Name: ________________________________ Section: ____________

ID#: ________________________________

CSE/MATH 455, Arnold | Lab 4 | October 18, 1996

Cubic Spline Interpolation

Bring the results for the last two problems to class on Monday.

Preparation: For this lab you need to copy the file `ginterp.m` from the directory `~dna2/pub/interpolation` into your directory. This file replaces the version of `ginterp.m` you used in Lab 3 (this one has splines, the older one didn't). You also need `ninterp.m`, `divdif.m`, and `pop` from the same directory. These are the same as you used in Lab 3.

1. The new version of `ginterp` can be used to graphically display either the Lagrange interpolating polynomial or the cubic spline interpolant (with not-a-knot end conditions) for data you enter. Experiment with `ginterp` to get a feeling for what the interpolating spline looks like.

2. Enter nine points whose x-coordinates about -0.8, -0.6, -0.4, ... , 0.8 and which roughly lie on a parabola. Set the interpolation type to "spline" (with the pop-up menu in the lower right corner), and interpolate. The curve you get should look something like a parabola. Now, without resetting, add a tenth point whose x-coordinate is about 0.1, but which lies a bit off the parabola, and interpolate again. Does the spline change a lot? Are the changes concentrated near $x = 0.1$ or evenly spread throughout the whole interval? Do the same thing for polynomial interpolation and compare. Try some other examples to explore the effect of outlying points on spline and polynomial interpolation.

3. If you use the spline interpolation routine to interpolate two points, what do you get? How does the the resulting function differ from the polynomial interpolant with the same two points? How about three points? Four points? Five points?

4. Recall the problem we had interpolating the function $y = 1/(1 + x^2)$ on the interval $[-5, 5]$ using polynomial interpolation with a lot of points. E.g., (after quitting `ginterp`) run:

```
t = linspace(-5,5,500);
x = linspace(-5,5,9);
y = 1./(1+x.^2);
plot(x,y,'o')
hold on
plot(t,ninterp(x,y,t))
```

which shows an 8th degree interpolating polynomial. Now do the same thing, but use `spline(x,y,t)` instead of `ninterp(x,y,t)`. Try this for more points as well.

5. Compare the polynomial interpolant and the spline interpolant to the U.S. census bureau population data for the years 1900, 1910, ... , 1990 (recall that you can get the census data with `load pop`). Observations?

6. If we interpolate a given function on a given interval by adding points which divide the interval into smaller and smaller subintervals, the spline interpolant becomes more and more accurate. The purpose of this problem is to see how fast. Use the simple function $f(x) = 1/x$ on the interval $[1, 2]$. Evaluate the function at 500 points in the interval (`t=linspace(1,2,500); yexact=1./t;`). Then divide the interval into subintervals of length 1/10 (`x=linspace(1,2,11)`); evaluate the spline interpolant with `yspline=spline(x,1./x,t)`; and find the maximum error on the interval with a command like `max(abs(yexact-yspline))`. Record your result. Now repeat with subintervals of size 1/20, 1/40, 1/80, and 1/160. Compute and record the ratio of successive errors. When you cut the interval size by a factor of two, by about what factor is the error decreased?

7. There is often an advantage to choosing interpolation points which are not evenly spaced. For example, consider the Matlab function `humps`. Plot `humps` on the interval $[0, 3]$, and plot the spline interpolant using 10 evenly spaced points. Now choose 10 points in $[0, 3]$ which are not evenly spaced but which you think will give a better spline interpolant. Use the method of the last problem to find the error. Adjust the 10 points to get the error as small as you can. On Monday bring in your points and your error and we'll see who is the winner in the class.

Can you think of way to write a program to select the points automatically?