CSE/MATH 455, Arnold Lab 7 October 31, 1997

The Lagrange Interpolating Polynomial

Preparation: For this lab you need to copy the following files from `~dna2` into your working directory: ginterp.m, ninterp.m, pop.

1. The Matlab program `ginterp` graphically exhibits the Lagrange interpolating polynomial for data you enter. Just use the mouse to enter points on a graph and hit the interpolate button to see the interpolating polynomial. Experiment with `ginterp` to get a feeling for what the interpolating polynomial looks like. Click the button marked "extr." (for extrapolation) on, so that you can see the graph all the way out to the edges of the window.

2. True or false: if $x_0 < x_1$ and $y_0 < y_1$, then the degree 1 interpolating polynomial is increasing? True or false: if $x_0 < x_1 < x_2$ and $y_0 < y_1 < y_2$, the the degree 2 interpolating polynomial is increasing? (The question of monotone interpolation—how to generate a function which interpolates a given set of points so that the function is increasing if the points are—is the subject of current research papers and Ph.D. theses.)

3. With the default axis range $[-1,1]\times[-1,1]$ in `ginterp`, use the mouse to enter 3 points on the segment of the x-axis connecting $(0,0)$ to $(0.5,0)$. If all three of your points are exactly on the x-axis, what should the interpolating polynomial be? Is it? Is it close? Try the same with 5 points and 10 points.

Quit `ginterp` to continue with the lab.

If `x` is a vector of $n+1$ distinct points, `y` a vector of $n+1$ values, and `t` any number or vector, then the Matlab function `ninterp(x,y,t)` returns the value of the Lagrange interpolating polynomial for the data given in `x`, `y`, evaluated at the point or points given by `t`. For example, to calculate the value $p(.5)$ where $p(x)$ is the polynomial of degree at most 2 passing through $(0,1)$, $(1,2)$, $(3,-2)$, you would use:
`ninterp([0 1 3],[1 2 -2],.5)`.

4. In this question you are to investigate how accurately we can approximate the function $y = \exp(-.4x^2)$ by the Lagrange interpolation polynomial interpolating the function at $n+1$ equally spaced points from -5 to 5, and how the approximation improves as $n \to \infty$.

These commands plot the 3 points $(x, \exp(-.4x^2))$, with $x = -5$, 0, and 5, and then superimpose a plot of the quadratic interpolating polynomial through these points on the interval $[-5,5]$ (using 100 points for the plot).

```
x = linspace(-5,5,3)
y = exp(-.4*x.^2)
plot(x,y,'o')
hold on
t = linspace(-5,5,100);
plot(t,ninterp(x,y,t))
```

Now plot the true function $y = \exp(-.4x^2)$ in red on top: `plot(t,exp(-.4*t.^2),'r')`. Does the interpolating quadratic seem like a reasonable approximation of the true function? How much is the error at the point of largest error? Now plot on top the $4th$ degree interpolating polynomial using 5 equally spaced x values from -5 to 5. Note how the accuracy of approximation changes. Then do the $8th$ degree polynomial using 9 equally spaced points, and the $16th$ degree polynomial using 17 points. How large is the error at the worst point in $[-5, 5]$? At about what value of x in $[-5, 5]$ does the worst error occur? Do you believe that as the number of points is increased, the interpolating polynomial converges to the function everywhere on $[-5, 5]$?

5. Now make an investigation like in the preceding problem, but for the function $1/(1+x^2)$. Answer all the same questions. Food for thought: the two functions look similar. Why is the behavior of their interpolating polynomials so different? (You'll probably need to take more math courses before you can answer this!)

6. The functions `ginterp` and `ninterp` were written by me. Matlab comes with the function `polyfit` which computes the coefficients of the *natural* form of the Lagrange interpolating polynomial by setting up the Vandermonde matrix and solving it. Thus if `x` is a vector of length $n + 1$ with distinct entries and `y` is another vector of length $n + 1$, then `p=polyfit(x,y,n)` defines `p` as a vector of length $n+1$ giving the coefficients of the interpolating polynomial listed from highest power to lowest power.[1] For example, the commands `x = [0 1 3]; y = [1 2 -2]; p = polyfit(x,y,2)` return `p = [-1 2 1]`, because the polynomial $p(x) = -x^2 + 2x + 1$ satisfies $p(0) = 1$, $p(1) = 2$, and $p(3) = -2$. The function `polyval` may then be used to evaluate the polynomial p using the coefficient vector `p`: if `t` is any number or vector, then `polyval(p,t)` is the value of polynomial at `t`.

The use of the natural form of the Lagrange interpolating polynomial is generally **a bad thing**. You shouldn't use `polyfit` except in the simplest cases. For example, let's interpolate the U.S. census data for this century. (The command `load pop` will load the population values for the years 1900, 1910, ... 1990, from the file `pop` into a variable pop in your Matlab workspace.) Compute the interpolating polynomial for this data, and plot it for the years from 1900 to 1990. Do this first using `polyfit`:

```
year = 1900:10:1990
load pop
plot(year,pop,'o')
hold on
t = 1900:1990;
plot(t,polyval(polyfit(year,pop,9),t))
```

Not too good!

Next use `ninterp` to make the same plot and compare the results.

[1] The function `polyfit(x,y,m)` also works if $m < n$, in which case it does a *least squares fit* of a lower degree polynomial to the data. We shall not be concerned with that case here.