Name: _____ Section: _____

ID#: _____

CSE/MATH 455, Arnold Lab 3 October 4, 1996

The Lagrange Interpolating Polynomial

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

1. The Matlab function `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.

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.

4. 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`. (The function `ninterp` uses the Newton form of the interpolating polynomial. It calls `divdif` to compute the necessary divided differences.) For example, these commands plot the 3 points $\left(x, \exp(-.4x^2)\right)$, 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). (Quit `ginterp` before running this.)

```
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))
```

Do the same for $4th$ degree interpolating polynomial using 5 equally spaced x values from -5 to 5, the $8th$ degree polynomial using 9 equally spaced points, and the $16th$ degree polynomial using 17 points.

5. Same problem as above, but for the function $1/(1+x^2)$. The two functions look similar. Do the interpolating polynomials? Why? What is the big difference between $\exp(-.4x^2)$ and $1/(1+x^2)$? (Hint: 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 pretty! 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.