MA2507 Computing Mathematics Laboratory: Week 11

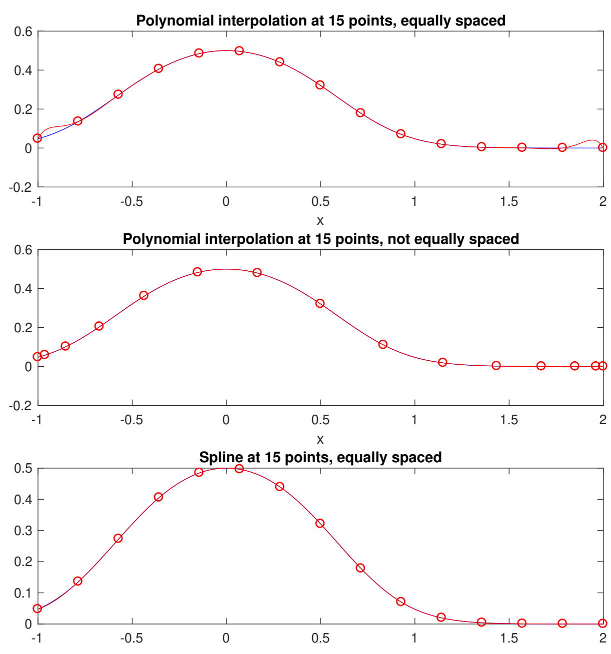
1. Polynomial interpolation: Given $n+1$ points (x_i, y_i) for $0 \le i \le n$, there is a unique polynomial $P(x)$ with a degree $\le n$, such that $P(x_i) = y_i$ for $0 \le i \le n$. This is the so-called polynomial interpolation. In MATLAB, this can be calculated by `polyfit`. The output of `polyfit` is the polynomial coefficients. We can use `polyval` to evaluate the polynomial. Let us try on

$$f(x) = \frac{1}{1+e^{3x^2}}$$

for $-1 \le x \le 2$.

```
f =@(x) 1./(1+exp(3*x.^2));
xx = linspace(-1,2,200);   % 200 points for drawing figures
n = 14
x = linspace(-1,2,n+1);
c = polyfit(x,f(x),n);
yy = polyval(c,xx);
subplot(3,1,1)
plot(xx,f(xx),'b',x,f(x),'ro', xx,yy,'r')
xlabel('x'), title('Polynomial interpolation at 15 points, equally spaced')
```

The above produces the top plot in the following figure. In the above, we take $n+1 = 15$ equally



spaced points from function $f(x)$, then use MATLAB commands `polyfit` and `polyval` to find the polynomial interpolant. The command `polyfit` requires the $n+1$ points given as the first and second inputs, but there is a thrid input for the polynomial degree. You can use a number less than n, it will try to "least squares". We see that the polynomial does not fit the original function well near the two end points. This is the so-called Runge phenomenon.

There is a way to improve this. The idea is to choose more points near the two end points. This can be done by first choosing $n+1$ points in $[-1,1]$ by

$$t_j = \cos\left(\frac{j\pi}{n}\right), \quad j = 0, 1, ..., n,$$

then transform t_j to x_j by $x_j = 0.5 - 1.5t_j$. These points t_j for $0 \le j \le n$ are the so-called Chebyshev points. Following the above MATLAB program, we add the following lines

```
t = cos((0:n)*pi/n);
x = 0.5-1.5*t;
c = polyfit(x,f(x),n;
yy = polyval(c,xx);
subplot(3,1,2)
plot(xx,f(xx),'b',x,f(x),'ro', xx,yy,'r')
xlabel('x'), title('Polynomial interpolation at 15 points, not equally spaced')
```

This leads to the second plot in the figure above.

2. Spline: Given $n+1$ points, we can use the command `spline` to produce a nice smooth curve. It produce a polynomial of degree 3 (or less) on each interval (x_i, x_{i+1}), and also the final function is smooth with continuous second order derivative. Now, let us try `spline` on 15 equally spaced points. We add the following lines to the above MATLAB program.

```
x = linspace(-1,2,n+1);
yy = spline(x,f(x),xx)
subplot(3,1,3)
plot(xx,f(xx),'b',x,f(x),'ro', xx,yy,'r')
xlabel('x'), title('Spline at 15 points, equally spaced')
```

This produces the third plot in the above figure. In the following, we consider n points (x_j, y_j) for $j = 1, 2, .., n$, assume $x_1 < x_2 < ... < x_n$, and denote the spline function as $S(x)$. We have $S(x_j) = y_j$ for $j = 1, 2, ..., n$, and S, S' and S'' are continuous. In addition, there are three choices

(a) $S''(x_1) = S''(x_n) = 0$.

(b) S' is given at x_1 and x_n.

(c) S''' is continuous at x_2 and x_{n-1}.

MATLAB `spline` gives (c) or (b) with extra input for S'. Here, we give some details about (a).

First, we define

$$h_j = x_{j+1} - x_j, \quad d_j = 6\frac{y_{j+1} - y_j}{x_{j+1} - x_j}, \quad j = 1, 2, ..., n-1.$$

Now, let $z_j = S''(x_j)$ for $j = 1, 2, ..., n$. For choice (a), we set $z_1 = z_n = 0$ and solve z_2, z_3, ..., z_{n-1} from

$$\begin{bmatrix} 2(h_1+h_2) & h_2 & & \\ h_2 & 2(h_2+h_3) & \ddots & \\ & \ddots & \ddots & h_{n-2} \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} z_2 \\ z_3 \\ \vdots \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} d_2 - d_1 \\ d_3 - d_2 \\ \vdots \\ d_{n-1} - d_{n-2} \end{bmatrix}.$$

On (x_j, x_{j+1}), $S(x)$ is a cubic polynomial given as

$$S(x) = c_{1j}(x_{j+1} - x)^3 + c_{2j}(x - x_j)^3 + c_{3j}(x_{j+1} - x) + c_{4j}(x - x_j),$$

where

$$c_{1j} = \frac{z_j}{6h_j}, \quad c_{2j} = \frac{z_{j+1}}{6h_j}, \quad c_{3j} = \frac{y_j}{h_j} - \frac{z_j h_j}{6}, \quad c_{3j} = \frac{y_{j+1}}{h_j} - \frac{z_{j+1} h_j}{6}.$$

Here is the MATLAB program.

```
function yy = nspline(x,y,xx)
% Spline with 2nd order derivatoive = 0 at endpoints.
% Part 1: calculate the spline function S(x)
n = length(x);
for j=1:n-1
    h(j) = x(j+1)-x(j);
    d(j) = 6*(y(j+1)-y(j))/h(j);
end
A = zeros(n-2,n-2);
b = zeros(n-2,1);
A(1,1)=2*(h(1)+h(2));
b(1) = d(2)-d(1);
for j=2:n-2
    A(j,j)=2*(h(j)+h(j+1));
    A(j-1,j) = h(j);
    A(j,j-1) = h(j);
    b(j) = d(j+1)-d(j);
end
z = A\b;
z = [0; z; 0];
for j = 1:n-1
    c1(j)=z(j)/(6*h(j));
    c2(j)=z(j+1)/(6*h(j));
    c3(j)= y(j)/h(j)-z(j)*h(j)/6;
    c4(j)= y(j+1)/h(j)-z(j+1)*h(j)/6;
end
% Part 2: evaluate function S on vector xx
k = 1;
for j= 1: length(xx)
    if xx(j) > x(k+1)
        k = k+1;
    end
    t = x(k+1)-xx(j);
    s = xx(j)-x(k);
    yy(j) = c1(k)*t^3+c2(k)*s^3+c3(k)*t+c4(k)*s;
end
end
```

3. Graphical input. MATLAB allows you to input points by clicking the mouse using `ginput`, but

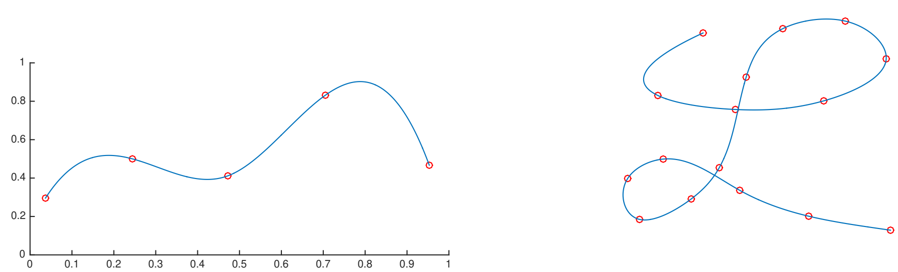
it does not show the points in the graphical window. I wrote the following program that draws a little circle for each point you input.

```
function [x,y]=myginp
% axis([0 1 0 1]); hold on
% hit Return to stop
j=0;
click=1;
while click > 0
    [xin,yin] = ginput(1);
    click = length(xin);
    if click == 1
        plot(xin,yin,'ro');
        j=j+1;
        x(j) = xin;
        y(j) = yin;
    end
end
```

You can use `myginp` with `spline` to draw a smooth function connecting a few points that you input by clicking the mouse.

```
axis([0 1 0 1])
hold on
[x,y]=myginp
xx = linspace(min(x),max(x),400);
yy = spline(x,y,xx);
plot(xx,yy)
hold off
```

The above program is used to draw the left plot in the following figure. We can also use `myginp`



to input a few points (x_i, y_i) for $1 \le i \le n$, and use `spline` to draw a smooth curve (not y as a function of x) connecting these points. We need to define t_i for $1 \le i \le n$, such that $t_1 = 0$, and $t_{i+1} = t_i + \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$, and draw spline functions of t. We have the following program:

```
axis([0 1 0 1])
hold on
```

```
[x,y]=myginp;
n=length(x);
t(1)=0;
for j=2:n
    t(j)=t(j-1)+sqrt((x(j)-x(j-1))^2+(y(j)-y(j-1))^2);
end
tt=linspace(0,t(n),500);
xx=spline(t,x,tt);
yy=spline(t,y,tt);
plot(xx,yy)
axis off
hold off
```

It is used to produce the right plot in the figure above.

It is also useful to use periodic boundary conditions. Namely, we want $S(x_1) = S(x_n)$, $S'(x_1) = S'(x_n)$ and $S''(x_1) = S''(x_n)$. That means, for the input, we must insist $y_n = y_1$. Now, we solve z_1, z_2, ..., z_{n-1} from

$$\begin{bmatrix} 2(h_{n-1}+h_1) & h_1 & 0 & \dots & h_{n-1} \\ h_1 & 2(h_1+h_2) & h_2 & & 0 \\ 0 & h_2 & 2(h_2+h_3) & \ddots & \\ \vdots & & \ddots & \ddots & h_{n-2} \\ h_{n-1} & 0 & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} d_1 - d_{n-1} \\ d_2 - d_1 \\ d_3 - d_2 \\ \vdots \\ d_{n-1} - d_{n-2} \end{bmatrix}.$$

After that we set $z_n = z_1$. The formula for S is the same.