunities to make use of symbolic algebra in the derivation and analysis of numerical algorithms. I will use the Voyage 200 to demonstrate some of these topics. I tend to prefer the Voyage 200 for my in-class demonstrations because of how easy it is to carry this handheld device and the viewscreen into my classroon (which has no other easily available technology). Similar things can be done on any CAS, and my students often go to a computer lab to complete assignments that involve symbolic mathematics.

Here is one example. I had a high school student doing some independent reading on polynomial interpolation. On a TI-89, he programmed the standard algorithms for computing the divided difference table and then for evaluating the polynomial in the Newton form. Normally the input for the evaluation routine is the floating-point number desired for x in the evaluation of $p(x)$. However when he ran the routine with a symbolic x, he got the symbolic form of the polynomial. There are many other symbolic tasks related to polynomial interpolation that can be explored at the same time you are presenting the numerical algorithms.

Polynomial Interpolation

Consider the problem of finding the polynomial $p(x)$ of degree at most $(n-1)$ that interpolates given data of the form $\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$. We need only assume that the nodes $\{x_i,\ i = 1, 2, ..., n\}$ are distinct. Generally the second coordinates $\{y_i,\ i = 1, 2, ..., n\}$ represent the evaluation of some unknown function $f(x)$, i.e. $f(x_i) = y_i$ for $i = 1, 2, ..., n$. Even though we simply ask for the polynomial to exactly match the function at these nodes, we hold out the hope that the polynomial $p(x)$ will approximate $f(x)$ well in between the nodes (and sometimes even extrapolating beyond the nodes).

The standard process is to represent the polynomial in the Newton form.

$$p(x) = a_1 + a_2(x-x_1) + a_3(x-x_1)(x-x_2) + ... + a_n(x-x_1)(x-x_2)...(x-x_{n-1})$$

The coefficients $\{a_i; i = 1, 2, ..., n\}$ can easily be computed using divided differences. Here is a Voyage 200 program to do this coefficient computation and another to evaluate the resulting polynomial at some value t. A standard reference for these alorithms is Cheney and Kincaid, *Numerical Mathematics and Computing, 2nd Ed.,* Brooks/Cole Publishing Co., 1985, p. 117.

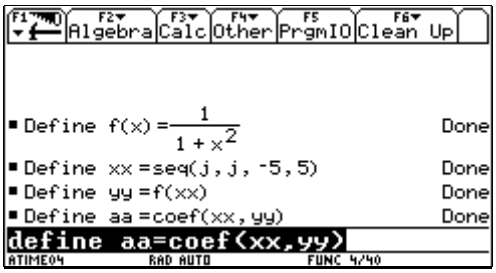
Voyage 200 Program: coef(x,y)

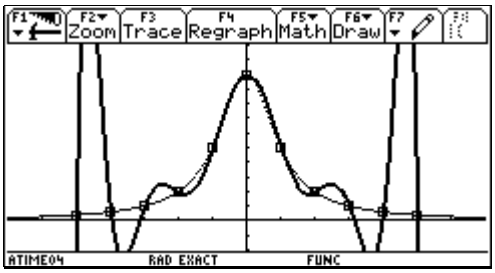
```
Func
  Local i,j,a,n
  dim(x)→n
  For i,1,n
    y[i]→a[i]
  EndFor
  For j,1,n–1
    For i,n,j+1,-1
      (a[i]–a[i-1])/(x[i]–x[i-j])→a[i]
    EndFor
  EndFor
Return a
EndFunc
```

Voyage 200 Program: eval(x,a,t)

```
Func
  Local i,n,val
  dim(x) →n
  a[n]→val
  For i,n–1,1,-1
    val∗(t-x[i])+a[i] →val
  EndFor
Return val
EndFunc
```

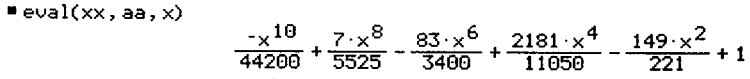
As an example, we will consider the Runge function $f(x)=\frac{1}{1+x^2}$ on the interval [–5,5] using nodes equally-spaced at the integers {–5,–4,–3,–2,–1,0,1,2,3,4,5}.





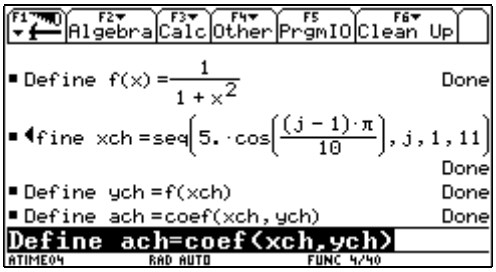
y1(x) = eval(xx,aa,x)

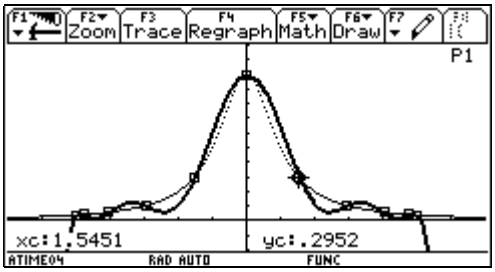
This is a famous example of how bad polynomial interpolation can be. However, when I had a high school student (doing independent study) work through his material and write these programs, he did something I had not anticipated. He ran the program eval(xx,aa,x) in the HOME screen to actually look at the polynomial.

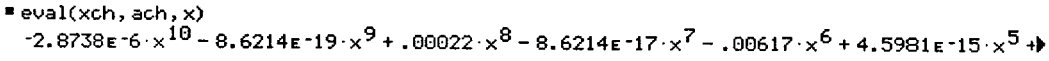


Notice that all of the data values are exact rational numbers, so the coefficients were computed exactly, and we can get an exact polynomial. Here it is interesting that the data was from an even function, our nodes are symmetric about the origin, and the resulting polynomial is exactly even. We always *tell* our students that they are getting an evaluation of a polynomial, but of all their previous experience with polynomials makes them want to see an exact representation as above before they truly believe you are working with a polynomial. Here you can highlight this formula in the history, copy it, and paste it into y2(x) to confirm that it gives the same graph.

Just to finish this example, most texts show other patterns for the nodes here can do better than equal spacing. The Chebyshev nodes (or scaled translations) do nicely here.



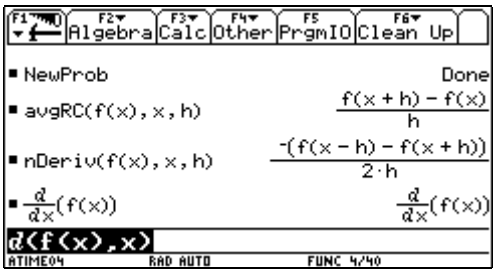


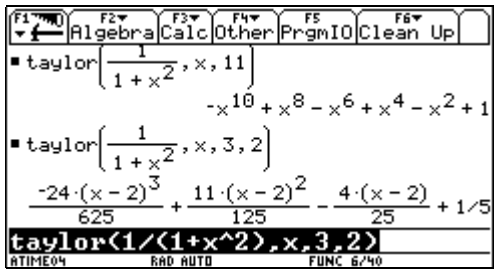


Here the symbolic polynomial has numerical approximations for coefficients. Still we can see that the odd-powered terms have very tiny coefficients (which are virtually numerical zeros).

Numerical Derivatives

The Voyage 200 offers two different numerical derivative approximations (in addition to symbolic differentiation). It is nice to show these symbolically using an unknown function.





Taylor Polynomials

Much of the error estimation in a rigorous numerical analysis course is based upon Taylor polynomial approximations with remainder. The fact that we can quickly get an exact Taylor polynomial is very convenient.

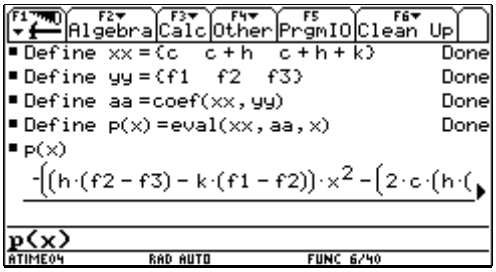
Numerical Integration

Since Riemann sums, the trapezoid rule, and Simpson´s rule are covered in the standard calculus sequence, a numerical analysis course needs to do a little more. At the very last it needs to strive for a deeper understanding of these elementary techniques. Eventually you should do some comparison with the internal routine for doing numerical integration (which is adaptive). The symbolic part might be in the derivation of some rules. The following was partly prompted by a discussion with a geology professor on our campus. He complained that students coming to him did not know how to integrate. I´ve done enough consulting to know to follow up with a few more questions. Was his integrand a discrete function? (Yes). Was his discrete data equally spaced? (Usually.) Seldom does a Calculus 2 student get asked to "integrate" a discrete data set which is not equally spaced.

Supppose we are given discrete data $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ where presumably $y_i \approx f(x_i)$ for i = 1, 2 ,..., n. We can generally assume that the x-values come in increasing order $x_1 < x_2 < ... < x_n$. We desire to approximate $\int_a^b f(x)\,dx$ where usually $a = x_1$ and $b = x_n$. If you have already done polynomial interpolation, then one thing you could do is to integrate the interpolation polynomial instead. However we just saw with the Runge function how far the polynomial might be from the function. Actually this works better than you might think because integration is an *averaging process*. For n greater than 10 or 15, this is not practical.

First you can start out reminding students that the partition for Riemann sums did not need to be equally spaced. $L = \sum_{i=1}^{n-1} y_i (x_{i+1} - x_i)$ gives a left-endpoint sum and $R = \sum_{i=1}^{n-1} y_{i+1} (x_{i+1} - x_i)$ gives a right-endpoint sum. Since there is no way to evaluate the function anywhere else, there are really no other Riemann sums to consider. The trapezoid rule also works over a partition that is not equally spaced (although it is seldom mentioned in calculus texts). Thus $T = \frac{1}{2}(L + R) = \sum_{i=1}^{n-1} \frac{y_i + y_{i+1}}{2} (x_{i+1} - x_i)$ approximates the integral by trapezoids with differing widths. The fun comes when we try to use something like Simpson´s rule (which does critically depend on equal spacing and an odd number of evaluations).

We use the symbolic capabilities of the CAS to derive a "parabola rule" for a data set with three entries $\{(c,f_1),(c+h,f_2),(c+h+k,f_3)\}$. First we find the interpolating quadratic polynomial, and then we integrate it over subintervals. Here we use the two programs from above.



$$p(x)\quad -\frac{(h\cdot(f2-f3)-k\cdot(f1-f2))\cdot x^2-(2\cdot c\cdot(h\cdot(f2-f3)-k\cdot(f1-f2))+h^2\cdot(f2}{}$$

$$\mathrm{f(ans(1),x,c,c+h)}\quad \frac{h\cdot(h^2\cdot(f2-f3)+2\cdot h\cdot k\cdot(f1+2\cdot f2)+3\cdot k^2\cdot(f1+f2))}{6\cdot(h+k)\cdot k}$$

$$\mathrm{f(ans(2),x,c+h,c+h+k)}\quad \frac{(3\cdot h^2\cdot(f2+f3)+2\cdot h\cdot k\cdot(2\cdot f2+f3)-k^2\cdot(f1-f2))\cdot k}{6\cdot h\cdot(h+k)}$$

Just to check, we add the two integrals and look at the special case where $k = h$ to rederive Simson´s rule.

$$\mathrm{ans(2)+ans(1)}\quad \frac{h^3\cdot(f2-f3)+h^2\cdot k\cdot(2\cdot f1+3\cdot f2+f3)+h\cdot k^2\cdot(f1+3\cdot f2+2\cdot f3)-k^3\cdot(f1-f2)}{6\cdot h\cdot k}$$

$$\mathrm{ans(1)|k=h}\quad \frac{h\cdot(f1+4\cdot f2+f3)}{3}$$

We now have something that we can do that is "Simpson-like" for data that is unequally spaced and/or for when n is not odd. Assuming $n \geq 3$, we apply the parabola rule on the beginning of the data set $\{(x_1,y_1),(x_2,y_2),(x_3,y_3)\}$. Note that we do not need to type the "nasty" formula above in the program we are creating. When we need that formula, we can go to the history and copy the formula. Then we move to the program editor and paste in the formula. We continue to take two more data pairs to get another term to add to what we have accumulated thus far. When n is odd, this will work fine and we finish with *full* piecewise parabola intervals. There we approximate $\int_{x_{n-1}}^{x_n} f(x)\,dx$ by taking the integral over this subinterval of the quadratic polynomial interpolating $\{(x_{n-2},y_{n-2}),(x_{n-1},y_{n-1}),\ldots,(x_n,y_n)\}$.

```
:numint(x,y)
:Prgm
:Local n,m,su,h,k,f1,f2,f3,tr
:dim(x)→n
:1→m:0→su
:While n-m≥2
:  x[m+2]-x[m+1]→k
:  x[m+1]-x[m]→h
:  y[m]→f1
:  y[m+1]→f2
:  y[m+2]→f3
:  su+(h^3*(f2-f3)+h^2*k*(2*f1+3*f2+f3)+h*k^2*(f1+3*f2+2*f3)-k^3*(f1-f2))/(6*h*k)→su
:  m+2→m
:EndWhile
:If m<n Then
:  x[m+1]-x[m]→k
:  x[m]-x[m-1]→h
:  y[m-1]→f1
:  y[m]→f2
:  y[m+1]→f3
:  su+(3*h^2*(f2+f3)+2*h*k*(2*f2+f3)-k^2*(f1-f2))*k/(6*h*(h+k))→su
:EndIf
:Σ((y[i]+y[i+1])*(x[i+1]-x[i])/2,i,1,n-1
 )→tr
:ClrIO
:Disp "trapezoid, parabola"
:Disp tr,su
:EndPrgm
```

Of course this "parabola rule" does not, in general have the same error properties as Simpson´s rule (the equal spacing is important in the error analysis). However for equal spacing and n odd we actually get Simpson´s rule in the above program.