Assignment 2

Due. In class Oct. 5 (Friday)

1. The following figures from the Census Bureau give the population of the United States:

Year	Population
1900	75,994,575
1910	91,972,266
1920	105,710,620
1930	122,775,046
1940	131,669,275
1950	150,697,361
1960	179,323,175
1970	203,235,298

   - Since there are eight points, there is a unique polynomial of degree 7 which interpolates the data. However, some of the ways of representing this polynomial are computationally more satisfactory than others. Here are four possibilities, each with t ranging over the interval $1900 \le t \le 1970$:

$$\sum_{j=0}^{7} a_j t^j,$$
$$\sum_{j=0}^{7} b_j (t-1900)^j,$$
$$\sum_{j=0}^{7} c_j (t-1935)^j,$$
$$\sum_{j=0}^{7} d_j \left(\frac{t-1935}{35}\right)^j.$$

     In each case, the coefficients are found by solving an 8-by-8 Vandermond system, but the matrices of various systems are quite different. Set up each of the four matrices, and find the estimate of its condition using Matlab function `cond()`. Then use Matlab operator "\" to find the coefficients. Check each of the representations to see how well it reproduces the original data.

   - Interpolate the data by a 7th-degree polynomial, using the best conditioned representation found above, and by the natural cubic spline using `ncspline.m`. Graph the resulting functions at one-year intervals over the period from 1900 to 1980. Find the 1980 census data. Which approach gives more accurate prediction?

Solution The condition numbers:

	model a	model b	model c	model d
cond	1.239e+32	1.785e+13	7.891e+10	5.354e+2

Relative errors of the reproduced data by evaluating the polynomials using the Horner's rule:

	model a	model b	model c	model d
relative error	4.0−3	4.7e−14	2.3e−16	3.3e−16

Predictions for 1980:

	model d	spline	real
prediction	402.33 million	227.15 million	226.44 million

2. In the natural spline, we set $\sigma_1 = \sigma_n = 0$. The following is an alternative way of setting σ_1 and σ_n.

   Let $c_1(x)$ and $c_n(x)$ be the unique cubics which pass through the first four and the last four data points, respectively. The two end conditions match the third derivatives of $s(x)$ to the third derivatives of these cubics, namely

   $$s'''(x_1) = c_1''' \quad \text{and} \quad s'''(x_n) = c_n'''.$$

   The constants c_1''' and c_n''' can be determined directly from the data without actually finding $c_1(x)$ and $c_n(x)$. We have already introduced the quantities

   $$\Delta_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i},$$

   which are approximations to the first derivatives. Let

   $$\Delta_i^{(2)} = \frac{\Delta_{i+1} - \Delta_i}{x_{i+2} - x_i} \quad \text{and} \quad \Delta_i^{(3)} = \frac{\Delta_{i+1}^{(2)} - \Delta_i^{(2)}}{x_{i+3} - x_i}.$$

   These quantities are known as divided differences; $2\Delta_i^{(2)}$ and $6\Delta_i^{(3)}$ are approximations to the second and third derivatives. In particular,

   $$c_1''' = 6\Delta_1^{(3)} \quad \text{and} \quad c_n''' = 6\Delta_{n-3}^{(3)}.$$

   Consequently, we require that

   $$\frac{\sigma_2 - \sigma_1}{h_1} = \Delta_1^{(3)} \quad \text{and} \quad \frac{\sigma_n - \sigma_{n-1}}{h_{n-1}} = \Delta_{n-3}^{(3)}.$$

   To make the final system of equations symmetric, these last two equations should be multiplied by h_1^2 and $-h_{n-1}^2$ to give

   $$-h_1\sigma_1 + h_1\sigma_2 = h_1^2\Delta_1^{(3)}$$

   and

   $$h_{n-1}\sigma_{n-1} - h_{n-1}\sigma_n = -h_{n-1}^2\Delta_{n-3}^{(3)}.$$

   Modify the natural cubic spline function `ncspline.m` using the above two equations for the end conditions.

   Compare the natural spline and the spline with the end conditions we have described. Specifically, generate test data by taking x_i, $i = 1, ..., n$, to be n equally spaced points in the interval $[0, \pi]$, including the end points, and $y_i = \cos(x_i)$. Obtain the spline coefficients from

`ncspline.m` without any modifications. Also, obtain the coefficients from the spline, called `cspline.m`, modified above. Compare the two different functions $s(x)$ with $\cos(x)$ for values of x other than the data points. Pay particular attention to values near the ends of the interval. Use several values of n, say, 10, 20, and 30. Why should $\cos(x)$ be used instead of $\sin(x)$ in this experiment?

Solution For the cubic spline program, see `cspline.m`. The script file for experiments:

```
n = input('n = ')
% n interpolating points
x = linspace(0.0, pi, n);
y = cos(x);

% call cubic spline
[b,c,d] = cspline(x,y);

% call natural cubic spline
[nb,nc,nd] = ncspline(x,y);

% 10 points on [0.0, 0.1]
z1 = linspace(0.0, 0.1, 10);
% 10 points on [pi-0.1, pi]
z2 = linspace(pi-0.1, pi, 10);

% compare cubic spline and cos
v1 = seval(z1, x, y, b, c, d); % near 0
v1 - cos(z1)
v2 = seval(z2, x, y, b, c, d); % near pi
v2 - cos(z2)

% compare natural cubic spline and cos
nv1 = seval(z1, x, y, nb, nc, nd); % near 0
nv1 - cos(z1)
nv2 = seval(z2, x, y, nb, nc, nd); % near pi
nv2 - cos(z2)
```

When $n = 10$ the error for the cubic spline near 0 and π is about 10^{-4}; the error for the natural cubic spline is about 10^{-3}. When $n = 20$ the error for the cubic spline near 0 and π is about 10^{-5}; the error for the natural cubic spline is about 10^{-3}. When $n = 30$ the error for the cubic spline near 0 and π is about 10^{-6}; the error for the natural cubic spline is about 10^{-4}. Near 0 and π, $\sin(x)$ is almost linear, since $\sin''(0) = \sin''(\pi) = 0$, which are the end conditions of the natural cubic spline, thus the natural cubic spline would work better than the above cubic spline for $\sin(x)$. Whereas, $\cos(x)$ is nonlinear near 0 and π.

3. The speed of sound in ocean water depends on pressure, temperature, and salinity, all of which vary with depth in fairly complicated ways. Let z denote depth in feet under the ocean surface (so that the positive z axis points down) and let $c(z)$ denote the speed of sound at depth z. We shall ignore the changes in sound speed observed in horizontal directions. It is possible to measure $c(z)$ at discrete values of z, and the following table is typical of those obtained:

z(ft)	$c(z)$(ft/sec)
0	5,042
500	4,995
1,000	4,948
1,500	4,887
2,000	4,868
2,500	4,863
3,000	4,865
3,500	4,869
4,000	4,875
5,000	4,875
6,000	4,887
7,000	4,905
8,000	4,918
9,000	4,933
10,000	4,949
11,000	4,973
12,000	4,991

To obtain values of $c(z)$ between data points, we can use cubic spline interpolation. We shall also need to evaluate $c'(z) = dc(z)/dz$, so the first step of this problem is the following:

(a) Use `ncspline` to obtain the coefficients of the cubic spline interpolating the above data. Modify `seval` so that it returns both the value of the spline and the value of its derivative at any point.

Solution The modified `seval`:

```
function [v, dv] = seval(z,x,y,b,c,d)
% [v, dv] = seval(z,x,y,b,c,d)
%
% Evaluate natural cubic spline at z
% and derivatives at z
%
% input:
%       z       points at which the spline is evaluated
%       x,y     coordinates interpolated
%       b,c,d   coefficients returned by ncspline.m
% output:
%       v       values of the spline at z
%       dv      derivatives at z
% dependency:
%       locate.m subinterval locator

n = length(z);
%
g = 1;          % initial guess of the location of z(1)
for j=1:n
    i = locate(x,z(j),g);       % locate z(j)
```

```
%   evaluate at z(j) in [x(i), x(i+1)]
    tmp = z(j) - x(i);
    v(j) = y(i)+tmp*(b(i)+tmp*(c(i)+tmp*d(i)));
%   derivative at z(j)
    dv(j) = b(i) + tmp*(2*c(i) + tmp*3*d(i));
    g = i;      % guess for the next location
end;
```

4. A common problem of applied mathematics is that of solving the integral equation

$$f(x) + \int_a^b K(x,t)y(t)dt = y(x),$$

where the functions $f(x)$ and $K(x,t)$ are given and the problem is to compute $y(x)$.

If we approximate the integral by the quadrature formula

$$\int_a^b K(x,t)y(t)dt \approx \sum_{i=1}^{n} \alpha_i K(x,t_i)y(t_i),$$

then the integral equation becomes a system of linear algebraic equations:

$$f(t_j) + \sum_{i=1}^{n} \alpha_i K(t_j,t_i)y(t_i) = y(t_j), \quad j = 1,2,...,n.$$

The solution $y(t_i)$, $i = 1,...,n$, is the desired discretized approximation to the function $y(t)$.

Using Simpson's rule, find an approximate solution of the integral equation

$$\frac{4x^3+5x^2-2x+5}{8(x+1)^2} + \int_0^1 \left(\frac{1}{1+t} - x\right) y(t)dt = y(x).$$

Write a MATLAB program to solve the above equation. The main program (script file) calls a general integral equation solver

```
y = inteqn(t, kernel, fun, coef)
```

where `t` is the vector of partition points, `kernel` is the kernel function K, `fun` is the function f, and `coef` is the vector of coefficients in the quadrature rule (Simpson's rule in this case). Your program should be general enough to take any coefficient vector $[\alpha_1,...,\alpha_n]^{\mathrm{T}}$. So, you should first generate the coefficient vector for the Simpson's rule before calling `inteqn`.

Fit a spline function to the discretized approximation of the function $y(t)$. Compare the resulting spline function with the true solution

$$y(x) = (1+x)^{-2}$$

at various points in the interval $[0,\ 1]$.

Solution The matrix form of the equation

$$f(t_j) + \sum_{i=1}^{n} \alpha_i K(t_j,t_i)y(t_i) = y(t_j), \quad j = 1,2,...,n.$$

is

$$f + Ky = y$$

where $f = [f(t_1), ..., f(t_n)]^{\mathrm{T}}$, $y = [y(t_1), ..., y(t_n)]^{\mathrm{T}}$, and

$$K = \begin{bmatrix} \alpha_1 K(t_1, t_1) & \alpha_2 K(t_1, t_2) & \cdots & \alpha_n K(t_1, t_n) \\ \alpha_1 K(t_1, t_1) & \alpha_2 K(t_1, t_2) & \cdots & \alpha_n K(t_1, t_n) \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1 K(t_1, t_1) & \alpha_2 K(t_1, t_2) & \cdots & \alpha_n K(t_1, t_n) \end{bmatrix}.$$

Thus y is the solution of

$$(I - K)y = f.$$

```
k = input('Enter number of pannels: ');
x = linspace(0,1,k+1);  x = x';   % evenly spaced knots
% Note: this x can be replaced by any partition of [0,1]

y = zeros(length(x),1);         % discrete approximation at x

% use Simpson's
n = 2*k + 1;                    % number of points in Simpson's rule
coef = zeros(n,1);              % coeficients in Simpson't rule
t = zeros(n,1);                 % points in Simpson's rule
% generate Simpson's rule coefficients and evaluation points
for i=1:k
    coef(2*i-1:2*i+1) = coef(2*i-1:2*i+1) + [1; 4; 1];
    t(2*i-1) = x(i);
    t(2*i) = (x(i) + x(i+1))/2;
end
t(n) = x(k+1);
coef = coef/(6*k);

% solve the integral equation
yt = inteqn(t, @knl, @fnc, coef);
% discrete approximation to y(x) at the partition x
y = yt(1:2:n);

% check results
yexact = 1./((1+x).*(1+x));
norm(y - yexact),

%%%%%%%%% subfunctions %%%%%%%%%

%%%%%% kernel function %%%%%%
function v = knl(x,t)

v = 1/(1+t) - x;

%%%%%% f function %%%%%%
```

```
function f = fnc(x)

f = (4*x.*x.*x + 5*x.*x - 2*x + 5)./(8*(x+1).*(x+1));

%%%%%% integral equation solver %%%%%%
function y = inteqn(t, kernel, fun, coef)
%
% Inputs
%    t          evaluation points of the quadrature rule
%    kernel     kernel function K
%    fun        function f
%    coef       quadrature rule coefficients
% Output
%    y          discrete solution values at t

n = length(t);
f = feval(fun, t);
%
for j=1:n
    for i = 1:n
        K(j,i) = feval(kernel, t(j), t(i));
    end
end

% A = eye(n) - K*diag(coef);
for j=1:n
    A(:,j) = -coef(j)*K(:,j);
    A(j,j) = 1.0 + A(j,j);
end
y = A\f;
```