CSE/MATH 455, Arnold Lab 9 November 14, 1997

Cubic Spline Interpolation

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

Preparation: This lab uses the files `ginterp.m`, `ninterp.m`, and `pop` from Lab 7. Copy them from `~dna2` if you haven't already.

1. The button labelled "poly." at the bottom of the ginterp window lets you switch between polynomial interpolation, cubic spline interpolation (with not-a-knot end conditions), and piecewise linear interpolation. Using `ginterp`, compare the three kinds of interpolation for various sorts of point layouts.

2. The purpose of this problem is to explore the sensitivity of the interpolating spline or polynomial to perturbations in one of the interpolation points. 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', 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. As another example put a bunch of points on the x-axis and interpolate, and then add one point a bit off the axis and interpolate again (try both polynomial and spline interpolation).

Before moving onto the remaining problem, quit `ginterp`.

3. Recall Runge's example of the noncovergence of Lagrange interpolation with equally spaced points to the bell-shaped curve $y = 1/(1 + x^2)$ on the interval $[-5, 5]$. E.g., 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 the 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. Does spline interpolation to this function seem to converge as more and more equally-spaced points are used?

4. 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?

5. 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 converges to the function. The purpose of this problem is to find the rate of convergence. 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? Can you conjecture what the error bound for spline interpolation looks like?

6. 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 class winner.

Thought problem: How could you design a program to select the points automatically?