Robert Heeter
BIOE 391 Numerical Methods – Due 11 March 2022

Problem Set 7 Solutions

1.

  a. **Output**
  PART A: Newton polynomial interpolation
  f(3.4) =
  4.700000 (1st order)
  5.260000 (2nd order)
  4.972000 (3rd order)

  b. **Output**
  PART B: Lagrange polynomial interpolation
  f(3.4) =
  4.700000 (1st order)
  5.260000 (2nd order)
  4.972000 (3rd order)

2. Output
PART A: Newton polynomial interpolation
Density (kg/m^3) at 330K:
1.035800 (1st order)
1.028720 (2nd order)
1.028888 (3rd order, best estimate)
1.027902 (4th order)
1.029021 (5th order)

PART B: Inverse interpolation with 3rd-order Newton interpolation
1.028888 kg/m^3 corresponds to a temperature of 330.000000 K

3.

  a. **Output**
  PART A: Newton polynomial interpolation (3rd order)
  T(x=4,y=3.2) = 43.368000 °C

  b. **Output**
  PART B: Interp2 2D interpolation
  T(x=4.3,y=2.7) = 46.006213 °C

4. Output
Depth of thermocline (m): 1.215110
Flux across interface (cal/(s*m^2)): 73.058930

Robert Heeter
BIOE 391 Numerical Methods – Due 11 March 2022

Figure (to show the cubic spline function with the data and the inflection area)

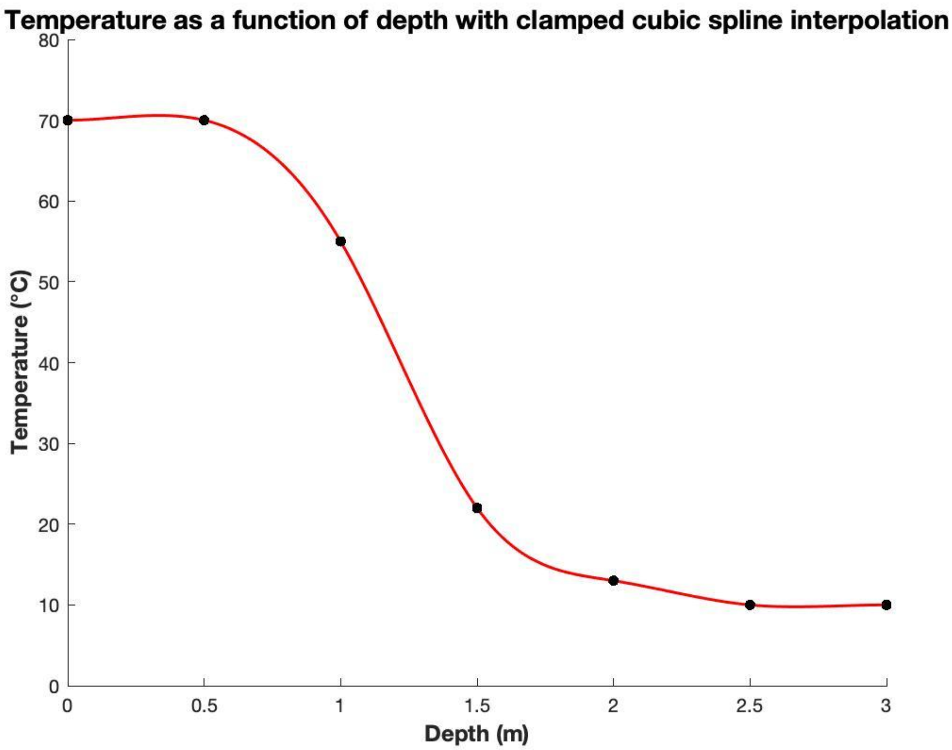
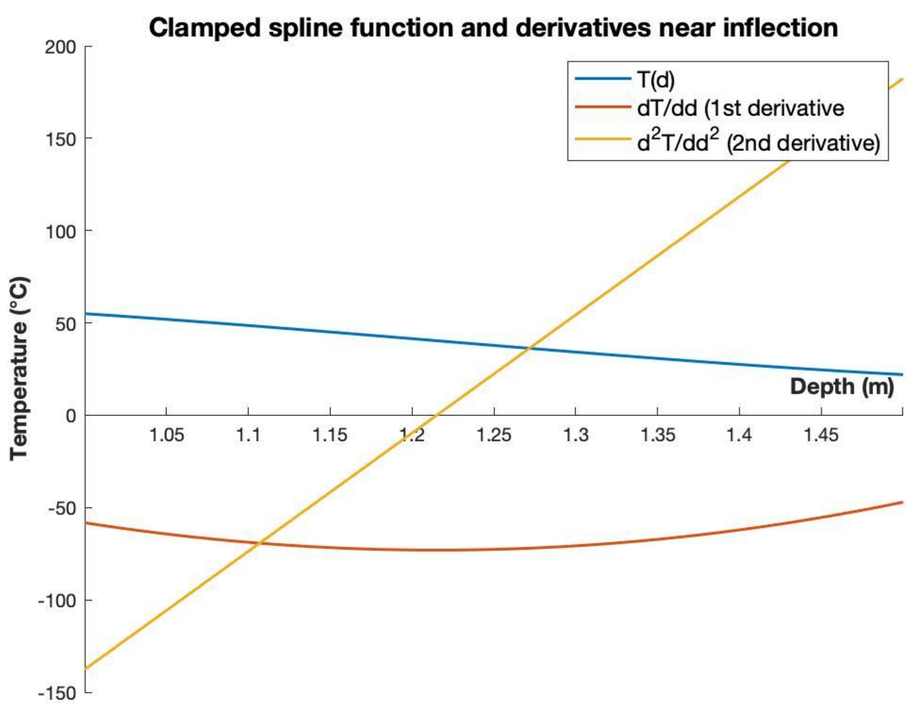


Figure (to show derivatives of the spline function near the inflection point to approximate inflection)

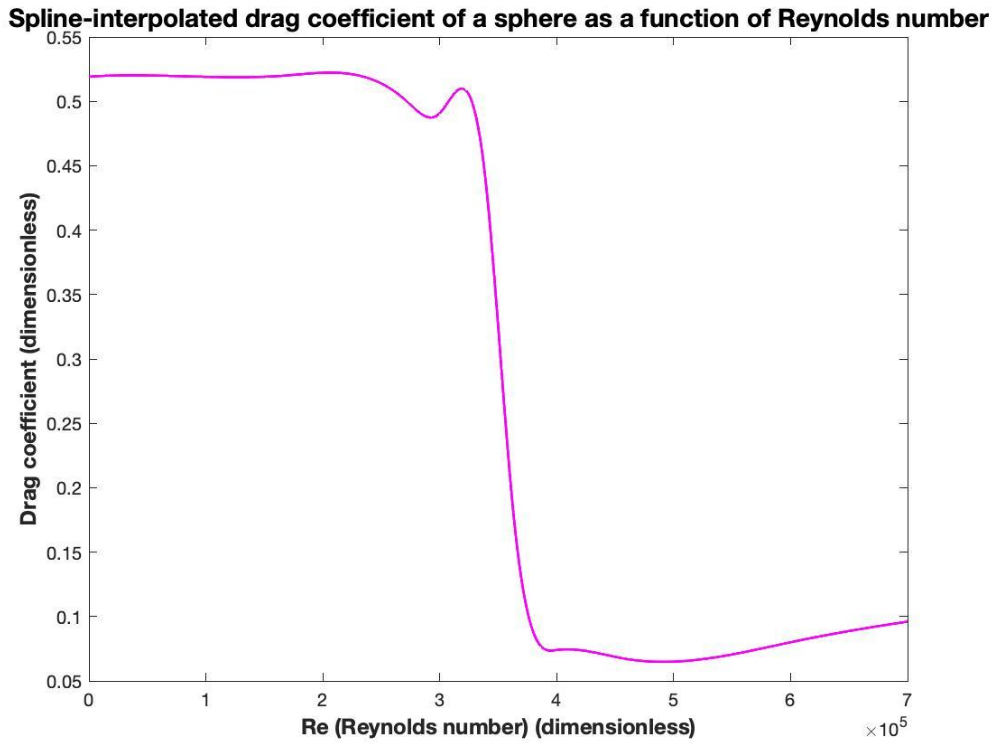


Robert Heeter
BIOE 391 Numerical Methods – Due 11 March 2022

5.

a. See MATLAB code for function *drag*

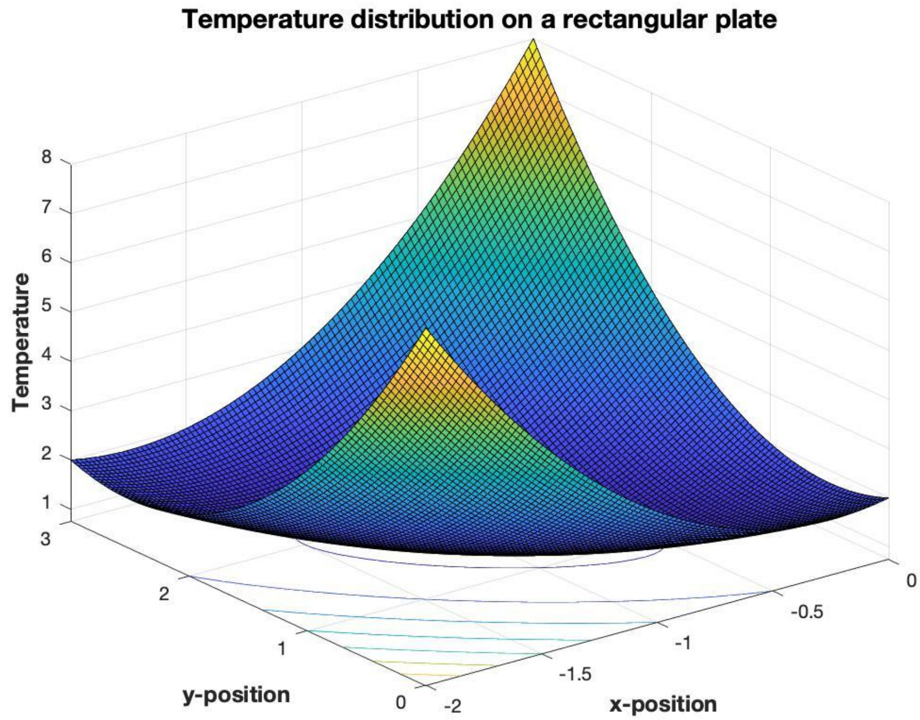
b. Figure



Robert Heeter
BIOE 391 Numerical Methods – Due 11 March 2022

6.

a. Figure



b. Output
PART B: Interp2 2D linear interpolation
Interpolated T(x=-1.63,y=1.627) = 1.462605
Actual T = 1.399909
True relative error = 4.478577 percent

c. Output
PART C: Interp2 2D spline interpolation
Interpolated T(x=-1.63,y=1.627) = 1.399909
Actual T = 1.399909
True relative error = 0.000000 percent

Robert Heeter
BIOE 391 Numerical Methods – Due 11 March 2022

Complete MATLAB Code

```
% Robert Heeter
% BIOE 391 Numerical Methods
% HOMEWORK 7 MATLAB SCRIPT

clc, clf, clear, close all

%% P1. PROBLEM 17.4
disp('P1. PROBLEM 17.4');

x = [1 2 2.5 3 4 5]'; % x values
fx = [0 5 7 6.5 2 0]'; % f(x) values

% PART A: Newton interpolating polynomials
fx_newt_O1 = newtint(x(4:5),fx(4:5),3.4); % use newtint function (below) for interpolation
fx_newt_O2 = newtint(x(3:5),fx(3:5),3.4);
fx_newt_O3 = newtint(x(2:5),fx(2:5),3.4);

% PART B: Lagrange interpolating polynomials
fx_lagr_O1 = lagrint(x(4:5),fx(4:5),3.4); % use lagrint function (below) for interpolation
fx_lagr_O2 = lagrint(x(3:5),fx(3:5),3.4);
fx_lagr_O3 = lagrint(x(2:5),fx(2:5),3.4);

% Display results
fprintf('PART A: Newton polynomial interpolation\nf(3.4) =\n%f (1st order)\n%f (2nd order)\n%f (3rd
order)\n\n',fx_newt_O1,fx_newt_O2,fx_newt_O3);
fprintf('PART B: Lagrange polynomial interpolation\nf(3.4) =\n%f (1st order)\n%f (2nd order)\n%f (3rd
order)\n\n',fx_lagr_O1,fx_lagr_O2,fx_lagr_O3);


%% P2. PROBLEM 17.11
disp('P2. PROBLEM 17.11');

T = [200 250 300 350 400 450]'; % temperature data (K)
rho = [1.708 1.367 1.139 0.967 0.854 0.759]'; % density of nitrogen gas (kg/m^3)

% PART A: Newton polynomial interpolation (orders 1-5)
rho_newt_O1 = newtint(T(3:4),rho(3:4),330); % use newtint function (below) for interpolation
rho_newt_O2 = newtint(T(3:5),rho(3:5),330);
rho_newt_O3 = newtint(T(2:5),rho(2:5),330); % 3rd-order is best estimate
rho_newt_O4 = newtint(T(2:6),rho(2:6),330);
rho_newt_O5 = newtint(T(1:6),rho(1:6),330);

% PART B: Inverse interpolation to check 3rd-order estimate
p = polyfit(T(2:5),rho(2:5),3); % use polyfit to create 3rd-order polynomial through points
p(end) = p(end)-rho_newt_O3; % convert a = p(x) to 0 = p(x)-a, where a is the 3rd-order estimate
[roots] = roots(p); % find root of 0 = p(x)-a to find a = p(x)

% Display results
fprintf('PART A: Newton polynomial interpolation\nDensity (kg/m^3) at 330K:\n%f (1st order)\n%f (2nd
order)\n%f (3rd order, best estimate)\n%f (4th order)\n%f (5th
order)\n\n',rho_newt_O1,rho_newt_O2,rho_newt_O3,rho_newt_O4,rho_newt_O5);
fprintf('PART B: Inverse interpolation with 3rd-order Newton interpolation\n%f kg/m^3 corresponds to a
temperature of %f K\n\n',rho_newt_O3,roots(3));


%% P3. PROBLEM 17.20
disp('P3. PROBLEM 17.20');

T = [100.0 90.00 80.00 70.00 60.00;
     85.00 64.49 53.50 48.15 50.00;
     70.00 48.90 38.43 35.03 40.00;
     55.00 38.78 30.39 27.07 30.00;
     40.00 35.00 30.00 25.00 20.00]; % temperatures of heated plate (Â°C)
```

Robert Heeter

BIOE 391 Numerical Methods – Due 11 March 2022

```
x = [0 2 4 6 8]; % x-position
y = [0 2 4 6 8]'; % y-position

% PART A: Temperature estimate (x=4, y=3.2)
T_newt_O3 = newtint(y(1:4),T(1:4,3),3.2); % use newtint function (below) for interpolation

% PART B: Temperature estimate (x=4.3, y=2.7)
T_int_2D = interp2(x(2:5),y(1:4),T(1:4,2:5),4.3,2.7,'cubic'); % use in-built interp2 function for 2D
interpolation

% Display results
fprintf('PART A: Newton polynomial interpolation (3rd order)\nT(x=4,y=3.2) = %f Â°C\n\n',T_newt_O3);
fprintf('PART B: Interp2 2D interpolation\nT(x=4.3,y=2.7) = %f Â°C\n\n',T_int_2D);


%% P4. PROBLEM 18.2
disp('P4. PROBLEM 18.2');

% Data
d = [0 0.5 1 1.5 2 2.5 3]'; % depth (m)
T = [70 70 55 22 13 10 10]'; % temperature (Â°C)

h = diff(d); % differences between d values
csV = zeros(7,1); % preallocate vector for csM*csC=csV
csM = zeros(7,7); % preallocate matrix for csM*csC=csV

for i = 1:5
    % Fill csM with middle values along tridiagonal
    csM(i+1,i) = h(i);
    csM(i+1,i+1) = 2*(h(i)+h(i+1));
    csM(i+1,i+2) = h(i+1);
    % Fill csV with middle values
    csV(i+1) = 3*(((T(i+2)-T(i+1))/(d(i+2)-d(i+1)))-((T(i+1)-T(i))/(d(i+1)-d(i))));
end

% Fill csM and csV with endpoint values for clamped condition
csM(1,1:2) = [2*h(1) h(1)];
csV(1) = 3*(((T(2)-T(1))/(d(2)-d(1))-0));

% Fill csM and csV with endpoint values for clamped condition
csM(7,6:7) = [h(6) 2*h(6)];
csV(7) = 3*(0-((T(7)-T(6))/(d(7)-d(6))));

% Solve for vector of constants C
csC = csM\csV;

% Solve for vectors for constants A,B,D
csA = T(1:6);
csB = ((T(2:7)-T(1:6))./h)-((h./3).*((2.*csC(1:6))+csC(2:7)));
csD = (csC(2:7)-csC(1:6))./(3.*h);
csC = csC(1:6);

% Create plot of interpolated spline
figure
hold on
for i = 1:6
    Td = @(x) csA(i) + csB(i).*(x-d(i)) + csC(i).*(x-d(i)).^2 + csD(i).*(x-d(i)).^3;
    fplot(Td,[d(i) d(i+1)],'-r','LineWidth',1.5);
end
plot(d,T,'.k','MarkerSize',15);
xlabel('Depth (m)','FontSize',12,'FontWeight','bold');
ylabel('Temperature (Â°C)','FontSize',12,'FontWeight','bold');
title('Temperature as a function of depth with clamped cubic spline
interpolation','FontSize',14,'FontWeight','bold');
hold off
```

Robert Heeter
BIOE 391 Numerical Methods – Due 11 March 2022

```
% Create polynomial equation and differentiate to find inflection
syms Td(x)
Td(x) = csA(3) + csB(3)*(x-d(3)) + csC(3)*(x-d(3))^2 + csD(3)*(x-d(3))^3; % inflection occurs in third
segment from graph above

dTdd = diff(Td); % First derivative
d2Tdd2 = diff(Td,2); % Second derivative

% Create plot of spline function containing inflection and its derivatives
figure
hold on
fplot(Td,[d(3) d(4)],'-','LineWidth',1.5)
fplot(dTdd,[d(3) d(4)],'-','LineWidth',1.5)
fplot(d2Tdd2,[d(3) d(4)],'-','LineWidth',1.5)
legend('T(d)','dT/dd (1st derivative','d^2T/dd^2 (2nd derivative)','FontSize',12);
xlabel('Depth (m)','FontSize',12,'FontWeight','bold');
ylabel('Temperature (Â°C)','FontSize',12,'FontWeight','bold');
title('Clamped spline function and derivatives near inflection','FontSize',14,'FontWeight','bold');
ax = gca;
ax.XAxisLocation = 'origin';
hold off

% Find zero of second derivative for depth of thermocline
[depth,~] = fzero(d2Tdd2,[0,2]);

% Find heat flux across interface with Fourier's law
k = 1; % constant for Fourier's law (cal/(s*m*Â°C))
J = -1*k*dTdd(depth); % Fourier's law

% Display results
fprintf('Depth of thermocline (m): %f\nFlux across interface (cal/(s*m^2)): %f\n\n',depth,J);


%% P5. PROBLEM 18.13
disp('P5. PROBLEM 18.13');

% PART A: Function "Drag" (see below)

% PART B: Plot of C_D (drag coefficient) vs. Re (Reynolds number)
ReCD = [2 5.8 16.8 27.2 29.9 33.9 36.3 40 46 60 100 200 400;
    0.52 0.52 0.52 0.5 0.49 0.44 0.18 0.074 0.067 0.08 0.12 0.16 0.19]; % tabulated data; first row is
Re (dimensionless), second row is C_D (dimensionless)
ReCD(1,:) = 10^4.*ReCD(1,:);

Re_in = 10^4.*(0:0.01:70)'; % input vector for plot
CD_out = drag(ReCD,Re_in); % use drag function (below) for interpolated drag coefficient values

figure
plot(Re_in,CD_out,'-m','LineWidth',1.5);
xlabel('Re (Reynolds number) (dimensionless)','FontSize',12,'FontWeight','bold');
ylabel('Drag coefficient (dimensionless)','FontSize',12,'FontWeight','bold');
title('Spline-interpolated drag coefficient of a sphere as a function of Reynolds
number','FontSize',14,'FontWeight','bold');
disp(' ');


%% P6. PROBLEM 18.14
disp('P6. PROBLEM 18.14');

% PART A: Meshplot of temperature function
T = @(x,y) 2 + x - y +2.*x.^2 + 2.*x.*y + y.^2; % temperature function
x = linspace(-2,0); % vector of x-values
y = (linspace(0,3))'; % vector of y-values
z = T(x,y); % matrix of temperature values

figure
```

```
surfc(x,y,z);
xlabel('x-position','FontSize',12,'FontWeight','bold');
ylabel('y-position','FontSize',12,'FontWeight','bold');
zlabel('Temperature','FontSize',12,'FontWeight','bold');
title('Temperature distribution on a rectangular plate','FontSize',14,'FontWeight','bold');

% PART B: Linear interpolation with interp2
x_points = linspace(-2,0,9);
y_points = (linspace(0,3,9))';
z_points = T(x_points,y_points);
T_int_lin = interp2(x_points,y_points,z_points,-1.63,1.627,'linear'); % use in-built interp2 function
for 2D linear interpolation

% PART C: Spline interpolation with interp2
T_int_spl = interp2(x_points,y_points,z_points,-1.63,1.627,'spline'); % use in-built interp2 function
for 2D spline interpolation

% Display results
T_actual = T(-1.63,1.627);
error_lin = (abs(T_int_lin-T_actual)/T_actual)*100;
error_spl = (abs(T_int_spl-T_actual)/T_actual)*100;

fprintf('PART B: Interp2 2D linear interpolation\nInterpolated T(x=-1.63,y=1.627) = %f\nActual T =
%f\nTrue relative error = %f percent\n\n',T_int_lin,T_actual,error_lin);
fprintf('PART C: Interp2 2D spline interpolation\nInterpolated T(x=-1.63,y=1.627) = %f\nActual T =
%f\nTrue relative error = %f percent\n\n',T_int_spl,T_actual,error_spl);


%% Additional Functions

function CD_out = drag(ReCD,Re_in)
% ABOUT: Determines drag coefficient for a sphere using spline
% interpolation from the Reynolds number.
% INPUTS: ReCD = table of Reynolds numbers and corresponding drag
% coefficients for interpolation; Re_in = input Reynolds numbers to
% interpolate
% OUTPUTS: CD_out = output drag coefficient values from interpolation

    CD_out = spline(ReCD(1,:),ReCD(2,:),Re_in); %  % use in-built spline function for cubic
interpolation

end


function yint = newtint(x,y,xx)
% ABOUT: Newton polynomial interpolation, from textbook .m file.
% INPUTS: x = independent variable; y = dependent variable; xx = value of
% independent variable at which interpolation is calculated
% OUTPUTS: yint = interpolated value of dependent variable

% Compute the finite divided differences in the form of a % difference table
n = length(x);
if length(y)~=n
    error('x and y must be same length');
end

b = zeros(n,n);

% Assign dependent variables to the first column of b
b(:,1) = y(:); % the (:) ensures that y is a column vector
for j = 2:n
    for i = 1:n-j+1
        b(i,j) = (b(i+1,j-1)-b(i,j-1))/(x(i+j-1)-x(i));
    end
end
```

Robert Heeter

BIOE 391 Numerical Methods – Due 11 March 2022

```
% Use the finite divided differences to interpolate
xt = 1;
yint = b(1,1);
for j = 1:n-1
    xt = xt*(xx-x(j));
    yint = yint+b(1,j+1)*xt;
end

end


function yint = lagrint(x,y,xx)
% ABOUT: Lagrange polynomial interpolation, from textbook .m file.
% INPUTS: x = independent variable; y = dependent variable; xx = value of
% independent variable at which interpolation is calculated
% OUTPUTS: yint = interpolated value of dependent variable

n = length(x);
if length(y)~=n
    error('x and y must be same length');
end

s = 0;
for i = 1:n
    product = y(i);
    for j = 1:n
        if i ~= j
            product = product*(xx-x(j))/(x(i)-x(j));
        end
    end
    s = s+product;
end
yint = s;

end
```