Homework – Splines Due Tues. 4/16/18

1. (25pts)a. The natural spline code takes x and y values, which are computed before invoking. The code is identical to the class spline code `cubic_splinenat` with a few lines changed at the top for the x input and function, then the graphing lines 11-14 and 72-79 were removed. The x input is `x = [0:4]`, while the y input evaluates

$$f(x) = 5e^{0.2x}, \qquad x \in [0, 4],$$

using `y = fcn_hw7(x)`. The function satisfies:

```
function y = fcn_hw7(x)
% Function definition for hw 7
y = 5*exp(0.2*x);
end
```

The best cubic spline $S_3(x)$ for $x \in [2, 3]$ is given by

$$\begin{aligned} S_3(x) &= s_1(x-2)^3 + s_2(x-2)^2 + s_3(x-2) + s_4 \\ &= 0.039025457(x-2)^3 + 0.125722978(x-2)^2 + 1.486722078(x-2) + 7.459123488. \end{aligned}$$

Using these spline values to find the integral, compared with the actual integral

$$\begin{aligned} \int_2^3 S_3(x)\,dx &= \left(s_1\frac{(x-2)^4}{4} + s_2\frac{(x-2)^3}{3} + s_3\frac{(x-2)^2}{2} + s_4 \right)\Big|_2^3 \\ &= \frac{s_1}{4} + \frac{s_2}{3} + \frac{s_3}{2} + s_4 \\ &= 8.254148551 \qquad \text{and} \\ \int_2^3 f(x)\,dx &= 8.257352569. \end{aligned}$$

Significantly, we see that the integration depends only on the coefficients of the cubic and the step size of the subintervals. The absolute error is

$$\left| \int_2^3 S_3(x)\,dx - \int_2^3 f(x)\,dx \right| = 0.003204018.$$

b. The spline quadrature program is designed for any function, f, interval $[a, b]$, and number of divisions N, and it is given by:

```
function sum = spline_quad(f,a,b,N)
%Quadrature using splines

x = linspace(a,b,N+1);
y = f(x);

```

```
P = cubic_splinenat(x,y);

h = (b-a)/N;
sum = 0;
for i = 1:N
    sum = sum + P(i,1)*h^4/4 + P(i,2)*h^3/3 + P(i,3)*h^2/2 + P(i,4)*h;
end
end
```

By invoking this code in MatLab with `spline_quad(@fcn_hw7,0,4,4)`, we obtain the approximation:

$$\int_0^4 S_3(x)\,dx \approx 30.65376468.$$

The exact integral is:

$$\int_0^4 5e^{0.2x}\,dx = 25(e^{0.8}-1) = 30.63852321.$$

The absolute error is 0.015241471.

c. Along with the spline quadrature we use our Composite Trapezoid Rule:

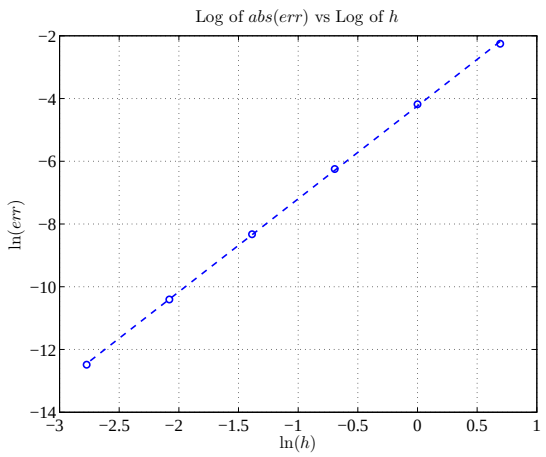
```
function T = comptrap(f,a,b,N)
% Composite Trapezoid Rule for function f(x)
% on [a,b] using N steps
h = (b-a)/N;
i = 0:N-1;
xi = a+i*h;
xi1 = a+(i+1)*h;
T = (h/2)*sum(f(xi)+f(xi1));
end
```

The quadrature programs are invoked with the following script that also calculates the order of convergence numerically and creates graphs for the convergence. This script doubles the value of N 5 times, computing the error against the actual integral.

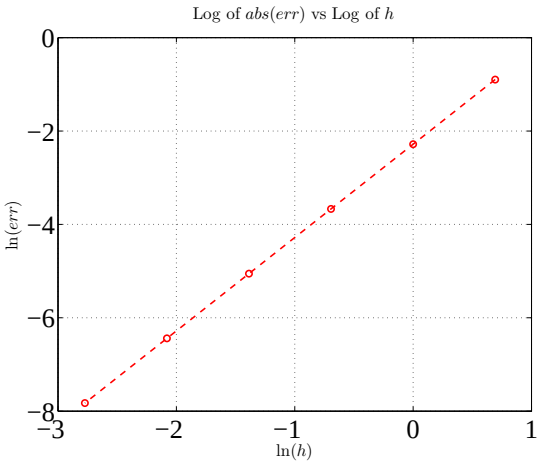
```
% Need to compare spline quadrature to composite trapezoid
% Use our function and double number of steps for N = 2 and double 5 times
a = 0; b = 4; N = 2;
act = 25*(exp(0.8)-1);
lnh = []; lnS = []; lnT = []; ES = []; ET = [];
for i = 1:6
    S = spline_quad(@fcn_hw7,a,b,N);
    ES = [ES,abs(S-act)];
    T = comptrap(@fcn_hw7,a,b,N);
    ET = [ET,abs(T-act)];
    lnS = [lnS,log(abs(S-act))];
    lnT = [lnT,log(abs(T-act))];
    lnh = [lnh,log((b-a)/N)];
    N = 2*N;
end
coS = polyfit(lnh,lnS,1)
coT = polyfit(lnh,lnT,1)
```

```
figure(2)
plot(lnh,lnS,'bo',lnh,coS(1)*lnh+coS(2),'b--');grid
% Set up fonts and labels for the Graph
fontlabs = 'Times New Roman';
xlabel('$\ln(h)$','FontSize',16,'FontName',fontlabs, ...
    'interpreter','latex');
ylabel('$\ln(err)$','FontSize',16,'FontName',fontlabs, ...
    'interpreter','latex');
mytitle = 'Log of $abs(err)$ vs Log of $h$';
title(mytitle,'FontSize',16,'FontName', ...
   'Times New Roman','interpreter','latex');
set(gca,'FontSize',16);
print -depsc hw7_lnerrA.eps
figure(3)
plot(lnh,lnT,'ro',lnh,coT(1)*lnh+coT(2),'r--');grid
xlabel('$\ln(h)$','FontSize',16,'FontName',fontlabs, ...
    'interpreter','latex');
ylabel('$\ln(err)$','FontSize',16,'FontName',fontlabs, ...
    'interpreter','latex');
mytitle = 'Log of $abs(err)$ vs Log of $h$';
title(mytitle,'FontSize',16,'FontName', ...
   'Times New Roman','interpreter','latex');
print -depsc hw7_lnerrB.eps
```

h	Spline Quad $\ln\lvert error\rvert$	CTR $\ln\lvert error\rvert$
2	0.105064	0.407428
1	0.015241	0.102060
1/2	0.0019339	0.025528
1/4	2.4217×10^{-4}	0.0063828
1/8	3.0292×10^{-5}	0.0015957
1/16	3.7877×10^{-6}	3.9894×10^{-4}



Spline Quadrature Error



Composite Trapezoid Error

From the MatLab program `polyfit`, we find the the best fitting line for the Cubic Spline quadrature is

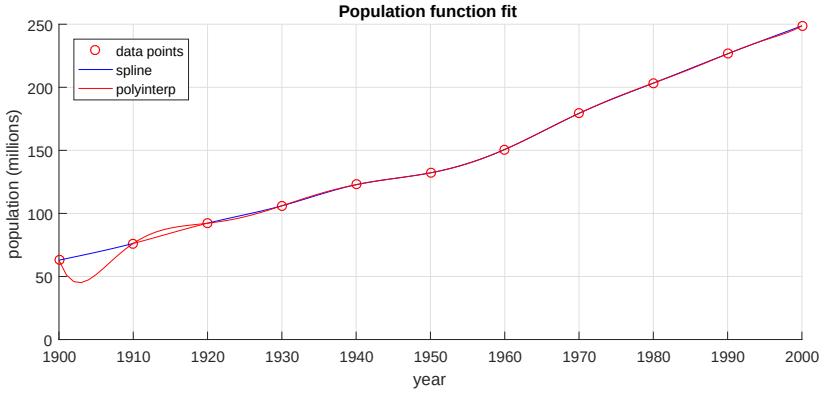
$$\ln|error| = 2.9634\ln(h) - 4.2354,$$

while for the Composite Trapezoid Rule, we obtain

$$\ln|error| = 1.9994\ln(h) - 2.2828.$$

The leading coefficients show numerically that the Cubic Spline quadrature technique has an order of convergence $\mathcal{O}\left(h^3\right)$, while the CTR has the expected convergence of $\mathcal{O}\left(h^2\right)$. However, a clock shows that the Spline method takes about 4 times longer, which would show that using the Composite Simpson's Rule would be much more efficient. The increase in order of convergence would suggest that the Spline method would be better for needs of higher accuracy.

2. (20pts) a. The programs from class `polyinterp.m` and `cubic_splinenat.m` use the census data and produce the following graph.



The polynomial and spline graphs are very similar for later years, while the polynomial fit works poorly for the first years. We have already shown that fitting a high order polynomial often oscillates more at the endpoints. The cubic spline is splitting the data into small smooth segments.

b. We define the vector of years `yr = [1900 1910 ... 2000]` and the vector of the population `pop = [62.948 76.212 ... 248.710]`. We use the MatLab program for creating vandermonde matrices, `V = vander(yr)`, which creates a very ill-conditioned matrix. Checking the 1-norm condition number, we obtain `cond(V,1)= 1.8682675e+48`. We used the `format longe` to obtain the coefficients of the interpolating polynomial with `c = V\pop'` and

obtained:

$$\begin{aligned}
c_1 &= -1.950005757027322e-18 \\
c_2 &= 2.950569627239955e-14 \\
c_3 &= -1.931690520362793e-10 \\
c_4 &= 7.095699514296991e-07 \\
c_5 &= -1.575499180270782e-03 \\
c_6 &= 2.083161299913479e+00 \\
c_7 &= -1.390802494656150e+03 \\
c_8 &= 9.070471111490215e+03 \\
c_9 &= 6.214881025066986e+08 \\
c_{10} &= -3.255033064388682e+11 \\
c_{11} &= 3.232923056984116e+13
\end{aligned}$$

Because of the ill-conditioning, results of these coefficients may vary by a fair amount. This wide variation suggests that the results are quite unreliable. In contrast, the coefficients of all the cubic splines on each interval vary substantially less. If the cubic spline on each interval is given by:

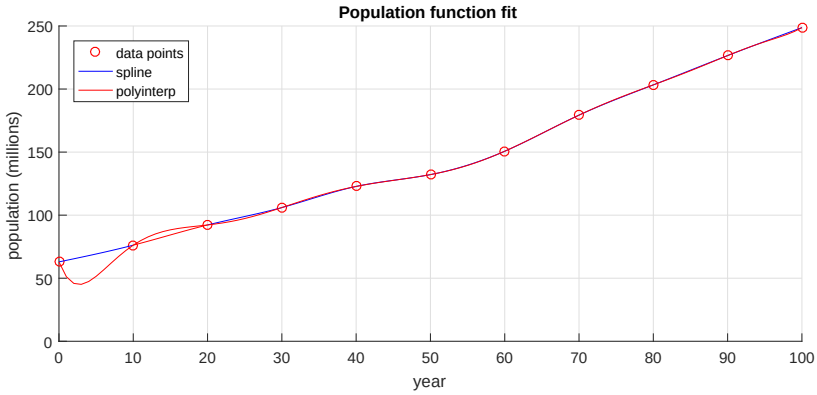
$$S_i(x) = s_3(x - x_i)^3 + s_2(x - x_i)^2 + s_1(x - x_i) + s_0,$$

then finding the maximum and minimum coefficients in absolute value from our spline program gives:

$$\begin{aligned}
0.000028407 &\leq |s_3| \leq 0.0052986, \\
6.6613 \times 10^{-17} &\leq |s_2| \leq 0.086612, \\
1.13259 &\leq |s_1| \leq 2.7309, \\
62.948 &\leq |s_0| \leq 226.546,
\end{aligned}$$

Thus, we expect the spline program to be significantly more stable.

c. The graph looks similar for the shifted time scale as seen below.



We define the vector of years `yr = [0 10 ... 100]`, and the vector of the population `pop ... = [62.948 76.212 ... 248.710]` is the same. We use the MatLab program for creating

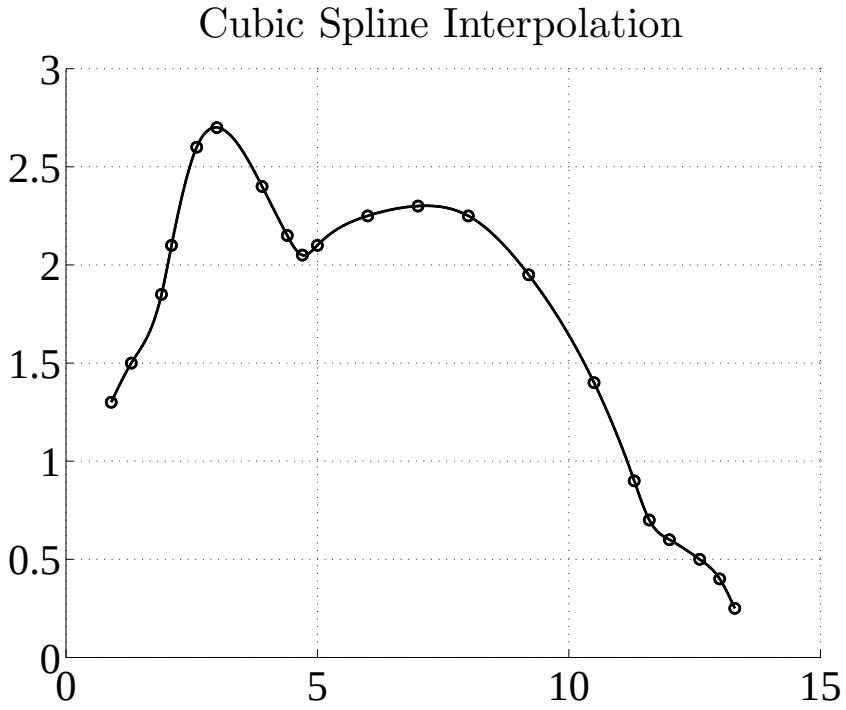
vandermonde matrices, `V = vander(yr)`, which creates an ill-conditioned matrix. Checking the 1-norm condition number, we obtain `cond(V,1)= 1.01634e+21`, which is substantially improved from Part b. We used the `format longe` to obtain the coefficients of the interpolating polynomial with `c = V\pop'` and obtained:

$$\begin{aligned}
c_1 &= 1.28160273369648E-14 \\
c_2 &= -6.90905533515114E-12 \\
c_3 &= 1.5954150132395E-09 \\
c_4 &= -2.05905810186661E-07 \\
c_5 &= 0.000016252454421407 \\
c_6 &= -0.000806762471070023 \\
c_7 &= 0.0249378754410694 \\
c_8 &= -0.457552691294531 \\
c_9 &= 4.45855621510378 \\
c_{10} &= -15.8087594445341 \\
c_{11} &= 62.948
\end{aligned}$$

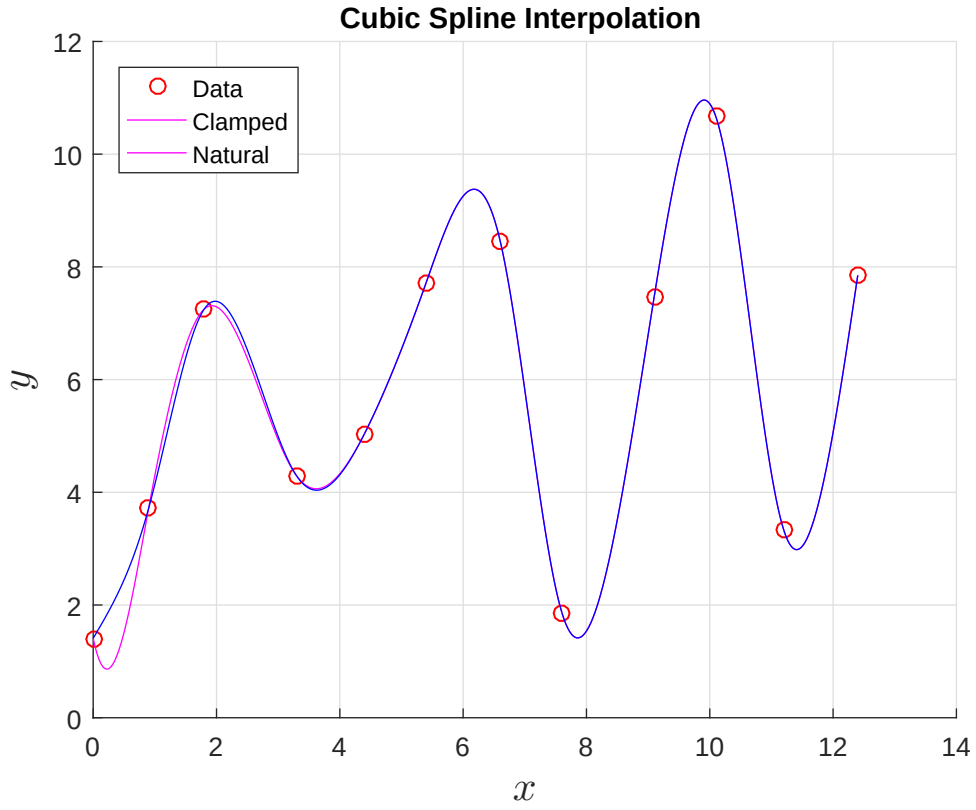
These coefficients should be more stable, so similar between different solutions. The spline coefficients are obviously the same for this simulation by the way spines are constructed.

WeBWorK: There are **2** problems in WeBWorK with written results.

1. (3pts) The graph (identical for all versions) is



2. (7pts) The splines with unclamped and clamped ends are shown on the graph below.



The graphs will vary between students, but there should be a distinct decrease initially for the clamped version, as all versions have a forced initial decline. The natural end conditions on both ends has less variation. The only changes from the class clamped version of the program are the following:

```
27  % Clamped boundary conditions:
28  A(1,1) = 2*h(1);
29  A(1,2) = h(1);
30  A(n,n) = 1;
```

and

```
38  % Vector b:
39  b = zeros(n,1);
40  b(1) = (3/h(1))*(a(2)-a(1)) - 3*fp(1);
```