Polynomial Interpolation and Approximation on the TI-89

Dennis Pence

Western Michigan University

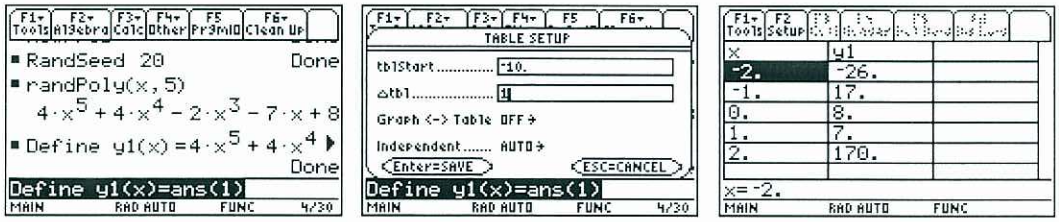
dennis.pence@wmich.edu

Abstract

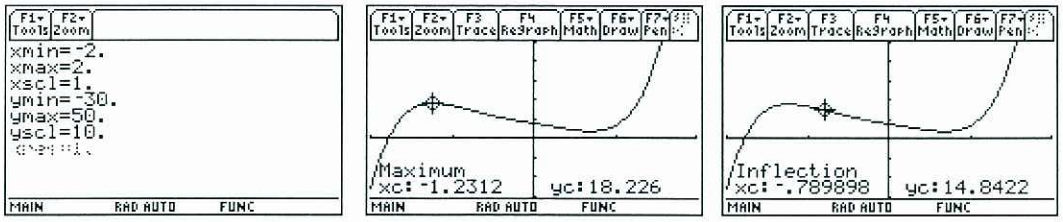
Numerical analysis courses always have a unit on polynomial interpolation and approximation, although much of this topic is anticipated in precalculus, calculus, and linear algebra courses. We will explore the features of the TI-89 that make it an excellent environment to explore polynomials at all of these levels.

Basic Operations with Polynomials

It is nice conduct classroom demonstrations with a somewhat *random example* so that you clearly show that these techniques apply to all polynomials. The TI-89 has a command `randPoly(x,n)` that gives a polynomial in the variable x of degree n that has coefficients that are random integers between -9 and 9. [*Warning:* I thought I was clever once asking students in a Calculus 1 class to issue this command and turn in an assignment about the resulting polynomial. I got only two different polynomials: (1) about a third of the class did an example with me in class and handed in the "second" random polynomial and (2) the rest handed in the "first" random polynomial to come out of a default calculator. The fix is to *first* ask them to enter their individual student identification number as the `RandSeed` and then proceed to anything else random.]



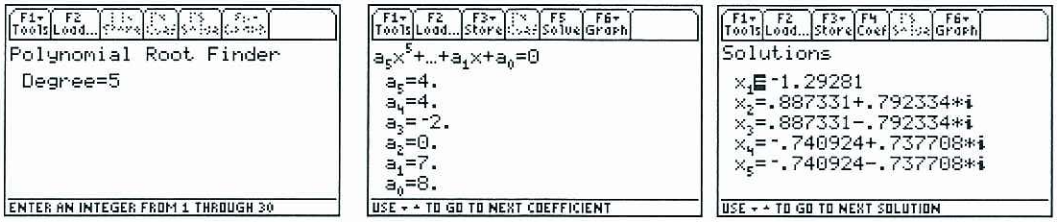
I try to model good calculator habits by almost always looking at a table before setting the viewing window. For a fifth-degree polynomial with single digit coefficients, no "Standard" viewing window is ever appropriate. "ZoomFit" is not helpful either because it does not help us select the appropriate x-range. Here after scrolling through the table above, we settle on $-2 \le x \le 2$.



The interactive F5 MATH graphical commands allow us to find a local minimum, a local maximum, or a point of inflection. From the graph (and our scroll through the table), this particular polynomial

$$y1(x) = 4x^5 + 4x^4 - 2x^3 - 7x + 8$$

seems to have only one real root. There is a very nice free Polynomial Root Finder application that will numerically compute all of the real and complex roots for this polynomial.



We can try to factor a polynomal (such as our original random fifth-degree polynomial). Note that in Auto mode, it moves to numerical factoring when it cannot factor in exact arithmetic.

```
▪ factor(y1(x),x)
  (4.·x+7.13965)·(x^2-1.59982·x+.794668)·(x^2+.814904·x+1.41003)

▪ cFactor(y1(x),x)
  .003906·(4.·x+7.13965)·(4.·x+-3.19963-1.57387·i)·(4.·x+-3.19963+1.57387·i)
                         ·(4.·x+1.62981-4.46141·i)·(4.·x+1.62981+4.46141·i)
```

To truly appreciate the computational issues for accurate root-finding, many numerical analysis textbooks mention the Wilkinson polynomial of degree 20

$$q(x) = (x-1)(x-2)(x-3)\cdots(x-19)(x-20)$$

and how sensitive this is to perturbations in the coefficients. For more details see Wilkinson [2]. In particular, changing the integer coefficient of x^{15} by a very little amount causes the roots to change dramatically (and many become complex). This makes a good project (where you can use the symbolic capabilities to expand to see the specific coefficients of the Wilkinson polynomial).

Lagrange Interpolation

Suppose that we are given a set of distinct *nodes* $\{x_i, i = 1,\ldots,n+1\}$ with $x_1 < x_2 < \cdots < x_{n+1}$ and also a set of *data values* $\{y_i, i = 1,\ldots,n+1\}$. Then the *Lagrange interpolation problem* for the data pairs $\{(x_i, y_i), i = 1,\ldots,n+1\}$ is to find a polynomial $p \in \mathscr{P}_n$ satisfying

$$p(x_i) = y_i,\ i = 1,\ldots,n+1.$$

All introductory numerical analysis textbooks (see Kincaid-Cheney [1]) present this topic and prove existence, uniqueness, and approximation properties for such interpolatory polynomials. Of course high school students learn how to do this in the case $n = 1$ (finding a linear polynomial passing through *two* data pairs).

Existence could be established by proving properties of the following matrix equation.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 & \cdots & x_{n+1}^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n+1} \end{bmatrix}$$

Here the unknowns c_j are the traditional coefficients for a power basis. While the TI-89 can certainly handle these matrix problems, the Vandermonde matrix in the above problem is well-known to be very *ill-conditioned*, particularly for higher degrees. The recommended method is to consider the *Newton form*

$$p(x) = a_0 + a_1(x-x_1) + a_2(x-x_1)(x-x_2) + \cdots + a_n(x-x_1)(x-x_2)\cdots(x-x_n)$$

where the coefficents a_j are efficiently computed using *divided differences*. Here is a TI-89 program `coef` to compute these coefficents and another TI-89 program `eval` to efficiently evaluate the resulting polynomial using *nested multiplication*. In the programs, the *list* structure is used to pass the data to the programs and to return the coefficents.

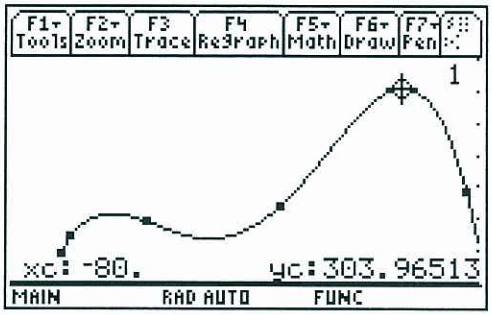
```
:coef(x,y)
:Func
: Local i,j,a,n
: dim(x)→n
: For i,1,n
:  y[i]→a[i]
: EndFor
: For j,1,n-1
:  For i,n,j+1,-1
:   (a[i]-a[i-1])/(x[i]-x[i-j])→a[i]
:  EndFor
: EndFor
:Return a
:EndFunc
```

```
: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
```

Here is an example where polynomial interpolation is not so good. Consider the following table of data values.

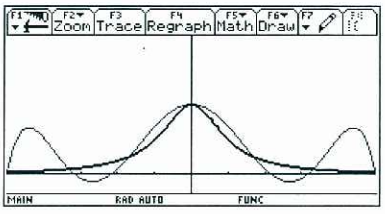
θ	-81.091	-81.058	-80.818	-80.387	-79.789
T	48	72	96	120	144

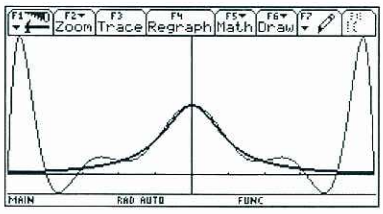
I was given this "real data" by a worker at NASA's Goddard Space Flight Center. In the table, θ represents the geodetic latitude of a satellite in degrees and T represents the time in seconds. He wanted to approximate the time T when the geodetic latitude was -80 degrees. Thus he had found the fourth-degree polynomial $p(\theta)$ interpolating the data in the table, and he evaluated $p(-80)$ to get the estimate of the time. Now most of us would "eye-ball" the table and get a rough estimate that the answer should be at about 132 seconds. Imagine the disappointment of the NASA worker when it turns out that $p(-80) \approx 304$.



If we plot the function $p(\theta)$ in a window $-81.25 \le \theta \le -79.75, \quad 0 \le T \le 350$, we see that while the data seemed to indicate a strictly increasing situation, the interpolating polynomial is definitely not always increasing. There are many alternatives here that seem to give us a more reasonable result. Linear interpolation (with just the last two columns in the table), quadratic interpolation (with the last three columns), and even cubic interpolation (with the last four columns) all give evaluations that fit our expectations better. We can also reverse the roles of θ and T, seeking a fourth-degree polynomial $q(T)$ that interpolates the data (with rows interchanged), and then we need to use something like Newton's method to solve the equation $q(T) = -80$ for the desired time.

We see below the polynomial of degree 6 (on the left) and the polynomial of degree 10 (on the right) that interpolate the Runge function $f(x) = \left(x^2 + 1\right)^{-1}$ at equally-spaced values on the interval $[-5, 5]$ (including the endpoints). This gives another visual demonstration of how things can go wrong with approximations derived by higher degree polynomial interpolations.

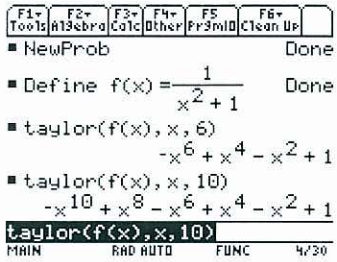


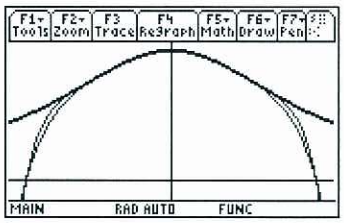


Taylor Polynomials

Calculus students find the tangent line approximation for a function at a point a, which is simply a linear polynomial which matches the function value and the derivative value at a. More generally Taylor polynomials approximate functions by matching the function value and several derivative values. Numerical analysis students study Hermite interpolation using polynomials where they match function and various derivatives of that function at one or more points. The TI-89 has many symbolic

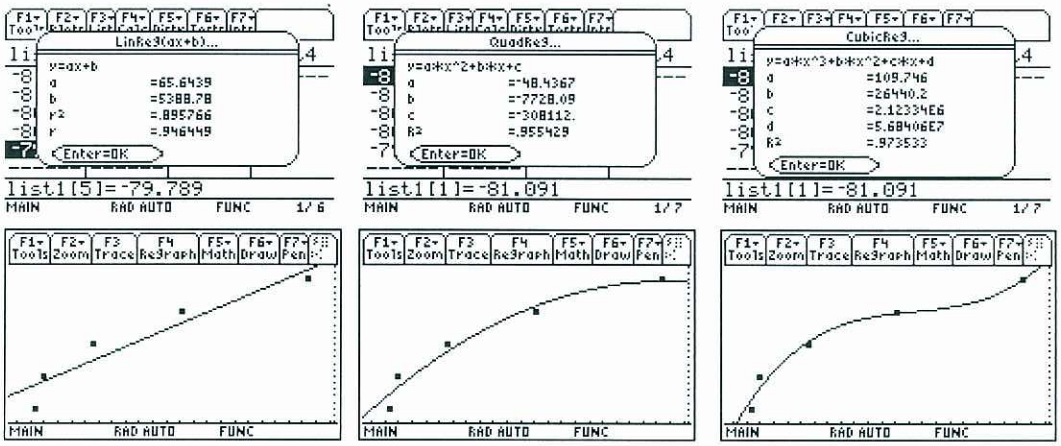
and graphical features that help explore all of these topics, no mater what the level of the course. Here we look at Taylor polynomials about $x = 0$ for the same Runge function.





Polynomial Regression

Linear regression shows up in many courses. In a low level course or a basic statistics course, the calculator is simply used as a tool (a black box) to compute a linear polynomial which approximates a set of data in the least squares sense. Typically multivariable calculus courses derive the formulas for linear regression as an optimization problem in two variables, and linear algebra courses develop linear regression as an orthogonal projection. Least-square-error approximations can use higher degree polynomials, and some of this is provided in the Stats/List Editor application of the TI-89. Here we consider various regressions for the NASA data above.



The point is that the TI-89 is a wonderful tool to explore many questions about polynomial interpolation and approximation. The fact that we can quickly get a visual representation of the data, the functions, and the resulting polynomials allows us to move on to the more difficult questions such as *what should be be using?* in many practical situations.

References

[1] David Kincaid and Ward Cheney, *Numerical Analysis, 2nd ed.* Brooks-Cole, 1996.

[2] James H. Wilkinson, "The Perfidious Polynomial," in *Studies in Numerical Analysis,* Gene H. Golub, editor, MAA Studies in Mathematics, Vol. 24, 1984, p. 1-28.