EXTRA HOMEWORK

Problem 1:

a. In this case, v interpolates f at 3811 points $x_0<x_1<....<x_{3810}$, and $h = \max_{1<=i<=3810} x_i - x_{i-1}=0.1$. Since $x_i=ih$, the interval [0,381] contains these points obviously. Hence, given the Piecewise Polynomial Interpolation Error Theorem, $|f(x)-v(x)| \le \frac{h}{2} \max_{0\le\xi\le381} |f'(\xi)|$. Since f(x)=sinx in the interval [0,381], f'(x)=cos(x) in the interval [0,381]. H ence,

$$|f(x)-v(x)| \le \frac{h}{2} \max_{0\le\xi\le381} |f'(\xi)| = 0.05 * \max_{0\le\xi\le381} |\cos(\xi)| \le 0.05.$$

b. As part a shows, the bound error is 0.05 in the interval [0,381]. Hence, the first digit in v(x) is guaraneed to agree with those of f(x).

Problem 2:

a. Given the interpolation conditions, we have the following equations:

(1) $s_i(x_i) = f(x_i)$, i = 0,1,..., n;

(2) Since the piecewise quadratic interpolation is C^1, it should be continuous at points x_i where i = 1,2,...,n-1, hence, $s_{i-1}(x_i) = s_i(x_i)$, i = 1,2,...,n-1;

(3) Since the pecewise qiuadratic interpolation is C^1, its derivative should be continuous at points x_i, where i = 1,2,...,n-1, hence , $s_{i-1}'(x_i) = s_i'(x_i)$.

For the first equation, we can get n+1 equations, for the second, we can get n-1 equations and for the third equation, we can get n-1 equations. Hence, there are 3n-1 equations. In addition, f'(a) is given, assume $f'(a)=\beta$. Hence, the 3n equations can solve the 3n unknowns.

An algorithm can got:

step 1: according to (1), $s_i(x_i) = a_i = f(x_i)$, i = 0, 1, … , n-1 $s_{n-1}(x_n) = a_{n-1} + b_{n-1}(x_n-x_{n-1}) + c_{n-1}(x_n-x_{n-1})^2 = f(x_n)$.

for i = 0 to n - 1

$a_i \leftarrow f(x_i)$

endfor

$b_{n-1} + c_{n-1} \leftarrow (f(x_n) - a_{n-1})/h$;

step 2: according to (2),

for i = 1 to n-1

$b_{i-1}h + c_{i-1}h^2 \leftarrow f(x_i) - a_{i-1}$;

endfor

step 3: according to (3),

for i = 1 to n-1

$b_{i-1} + 2c_{i-1}h \leftarrow b_i$;

endfor

step 4: according to (4),

$b_0 + 2c_0 = f'(a)$;

b. Accoding to Piecewise Polynomial Interpolation Error Theorem, for piecewise quadratic interpolation, $|f(x) - v(x)| \le (h^2/(2^3*3!))\max_{a\le\xi\le b}|f'''(\xi)|$. Since f is C^3 in the interval [a,b], hence the error bound is $(h^2/48)\max_{a\le\xi\le b}|f'''(\xi)|$.

Problem 3:

a. Since $x_1, x_2, ..., x_n$ are the n zeros of $\Phi_n(x)$. Hence $\Phi_n(x)$ can be written as,

$$\Phi_n(x) = \beta(x-x_1)(x-x_2)(x-x_3)...(x-x_n).$$

Since $L_j(x) = \prod_{i=1, i\neq j}^{n} \frac{(x-x_i)}{(x_j-x_i)}$, $1 \le j \le n$, hence,

$$\int_a^b w(x)L_j(x)L_k(x)dx = \int_a^b w(x) \prod_{i=1, i\neq j}^{n} \frac{(x-x_i)}{(x_j-x_i)} \prod_{i=1, i\neq k}^{n} \frac{(x-x_i)}{(x_k-x_i)} dx =$$

$$\frac{1}{m}\int_a^b w(x)(x-x_1)...(x-x_{j-1})(x-x_{j+1})...(x-x_n)(x-x_1)...(x-x_{k-1})(x-x_{k+1})...(x-x_n)dx =$$

$$\frac{1}{m}\int_a^b w(x)(x-x_1)..(x-x_{k-1})(x-x_{k+1})...(x-x_{j-1})(x-x_{j+1})...(x-x_n)(x-x_1)...(x-x_{k-1})(x-x_k)(x-x_{k+1})...(x-x_n)dx$$

$$= \frac{1}{m}\int_a^b w(x)(x-x_1)...(x-x_{k-1})(x-x_{k+1})...(x-x_{j-1})(x-x_{j+1})...(x-x_n)\prod_{i=1}^{n}(x-x_i)dx =$$

$$\frac{1}{m}\int_a^b w(x) \prod_{i=1, i\neq k, i\neq j}^{n}(x-x_i)\prod_{i=1}^{n}(x-x_i)dx \quad (1)$$

Where $m = \prod_{i=1, i\neq j}^{n} \frac{1}{(x_j-x_i)} \prod_{i=1, i\neq k}^{n} \frac{1}{(x_k-x_i)}$ is a constant. Given the property of a given family of orthogonal polynomials that $\Phi_n(x)$ is orthogonal to all polynomials of degree < n, then,

$\frac{1}{m}\int_a^b w(x) \prod_{i=1, i\neq k, i\neq j}^{n}(x-x_i)\prod_{i=1}^{n}(x-x_i)dx = \frac{1}{m}\frac{1}{\beta}\int_a^b w(x) \prod_{i=1, i\neq k, i\neq j}^{n}(x-x_i)\beta(x-x_1)...(x-x_n)dx =$

$\frac{1}{m}\frac{1}{\beta}\int_a^b w(x) \prod_{i=1, i\neq k, i\neq j}^{n}(x-x_i)\Phi n(x)dx = 0$ since $\prod_{i=1, i\neq k, i\neq j}^{n}(x-x_i)$ has a degree < n. Hence,

$\int_a^b w(x)L_j(x)L_k(x)dx$ =0 and the Lagrange polynoimals of degree n-1 based on these points are orthogonal to each other.

b. $||p_{n-1}(x)||^2 = \int_a^b w(x)[p_{n-1}(x)]^2 dx = \int_a^b w(x)(y_1L_1 + y_2L_2 + ... + y_nL_n)(y_1L_1 + y_2L_2 + ... + y_nL_n)dx =$

$= \int_a^b w(x)(y_1^2L_1^2 + y_1^2L_1^2 + ... + y_n^2L_n^2)dx + \int_a^b w(x)(\sum_{i=1}^{n}\sum_{j=1}^{n} y_iy_jL_iL_j)dx = \int_a^b w(x)(y_1^2L_1^2 + y_1^2L_1^2 + ... + y_n^2L_n^2)dx + 0 =$

$\sum_{i=1}^{n} y_i^2 \int_a^b w(x)\, L_i^2 dx = \sum_{i=1}^{n} y_i^2 ||L_k||^2$. In the process, $\int_a^b w(x)(\sum_{i=1}^{n}\sum_{j=1}^{n} y_iy_jL_iL_j)dx$ = 0 as part a shows.

Problem 4:

a. since f(x) can be evaluated at points $x_0 \pm ih$, i = 0,1,2,... Hence,

$f(x_0+h) \approx f(x_0) + h'f'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) + \frac{h^4}{24}f^{(4)}(x_0) + \frac{h^5}{120}f^{(5)}(\xi_1)$ (1)

$f(x_0-h) \approx f(x_0) - h'f'(x_0) + \frac{h^2}{2}f''(x_0) - \frac{h^3}{6}f'''(x_0) + \frac{h^4}{24}f^{(4)}(x_0) - \frac{h^5}{120}f^{(5)}(\xi_2)$ (2)

after (1) - (2), we can get,

$f(x_0+h) - f(x_0-h) \approx 2hf'(x_0) + \frac{h^3}{3}f'''(x_0) + \frac{h^5}{60}f^{(5)}(\xi_3)$ (3)

Here, $\xi_3 \in [x_0 - h, x_0 + h]$.

According to (3) and f(x) can be evaluated at $x_0 \pm 2h$, we can get,

$f(x_0 + 2h) - f(x_0 - 2h) \approx 4hf'(x_0) + \frac{8h^3}{3} f'''(x_0) + \frac{32h^5}{60} f^{(5)}(\xi_4)$ (4)

Here, $\xi_4 \in [x_0 - 2h, x_0 + 2h]$.

after (4)-2*(3), we can get,

$f(x_0 + 2h) - f(x_0 - 2h) - 2(f(x_0 + h) - f(x_0 - h)) \approx 2h^3 f'''(x_0) + \frac{h^5}{2} f^{(5)}(\xi_5)$

$\Rightarrow f'''(x_0) = \frac{1}{2h^3}(f(x_0 + 2h) - f(x_0 - 2h) - 2f(x_0 + h) + 2f(x_0 - h)) - \frac{h^2}{4} f^{(5)}(\xi_5)$. (5)

Here, $\xi_5 \in [\xi_3, \xi_4]$.

b. Using $\frac{1}{2h^3}(f(2h) - f(-2h) - 2f(h) + 2f(h)) - \frac{h^2}{4} f^{(5)}(0)$ to approximate $f'''(0)$ with different h. In this case, f'''(0)=e^0=1.

different h	approximation	absoluate error
10^{-1}	1.000461391922780	4.613919227796082e-04
10^{-2}	1.000008395206649	8.395206649192133e-06
10^{-3}	1.000000153626301	1.536263010137873e-07
10^{-4}	1.000000002796942	2.796942233374011e-09
10^{-5}	0.999999999957539	4.246070162139404e-11
10^{-6}	1.000000000574193	5.741926933922059e-10
10^{-7}	1.000000046367970	4.636797013013449e-08
10^{-8}	1.000000293781834	2.937818339709963e-07
10^{-9}	0.999923764079245	7.623592075545194e-05

As the table shows, as h decreases by a factor 10, the absoluate error decreases by a factor almost 100. However, when h is rather small, the error becomes larger. Hence, for the larger values of h, the formual is indeed second order accurate. Moreover, when h = 10^{-5}, the approximation is closest to 1.

c. As the formula in part a shows, when h is rathe close to 0, the truncation error will become 0. However, the denominator h^3 will become close to 0 leading to the item - $\frac{1}{2h^3}(f(x_0 + 2h) - f(x_0 - 2h) - 2f(x_0 + h) + 2f(x_0 - h)$ becomes larger if f(x)'s order is larger than 3. I.e., $lim_{h \to 0} \frac{f(h)}{h^3}$ =0. Just as part b shows, since $lim_{h \to 0} \frac{e^h}{h^3}$ =0, hence when h is rather close to 0, the absoluate error will becomes larger. As the table above shows, when h is larger than 10^{-5}, the absoluate error becomes larger.

d. Just continue to expand the Taylor explansion to the seventh order. Then, we can get,

$f(x_0+2h) - f(x_0-2h) - 2(f(x_0+h)-f(x_0-h)) \approx 2h^3 f'''(x_0) + \frac{h^5}{2} f^{(5)}(x_0) + O(h^7)$. Replacing h with 2h, we can get a new equation, then combine these two equations to reduce the fifth order item, we can get a new equation with the truncation error $O(h^4)$. Hence, the points needed are $f(x_0 \pm h), f(x_0 \pm 2h), f(x_0 \pm 4h),$ six points in total.

Problem 7:

For this problem, the difference between it with the probem of homework 8 is the different boundary condition. Although the Laplcace condition is still $\nabla^2 u = 0$, the Neumann condition on boundary B1 is ∇u*n = 10 and the Neumann condition on boundary B4 is ∇u*n = -h.5. Hence, the local r.h.s is no longer all zeros. So we have to figure out the non-zero local r.h.s. I rewrote the function **Local_Matrix** and **generate_grid** in homework 8.

```
function [ sort, P, Val, DT, DT_I ] = generate_grid( h )
figure()
axis([0 2 0 2]);
axis equal;
M = floor(2/h);
h = 2/M;
sort = [];
Val = [];

P = [];
for i = 0:h:1
   for j = 0:h:2
      P = [P;j,i];
   end
end
for i = 1+h:h:2
   for j = 1:h:2
      P = [P;j,i];
   end
end
total_number = size(P,1);

for id = 1:total_number
   if P(id,1) == 0 && P(id, 2) == 0
      n1 = id;
   elseif P(id,1) == 2 && P(id,2) == 0
      n2 = id;
   elseif P(id,1) == 2 && P(id,2) == 2
      n3 = id;
   elseif P(id,1) == 1 && P(id,2) == 2
      n4 = id;
   elseif P(id,1) == 1 && P(id,2) == 1
      n5 = id;
   elseif P(id,1) == 0 && P(id,2) == 1
      n6 = id;
```

```
    end
    if P(id, 2) == 0 || (P(id, 2) == 1 && P(id, 1) <= 1) || (P(id,1) == 1 && P(id,2) >= 1)|| P(id,1) == 2
        sort = [sort;1];
        Val = [Val;0];
    elseif P(id,1) == 0
        sort = [sort;2];
        Val = [Val;NaN];
    elseif P(id,2) == 2
        sort = [sort;3];
        Val = [Val;NaN];
    else
        sort = [sort;0];
        Val = [Val;NaN];
    end
end
 C = [n1 n2;n2 n3;n3 n4;n4 n5;n5 n6;n6 n1];
for id = 1:total_number
    if P(id,1) == 0 && P(id,2) <= 1
        sort = [sort;1];
        Val = [Val;1];
    elseif P(id,1) == 2
        sort = [sort;1];
        Val = [Val;0];
    else
        sort = [sort;0];
        Val = [Val;NaN];
    end
end

plot(P(C'),P(C'+size(P,1)),'-r','LineWidth', 3);
DT = delaunayTriangulation(P, C);
IO = isInterior(DT);
hold on
triplot(DT(IO, :),DT.Points(:,1), DT.Points(:,2),'LineWidth', 1)
hold off
DT_I = DT(IO,:);
end
```

EXPLANATION: Here, the boundary conditions are different from homework 8.

```
function [ A, f ] = Local_Matrix(x,y,P_TMP)
[a,b,c, Area_A] = Diff_Triangle(x,y);
A = zeros(3,3);
res = [];
res1 = [];
sign = 0;
sign1 = 0;
for i = 1:3
    if P_TMP(i,1) == 0
        res = [res;P_TMP(i,2)];
        sign = sign + 1;
```

```
    end
    if P_TMP(i,2) == 2
        res1 = [res1;P_TMP(i,1)];
        sign1 = sign1 + 1;
    end
end
f = zeros(3,1);
for i = 1:3
    if sign < 2 && sign1 < 2
        f(i) = 0;
    elseif sign == 2
        if res(2) < res(1)
            res = res([2;1],:);
        end
        syms y;
        f(i) = int(-10*(a(i)+c(i)*y),res(1),res(2));
    elseif sign1 == 2

        if res1(2) < res1(1)
            res1 = res1([2;1],:);
        end
        syms x;
        f(i) = int(5*(a(i)+b(i)*x),res1(1),res1(2));
    end
   for j = 1:3
       A(i,j) = Area_A*(b(i)*b(j) + c(i)*c(j));
   end
end
end
```

EXPLANATION: Firstly, I count the number of points on the boundary B1 and B4 within an element. If sign = 2, it implies that there are two points on B1, $f = -10\int_a^b((a_i + c_i y)/2A)dy$, where a and b are the ordinates of the two points on B1. And if sign1 = 2, it implies that there are two points on B4, $f = 5\int_c^d((a_i + b_i x)/2A)dx$, where c and d are the abscissas of the two points on B4. Otherwise, f = 0.

Set h = 0.05 and solve Ax=b using bicg method, then I got the visualization of the results as Figure 1 and Figure 2 below show.

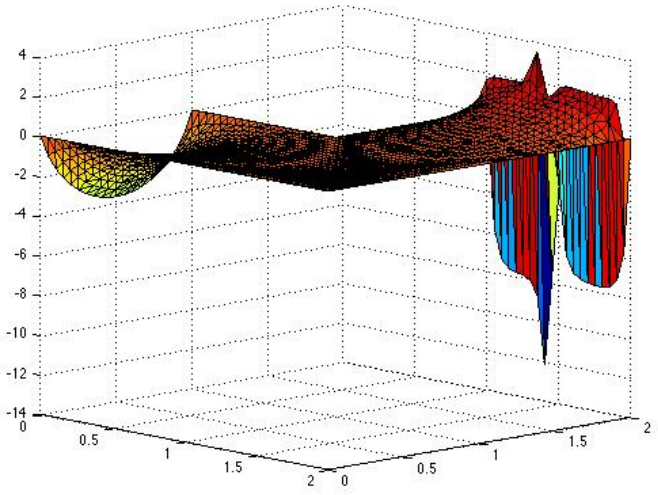

Figure 1.

As Figure 1 shows, the temperature near the boundary B1 is below 0. That's because $\nabla u \cdot n = 10$, the temperature interior near B1 is higher than that outside B1 and the heat inside will flow away, making the temperature values inside near B1 smaller. In addition, the Neumann condition on B4 is $\nabla u \cdot n = -5$ and it implies the temperature inside B4 is lower than that outside B4, making heat outside flow inside and the temperatures inside B4 higher than 0. As Figure 1 and Figure 2 show, the temperature values inside B1 are lower than 0, the temperature values inside B4 are higher than 0 and other parts all have temperature values 0 as expected.

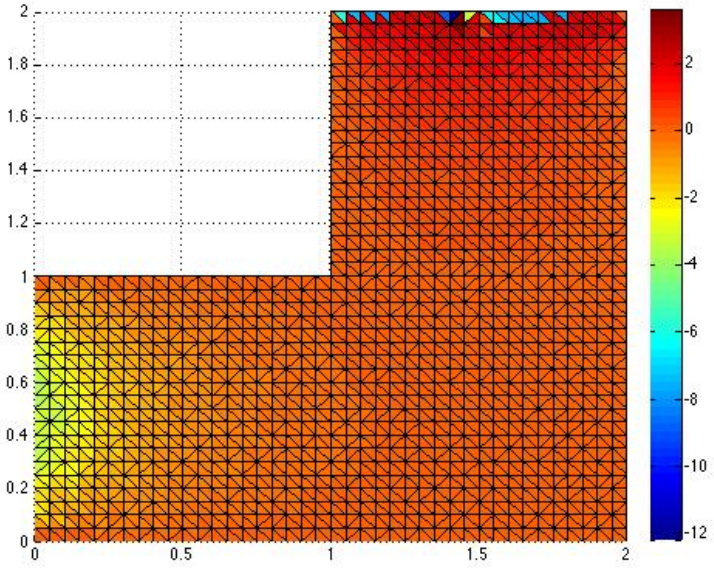

Figure 2.

Problem 9:

As the equation 5.66 in cs 6220 notes show, the error upper bound for two-dimension elements are $\frac{1}{8}(|\frac{\partial^2 uFE}{\partial x^2}|+|\frac{\partial^2 uFE}{\partial x \partial y}|+|\frac{\partial^2 uFE}{\partial y^2}|)h^2$. I used the the equation 5.68 in cs6220 notes to create a finite difference approximation of the second derivative using first derivates. I wrote the matlab code below to find out the points who have a larger error upper bound than others. In the input, P is the coordinates of the points and the vector res is the values of all points.

```
function [bad_point] = judge(res, P)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
total_number = size(P,1);
FDX = zeros(total_number, 1);
FDY = zeros(total_number, 1);
[TMP1, index1] = sortrows(P,1);
[TMP2, index2] = sortrows(P,2);
res1 = res(index1,:);
res2 = res(index2,:);
for i = 1:total_number - 1
  FDY(i) = abs((res1(i+1)-res1(i))/(TMP1(i+1,2)-TMP1(i,2)));
  FDX(i) = abs((res2(i+1)-res2(i))/(TMP2(i+1,1)-TMP2(i,1)));
end
  FDY(i+1) = FDY(i);
  FDX(i+1) = FDX(i);
FD = zeros(total_number, 1);
for i = 1:total_number - 1
    FD(i) = abs(FDX(index2(i+1))-FDX(index2(i))) + abs(FDY(index1(i))-FDY(index1(i)));
end
FD(i+1) = FD(i);
bad_point = FD;
pos = [];
for i = 1:size(FD,1)
    if FD(i) > 0.3
      pos = [pos;P(i,:)];
    end
end
disp(pos);
```

The results shown below are the coordinates points whose absoluate error bound is larger than 0.3. It can be found that most points are on the centerial corner of the L-shape region. Hence, more points should be added everywhere especailly in the center.

```
    0.6667    0.3333
    1.3333    0.6667
    1.3333    1.3333
    1.0000    1.6667
    1.3333    2.0000
```

2.0000	1.3333
0.8333	0.8333
1.1667	0.8333
1.5000	0.5000
1.5000	0.8333
1.0833	0.9167
1.0833	1.0833