```
// example 4 : poisson-buildmesh.edp   [slide page 21]
// error estimation for finite element solution of Poisson equation
// with P2 element
// Delaunay triangluation is realized by buildmesh
// for tutorial by Japan SIAM, Tokyo, 11-12 Feb.2016, Atsushi Suzuki

int n1, n2, n3;
n1 = 20;
n2 = n1 * 2;
n3 = n2 * 2;
border bottom(t=0,1){x=t;y=0;   label=1;};
border right(t=0,1) {x=1;y=t;   label=2;};
border top(t=0,1)   {x=1-t;y=1; label=3;};
border left(t=0,1)  {x=0;y=1-t; label=4;};

mesh Th1=buildmesh(bottom(n1)+right(n1)+top(n1)+left(n1));
mesh Th2=buildmesh(bottom(n2)+right(n2)+top(n2)+left(n2));
mesh Th3=buildmesh(bottom(n3)+right(n3)+top(n3)+left(n3));
plot(Th1);
plot(Th2);
plot(Th3);
fespace Vh1(Th1,P2);
fespace Vh2(Th2,P2);
fespace Vh3(Th3,P2);

Vh1 u1,v1,w1;
Vh2 u2,v2,w2;
Vh3 u3,v3,w3;
real err1, err2, err3, hh1, hh2, hh3;

func f = 5.0/4.0 * pi * pi * sin(pi * x) * sin(pi * y / 2.0);
func h = (-pi)/2.0 * sin(pi * x);
func g = sin(pi * x) * sin(pi * y / 2.0);
// for error estimation
func sol = sin(pi * x) * sin(pi * y / 2.0);
func solx = pi * cos(pi * x) * sin(pi * y / 2.0);
func soly = (pi / 2.0) * sin(pi * x) * cos(pi * y / 2.0);

solve poisson1(u1,v1) =
int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
-  int2d(Th1)( f*v1 )
-  int1d(Th1,1) ( h*v1 )
+  on(2,3,4,u1=g);

solve poisson2(u2,v2) =
int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
-  int2d(Th2)( f*v2 )
-  int1d(Th2,1) ( h*v2 )
+  on(2,3,4,u2=g);

solve poisson3(u3,v3) =
int2d(Th3)( dx(u3)*dx(v3)+dy(u3)*dy(v3) )
-  int2d(Th3)( f*v3 )
-  int1d(Th3,1) ( h*v3 )
+  on(2,3,4,u3=g);

fespace Vh10(Th1,P0); // to measure mesh size
fespace Vh20(Th2,P0);
fespace Vh30(Th3,P0);

Vh10 h1 = hTriangle;
Vh20 h2 = hTriangle;
Vh30 h3 = hTriangle;
hh1 = h1[].sum / h1[].n;
hh2 = h2[].sum / h2[].n;
hh3 = h3[].sum / h3[].n;
cout << "h1 max " << h1[].max << " h1 min " << h1[].min
     << " mean = " << hh1 << endl;
cout << "h1 max " << h2[].max << " h2 min " << h2[].min
     << " mean = " << hh2 << endl;
```

```
cout << "h1 max " << h3[].max << " h3 min " << h3[].min
     << " mean = " << hh3 << endl;

// int2d uses qf5pT : 5th order integration quadrature
err1 = int2d(Th1)( (dx(u1) - solx) * (dx(u1) - solx) +
                   (dy(u1) - soly) * (dy(u1) - soly) +
                   (u1 - sol) * (u1 - sol));
err1 = sqrt(err1);

err2 = int2d(Th2)( (dx(u2) - solx) * (dx(u2) - solx) +
                   (dy(u2) - soly) * (dy(u2) - soly) +
                   (u2 - sol) * (u2 - sol));
err2 = sqrt(err2);

err3 = int2d(Th3)( (dx(u3) - solx) * (dx(u3) - solx) +
                   (dy(u3) - soly) * (dy(u3) - soly) +
                   (u3 - sol) * (u3 - sol));
err3 = sqrt(err3);

cout << "DOF=" << u1[].n << "\t h=" << hh1 << " err-H1=" << err1 << endl;
cout << "DOF=" << u2[].n << "\t h=" << hh2 << " err-H1=" << err2 << endl;
cout << "DOF=" << u3[].n << "\t h=" << hh3 << " err-H1=" << err3 << endl;
cout << "O(h^2)=" << log(err1/err2)/log(hh1/hh2) << endl;
cout << "O(h^2)=" << log(err2/err3)/log(hh2/hh3) << endl;
```