```
%%%%%%%%%%%%%%%%%%%%%% Global Variables %%%%%%%%%%%%%%%%%%
global nodeco  elnode  bdynde  bdyedge  nVert  nedge
global GlobalV  GlobalP  GlobalS  GlobalG
global dimTvel  dimTpre  dimTstr  dimTGrv
global vel_bas_type  pre_bas_type  str_bas_type  Grv_bas_type
global quad_rul num mesh

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vel_bas_type = 'CtsLin' ;
quad_rul = 'quad_73';
num = 1;
listmesh = ["R4x4","R6x6","R8x8","R10x10","R12x12"];

l = zeros(5,5);
for i=1:5
    mesh = listmesh(i);
    [uL2Error, GraduL2Error, H1uError]=Drv_ConDiff;
    l(i,:)=[uL2Error,2*i+2, 0,H1uError, 0];
end
for i=1:4
    l(i+1,3) = log(l(i,1)/l(i+1,1))/log(l(i+1,2)/l(i,2));
    l(i+1,5) = log(l(i,4)/l(i+1,4))/log(l(i+1,2)/m(i,2));
end
lin=table(l(:,2),l(:,1),l(:,3),l(:,4),l(:,5));
lin.Properties.VariableNames=["Mesh","uL2Error","alpha
uL2","H1uError","alpha H1u"];

disp("Part 2")
disp("Using Continuous Linears")
disp(lin)
disp("Thus the experimental convergence rate is about 1 for H_1 norm, which
is what we would expect since" +...
    " the error in the H_1 norm should be in the first term because we are
using continuous linear aproximations" + ...
    " and the error for the L2 norm is about 2 which is again what we would
expect using continuous linears.")
disp("I believe that my program works correctly as it solves the given
problems to the error" + ...
    " that we would expect given the continuous linear approximation.")

vel_bas_type = 'CtsQuad' ;
l = zeros(5,5);
for i=1:5
    mesh = listmesh(i);
    [uL2Error, GraduL2Error, H1uError]=Drv_ConDiff;
    l(i,:)=[uL2Error,2*i+2, 0,H1uError, 0];
end
for i=1:4
    l(i+1,3) = log(l(i,1)/l(i+1,1))/log(l(i+1,2)/l(i,2));
    l(i+1,5) = log(l(i,4)/l(i+1,4))/log(l(i+1,2)/m(i,2));
end
```

```
quad=table(l(:,2),l(:,1),l(:,3),l(:,4),l(:,5));
quad.Properties.VariableNames=["Mesh","uL2Error","alpha
uL2","H1uError","alpha H1u"];


disp("Using Continuous Quadratics")
disp(quad)

disp("I believe that my program works correctly as it solves the given
problems to machine error" + ...
    " which is what we would expect given the continuous quadratic
approximation on a quadratic solution.")


vel_bas_type = 'CtsLin' ;
quad_rul = 'quad_73';
num = 0;

l = zeros(5,5);
for i=1:5
    mesh = listmesh(i);
    [uL2Error, GraduL2Error, H1uError]=Drv_ConDiff;
    l(i,:)=[uL2Error,2*i+2, 0,H1uError, 0];
end
for i=1:4
    l(i+1,3) = log(l(i,1)/l(i+1,1))/log(l(i+1,2)/l(i,2));
    l(i+1,5) = log(l(i,4)/l(i+1,4))/log(l(i+1,2)/m(i,2));
end
lin=table(l(:,2),l(:,1),l(:,3),l(:,4),l(:,5));
lin.Properties.VariableNames=["Mesh","uL2Error","alpha
uL2","H1uError","alpha H1u"];

disp("Part 3a")
disp("Using Continuous Linears")
disp(lin)
disp("Thus the experimental convergence rate is about 1 for H_1 norm, which
is what we would expect since" +...
    " the error in the H_1 norm should be in the first term because we are
using continuous linear aproximations" + ...
    " and the error for the L2 norm is about 2 which is again what we would
expect using continuous linears.")

vel_bas_type = 'CtsQuad' ;
l = zeros(5,5);
for i=1:5
    mesh = listmesh(i);
    [uL2Error, GraduL2Error, H1uError]=Drv_ConDiff;
    l(i,:)=[uL2Error,2*i+2, 0,H1uError, 0];
end
for i=1:4
    l(i+1,3) = log(l(i,1)/l(i+1,1))/log(l(i+1,2)/l(i,2));
    l(i+1,5) = log(l(i,4)/l(i+1,4))/log(l(i+1,2)/m(i,2));
end
quad=table(l(:,2),l(:,1),l(:,3),l(:,4),l(:,5));
```

```
quad.Properties.VariableNames=["Mesh","uL2Error","alpha
uL2","H1uError","alpha H1u"];


disp("Using Continuous Quadratics")
disp(quad)

disp("Thus the experimental convergence rate is about 2 for H_1 norm, which
is what we would expect since" +...
    " the error in the H_1 norm should be in the second term because we are
using continuous quadratic aproximations" + ...
    " and the error for the L2 norm is about 3 which is again what we would
expect using continuous quadratics.")


quad_rul = 'quad_75';

l = zeros(5,5);
for i=1:5
    mesh = listmesh(i);
    [uL2Error, GraduL2Error, H1uError]=Drv_ConDiff;
    l(i,:)=[uL2Error,2*i+2, 0,H1uError, 0];
end
for i=1:4
    l(i+1,3) = log(l(i,1)/l(i+1,1))/log(l(i+1,2)/l(i,2));
    l(i+1,5) = log(l(i,4)/l(i+1,4))/log(l(i+1,2)/m(i,2));
end
quad=table(l(:,2),l(:,1),l(:,3),l(:,4),l(:,5));
quad.Properties.VariableNames=["Mesh","uL2Error","alpha
uL2","H1uError","alpha H1u"];

disp("Part 3b")
disp("Using Continuous Quadratics")
disp(quad)

disp("I did not see a difference in the experimental convergence rate after
changing the quad rule." + ...
    " If anything, it got worse.")
```

Part 2

Using Continuous Linears

Mesh	uL2Error	alpha uL2	H1uError	alpha H1u
4	0.1256	0	1.2966	0
6	0.056026	1.991	0.86088	1.0101
8	0.031547	1.9964	0.64457	1.0059
10	0.020196	1.9987	0.5152	1.004
12	0.014025	1.9999	0.4291	1.003

Thus the experimental convergence rate is about 1 for H_1 norm, which is what we would expect since the error in the H_1 norm should be in the first term because we are using continuous linear aproximations and the error for the L2 norm is about 2 which is again what we would expect using continuous

linears.
I believe that my program works correctly as it solves the given problems to the error that we would expect given the continuous linear approximation.
Using Continuous Quadratics

Mesh	uL2Error	alpha uL2	H1uError	alpha H1u
4	3.047e-15	0	2.5158e-14	0
6	1.3038e-14	-3.5854	1.1881e-13	-3.8286
8	8.864e-15	1.3414	7.6957e-14	1.5096
10	7.0471e-14	-9.2909	2.8889e-13	-5.9281
12	1.893e-14	7.2094	1.8826e-13	2.3487

I believe that my program works correctly as it solves the given problems to machine error which is what we would expect given the continuous quadratic approximation on a quadratic solution.
Part 3a
Using Continuous Linears

Mesh	uL2Error	alpha uL2	H1uError	alpha H1u
4	0.32222	0	3.2795	0
6	0.17828	1.4598	2.3855	0.78498
8	0.098058	2.0779	1.7869	1.0043
10	0.060215	2.1853	1.4122	1.0545
12	0.041956	1.9816	1.1701	1.0317

Thus the experimental convergence rate is about 1 for H_1 norm, which is what we would expect since the error in the H_1 norm should be in the first term because we are using continuous linear aproximations and the error for the L2 norm is about 2 which is again what we would expect using continuous linears.
Using Continuous Quadratics

Mesh	uL2Error	alpha uL2	H1uError	alpha H1u
4	0.092447	0	1.4015	0
6	0.019723	3.8101	0.51914	2.4493
8	0.0084326	2.9535	0.30287	1.8732
10	0.0046116	2.7047	0.205	1.7491
12	0.0028562	2.6276	0.14932	1.738

Thus the experimental convergence rate is about 2 for H_1 norm, which is what we would expect since the error in the H_1 norm should be in the second term because we are using continuous quadratic aproximations and the error for the L2 norm is about 3 which is again what we would expect using continuous quadratics.
Part 3b
Using Continuous Quadratics

Mesh	uL2Error	alpha uL2	H1uError	alpha H1u
4	0.08336	0	1.3742	0
6	0.018036	3.7755	0.50222	2.4825

```
   8      0.008108     2.7791      0.29605      1.8371
  10      0.004465     2.6735      0.20179      1.7178
  12     0.0028011     2.5574      0.14771       1.711
```

```
I did not see a difference in the experimental convergence rate after
changing the quad rule. If anything, it got worse.
```

Published with MATLAB® R2023b

```
function [localmat] = inner_prod_ten0_Grad_ten0_Vec(triag_no, quad_rul, ...
   vc_fun, ten0a_type, ten0b_type)
%
% This function computes, for triangle triag_no, the integrals
% of V.\del u*v
%


%%%%%%%%%%%%%%%%%%%%%% Global Variables %%%%%%%%%%%%%%%%%%%
global nodeco  elnode  bdynde  bdyedge  nVert  nedge
global GlobalV  GlobalP  GlobalS  GlobalG
global dimTvel  dimTpre  dimTstr  dimTGrv
global vel_bas_type  pre_bas_type  str_bas_type  Grv_bas_type
global quad_rul num

% Description of triangle.
cotri(1:3,1) = nodeco(elnode(triag_no, 1:3), 1) ;
cotri(1:3,2) = nodeco(elnode(triag_no, 1:3), 2) ;

Jmat = [(cotri(2,1) - cotri(1,1)), (cotri(3,1) - cotri(1,1)) ; ...
        (cotri(2,2) - cotri(1,2)) , (cotri(3,2) - cotri(1,2)) ] ;
detJ = abs(Jmat(1,1)*Jmat(2,2) - Jmat(1,2)*Jmat(2,1));
JInv = inv(Jmat) ;

% Evaluation of quadrature points and quadrature weights.
[quad_pts, quad_wghts] = feval(quad_rul) ;
nqpts = size(quad_pts,1) ;

% Adjust points and weights to account for size of true triangle.
xy_pts = ( Jmat * quad_pts.' ).' ;
xy_pts(:,1) = cotri(1,1) + xy_pts(:,1) ;
xy_pts(:,2) = cotri(1,2) + xy_pts(:,2) ;
quad_wghts = detJ * quad_wghts ;

% Evaluate the scalar multiplier at the quadrature points.
vcfun_vals = feval(vc_fun, xy_pts, triag_no) ;

% Evaluate Basis Functions and their Gradients at quad. points.
[ten0a, Gradten0a] = feval(ten0a_type, quad_pts) ;
nbas0a = size(ten0a,1) ;

[ten0b, Gradten0b] = feval(ten0b_type, quad_pts) ;
nbas0b = size(ten0b,1) ;

% Do appropriate multiplies to get the true Gradients.
for iq = 1:nqpts
   Gradtrue1(:,:,iq) = Gradten0b(:,:,iq) * JInv ;
end

% MATLAB will not take the transpose nor do the multiplication of my
% Gradtrue matrices -- Hence we introduce tempM1 and tempM2
tempM1(:,:) = Gradtrue1(:,1,:) ;
```

```
tempM2(:,:) = Gradtrue1(:,2,:) ;

% Now to do the evaluations of the integrals.
for iq = 1:nqpts
   tempM1(:,iq) = quad_wghts(iq) * vcfun_vals(1,iq) * tempM1(:,iq) ;
   tempM2(:,iq) = quad_wghts(iq) * vcfun_vals(2,iq) * tempM2(:,iq) ;
end

mat1 = ten0a*(tempM1+tempM2)';

localmat = [ mat1 ] ;
```

Published with MATLAB® R2023b

```
function [localmat] = inner_prod_ten0(triag_no, quad_rul, scal_fun,
ten0_type)

%
% This function computes, the values for integral of f*v
% at the requested xy_pts points in triangle triag_no.
%  The vector of values is returned in localmat.
%
%

%%%%%%%%%%%%%%%%%%%% Global Variables %%%%%%%%%%%%%%%%%
global nodeco  elnode  bdynde  bdyedge  nVert  nedge
global GlobalV  GlobalP  GlobalS  GlobalG
global dimTvel  dimTpre  dimTstr  dimTGrv
global vel_bas_type  pre_bas_type  str_bas_type  Grv_bas_type
global quad_rul num

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Description of triangle.
cotri(1:3,1) = nodeco(elnode(triag_no, 1:3), 1) ;
cotri(1:3,2) = nodeco(elnode(triag_no, 1:3), 2) ;

Jmat = [(cotri(2,1) - cotri(1,1)), (cotri(3,1) - cotri(1,1)) ; ...
        (cotri(2,2) - cotri(1,2)) , (cotri(3,2) - cotri(1,2)) ] ;
detJ = abs(Jmat(1,1)*Jmat(2,2) - Jmat(1,2)*Jmat(2,1));
JInv = inv(Jmat) ;

% Evaluation of quadrature points and quadrature weights.
[quad_pts, quad_wghts] = feval(quad_rul) ;
nqpts = size(quad_pts,1) ;

% Adjust points and weights to account for size of true triangle.
xy_pts = ( Jmat * quad_pts.' ).' ;
xy_pts(:,1) = cotri(1,1) + xy_pts(:,1) ;
xy_pts(:,2) = cotri(1,2) + xy_pts(:,2) ;
quad_wghts = detJ * quad_wghts ;

% Evaluate the scalar multiplier at the quadrature points.
sfun_vals = feval(scal_fun, xy_pts, triag_no) ;

% Evaluate Basis Functions and their Gradients at quad. points.
[ten0a, Gradten0a] = feval(ten0_type, quad_pts) ;
nbas0a = size(ten0a,1) ;

% Now to do the evaluations of the integrals.
for iq = 1:nqpts
   ten0a(:,iq) = quad_wghts(iq) * ten0a(:,iq) ;
end

mat1 = ten0a*sfun_vals';
```

```
localmat = [ mat1 ] ;
```

Published with MATLAB® R2023b

```
function [CQuadVal, GradCQuadVal] = CtsQuad(quad_pts)
%
% This function computes the values of the continuous
% quadratic basis functions, and of its gradient, at
% the quadrature points quad_pts --- on the reference triangle.
%

nqpt = size(quad_pts,1) ;

x=quad_pts(:,1)';
y=quad_pts(:,2)';

CQuadVal(1,:) = 2*(1 - x - y).*(1/2 - x - y) ;
CQuadVal(2,:) = 2*x.*(x - 1/2) ;
CQuadVal(3,:) = 2*y.*(y - 1/2) ;
CQuadVal(4,:) = 4*x.*y ;
CQuadVal(5,:) = 4*(1 - x - y).*y ;
CQuadVal(6,:) = 4*(1 - x - y).*x ;

GradCQuadVal(1,1,:) = -2*(1.0 - x - y)-2*(1/2 - x - y);
GradCQuadVal(2,1,:) =  2*x+2*(x-1/2) ;
GradCQuadVal(3,1,:) =  zeros(1,nqpt) ;
GradCQuadVal(4,1,:) =  4*y ;
GradCQuadVal(5,1,:) =  -4*y ;
GradCQuadVal(6,1,:) =  4*(1-x-y)-4*x ;

GradCQuadVal(1,2,:) = -2*(1.0 - x - y)-2*(1/2 - x - y);
GradCQuadVal(2,2,:) = zeros(1,nqpt) ;
GradCQuadVal(3,2,:) = 2*y+2*(y-1/2) ;
GradCQuadVal(4,2,:) = 4*x ;
GradCQuadVal(5,2,:) = 4*(1-x-y)-4*y ;
GradCQuadVal(6,2,:) = -4*x ;
```

Published with MATLAB® R2023b

```
function [quad_pts, quad_wghts] = quad_75
%
% This function contains weights and quadrature points
% which are exact for polynomials of degree five.
%

quad_pts(1,:) = [(6+sqrt(15))/21, (9-2*sqrt(15))/21] ;
quad_pts(2,:) = [(6+sqrt(15))/21, (6+sqrt(15))/21] ;
quad_pts(3,:) = [(9-2*sqrt(15))/21, (6+sqrt(15))/21] ;
quad_pts(4,:) = [(6-sqrt(15))/21, (6-sqrt(15))/21] ;
quad_pts(5,:) = [(9+2*sqrt(15))/21, (6-sqrt(15))/21] ;
quad_pts(6,:) = [(6-sqrt(15))/21, (9+2*sqrt(15))/21] ;
quad_pts(7,:) = [1/3, 1/3] ;


quad_wghts(1:3) = (155+sqrt(15))/2400* ones(1,3) ;
quad_wghts(4:6) = (155-sqrt(15))/2400* ones(1,3) ;
quad_wghts(7) = 270/2400 ;
```

Published with MATLAB® R2023b

```
function [localmat] = ffun(xy_pts, triag_no)

%
% This function computes, the values for ffun
% at the requested xy_pts points in triangle triag_no.
%  The vector of values is returned in localmat.
%
%

%%%%%%%%%%%%%%%%%%%%%% Global Variables %%%%%%%%%%%%%%%%%%
global nodeco  elnode  bdynde  bdyedge  nVert  nedge
global GlobalV  GlobalP  GlobalS  GlobalG
global dimTvel  dimTpre  dimTstr  dimTGrv
global vel_bas_type  pre_bas_type  str_bas_type  Grv_bas_type
global quad_rul num

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%
npts = size(xy_pts,1) ;
x = xy_pts(:,1)';
y = xy_pts(:,2)';
```

localmat is a vector of values

```
%'num' is used to change between the two problems in the homework
if num == 1
    localmat = 2*pi*x.*y+10;
else
    localmat = 2*(x.*sin(2*pi*x.*y)+x)+3*(sin(2*pi*x.*y)
+2*pi*x.*y.*cos(2*pi*x.*y)+1)+4*pi^2*x.*y.^2.*sin(2*pi*x.*y)+ ...
                4*pi^2*x.^3.*sin(2*pi*x.*y)-4*pi*y.*cos(2*pi*x.*y)
+4*pi*x.^2.*cos(2*pi*x.*y);
end
```

Published with MATLAB® R2023b