Scilab Tutorial 7 for Computational Science

1 Introduction

This tutorial is designed to familiarise you with interpolation.

2 Interpolating Runge's Function

Consider the function

$$f(t) = \frac{1}{1+25t^2}.$$

This function is known as Runge's function.

Define a Scilab function to calculate Runge's function.

```
function y=runge(t)
//
// Runge's Function
// Standard example of polynomial interpolation
// creating oscillations
//
y=(1.0)./(1+25*t.^2)
endfunction
```

Note we have put parentheses around the `(1.0)` to ensure that the `./` command is used. An expression of the form `1./(..)` would actually use the `/` matrix operator!

Consider a polynomial

$$p_n(t) = \sum_{j=1}^{n} x_j t^{j-1}$$

which interpolates (coincides with) Runge's function at n evenly spaced points between -1 and 1 (i.e. at the points `-1:2/(n-1):1`|. We have n unknowns x_i for the coefficients of the polynomial and n data points. This will provide us with a linear equation for the polynomial coefficients given the values of the data points.

Create a function called `polyfit`, with calling sequence `pp=polyfit(tdata, ydata)`, that fits a polynomial of degree $n-1$ to n data points `t`, `y` and returns a polynomial `pp`. You can easily add an extra argument to calculate the least square fit of an $m < n$ degree polynomial to the n data points. The result of `polyfit` should then be able to be evaluated using the command

```
tt = -1:0.01:1;
yy = horner(pp,tt)
```

Hint: You can use the standard backslash operator to solve for the coefficients once you have the Vandermonde matrix. The `feval` command together with an appropriately defined function

```
function z=vandermonde(t,q)
z=t.^q;
endfunction
```

may be of help in producing the associated Vandermonde matrix.

You might like to use `deff` to define your function

```
deff('z=vandermonde(t,q)','z=t.^q')
```

This is useful for short functions.

Use your `polyfit` function to calculate the coefficients of the degree 5 and degree 10 polynomials which interpolate Runge's function at 6 and 11 evenly spaced points in the interval $[-1, 1]$.

How successful are the polynomial approximations about zero and the end points?

3 Taylor Polynomials

Note that the expression $1/(1 + t) = 1 - t + t^2 - t^3...$ easily gives the Taylor polynomials centred about the origin of the Runge function to have the form

$$1 - 25t^2 + (25t^2)^2 - (25t^2)^3...$$

Use this to plot the Taylor polynomial of degree four centred at zero for the Runge function, over the interval [-1,1]. You should also plot the Runge function and the Taylor polynomial over a smaller interval, say [-0.2,0.2]. Try to explain its accuracy, and what you would expect the accuracy of the degree eight Taylor polynomial to be like.

4 Chebyshev Interpolation

Let us compute and graph polynomial approximations to the Runge function of degree four and eight using interpolation at the Chebyshev points: for the case of degree four, the SCILAB expression to compute the five points needed is

```
tdata = cos((1:2:(2*5-1))*%pi/(2*5))
ydata = runge(tdata)
chebp = polyfit(tdata,ydata)
tt    = -1:.01:1;
rr    = runge(tt);
yy    = horner(chebp,tt);
xbasc();
plot(tt,yy,style=1);
plot2d(tt,rr,style=2);
plot2d(tdata,ydata,style=-1)
```

The results are more accurate than with equally spaced points or Taylor polynomials but still not very good: try to explain these observations.

Try higher order Chebyshev interpolants.

5 Spline Interpolation

The following SCILAB commands use the command `splin` to compute the natural spline interpolate through the points specified by the vectors `x` and `y`. You use `interp` to evaluate the spline at the points `xx`.

```
tdata = -1:.5:1;
ydata = runge(xs);
ddata = splin(tdata,ydata);
tt    = -1:.01:1;
rr    = runge(tt);
ss    = interp(tt,tdata,ydata,ddata);
```

Thus the spline approximation can then be graphed using

```
xbasc();
plot2d(tt,ss,style=1)
plot2d(tt,rr,style=2);
plot2d(tdata,ydata,style=-1);
```

Change the code to plot the cubic spline approximations of the Runge function using a set of nine equally spaced points, interpolating the Runge function.

6 Monotonic Cubic Interpolation

From the previous example we see that cubic spline interpolation can still introduce unwanted oscillations. There are many techniques, for instance a fairly common method is that of Fritsch and Carlson (SIAM J. Numer. Anal. 17,

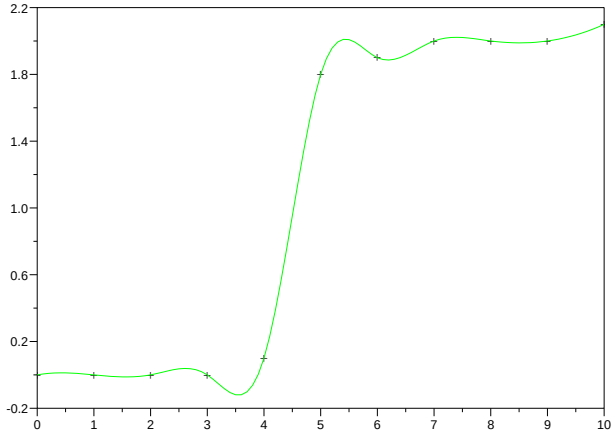
pp 238-246 1980). SCILAB implements a monotonic interpolation method as an option to the `splin` function.

Here is some data

```
tdata = 0:1:10;
ydata = [0 0 0 0 0.1 1.8 1.9 2.0 2.0 2.0 2.1];
```

Lab Book: Fit an ordinary spline function to this data set. Produce a plot

I obtained the following:



The result is quite oscillatory, maybe not really what we want.

Now re do the spline interpolation of this data, but use the extra argument `'monotone'`. I.e. use

```
ddata = splin(tdata,ydata, 'monotone');
```

Now you can use `tdata` and `ydata` values together with the new derivative values to produce an interpolant using the `interp` function.

Lab Book: Plot the data points, together with plots of the ordinary spline and monocubic interpolant. Comment on the respective interpolants.

Hopefully you can see that in some cases it is useful to try to maintain the shape implicit in your data sets. This is particularly true for coarse data sets.