For the finite element method it is just the opposite. If we have the Neumann boundary condition,

$$\nabla u \cdot \mathbf{n} = 0 \tag{4.114}$$

this is satisfied automatically within the Galerkin and variational formulations. This can be seen by using Green's divergence theorem

$$\int_\Omega \nabla \cdot \mathbf{A} dx = \int_\Gamma \mathbf{A} \cdot \mathbf{n} \, dS \tag{4.115}$$

and applying it to the left hand side of the Galerkin finite element formulation

$$\begin{aligned} \int_\Omega \nabla v \cdot \nabla w \, d\Omega \equiv \int_\Omega [\frac{\partial v}{\partial x_1}\frac{\partial w}{\partial x_1} + \frac{\partial v}{\partial x_2}\frac{\partial w}{\partial x_2}] \, d\Omega \\ = \int_\Gamma [v\frac{\partial w}{\partial x_1} n_1 + v\frac{\partial w}{\partial x_2} n_2] \, dS - \int_\Omega v[\frac{\partial^2 w}{\partial x_1^2} + \frac{\partial^2 w}{\partial x_2^2}] \, d\Omega \\ = \int_\Gamma v\frac{\partial w}{\partial n} \, dS - \int_\Omega v\nabla^2 w \, d\Omega \end{aligned} \tag{4.116}$$

If we multiply our original differential equation, $\nabla^2 u = -f$ by an arbitrary test function and integrate, we obtain,

$$(f, v) = -\int_\Omega (\nabla^2 u) v \, d\Omega = -\int_\Gamma \frac{\partial u}{\partial n} v \, dS + \int_\Omega \nabla u \cdot \nabla v \, d\Omega = a(u, v) \tag{4.117}$$

where the boundary integral term, $\frac{\partial u}{\partial n}$ vansishes and we obtain the standard Galerkin finite element formulation.

To apply the Dirichlet condition, we have to work a bit harder. To apply the Dirichlet boundary condition directly, one usually modifies the (a_{ij}) matrix and f_i vector such that one can use standard linear system solvers. This is accomplished by implementing the following steps: Assume we know the ith value of u_i,

1. Subtract from the jth member of the r.h.s. the product of a_{ji} and the known value of u_i (call it $\bar{u}_i$, this yields the new right hand side, $\hat{f}_j = f_j - a_{ji}\bar{u}_i$.

2. Zero the ith row and column of A: $\hat{a}_{ij} = \hat{a}_{ji} = 0$.

3. Assign $\hat{a}_{ii} = 1$.

4. Set the jth member of the r.h.s. equal to $\bar{u}_i$.

5. Continue for each Dirichlet condition.

6. Solve the augmented system, $\hat{A}u = \hat{f}$.

Consider, for example, the 3×3 system,

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}. \tag{4.118}$$

where u_3 is known, $\bar{u}_3$. Following the steps enumerated above, we obtain the following augmented system,

$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 - a_{13}\bar{u}_3 \\ f_2 - a_{23}\bar{u}_3 \\ \bar{u}_3 \end{pmatrix}. \tag{4.119}$$

Solving this system using a standard solver for $Ax = b$ yields the solution set, $(u_1, u_2, \bar{u}_3)$.

As one might imagine, the process of preparing the augmented system is somewhat time consuming if one has a large number of Dirichlet boundary conditions. Another, simpler, way of applying the Dirichlet boundary condition is to use a *penalty method.* To explain this method, let's assume that node 5 is known on our boundary, $u_5 = \bar{u}_5$. From the system of linear equations, we have,

$$a_{51}u_1 + a_{52}u_2 + \ldots + a_{55}u_5 + \ldots + a_{5N}u_N = f_5 \tag{4.120}$$

To enforce the boundary condition at node 5 using the penalty method, we would add the term $\hat{P}u_5$ on the left side and the term $\hat{P}\bar{u}_5$ on the right side,

$$a_{51}u_1 + a_{52}u_2 + \ldots + (a_{55} + \hat{P})u_5 + \ldots + a_{5N}u_N = f_5 + \hat{P}\bar{u}_5 \tag{4.121}$$

then if the number $\hat{P}$ is considerably larger than the terms, $a_{51}, a_{52}, \ldots, a_{5N}$ and f_5, then the Dirichlet boundary condition is approximately satisfied. For example, if $\hat{P}$ is chosen to be,

$$\hat{P} = N \cdot 10^p \cdot \max(a_{ij}) \tag{4.122}$$

then the node 5 has the value,

$$u_5 = \bar{u}_5 + O(10^{-p}). \tag{4.123}$$

Thus $u_5 = \bar{u}_5$ is satisfied to withing an error 10^{-p}. A word of caution at this point. While this method usually works well when using direct solvers, it can lead to diseasterous results when using iterative methods as the penalty method can cause the system to become ill-conditioned.

4.7 A 2D Example

Consider the example of solving the Poisson equation in a 2D rectangular domain.

$$\nabla^2 u(x, y) = -f(x, y) \quad \text{in } \Omega \tag{4.124}$$

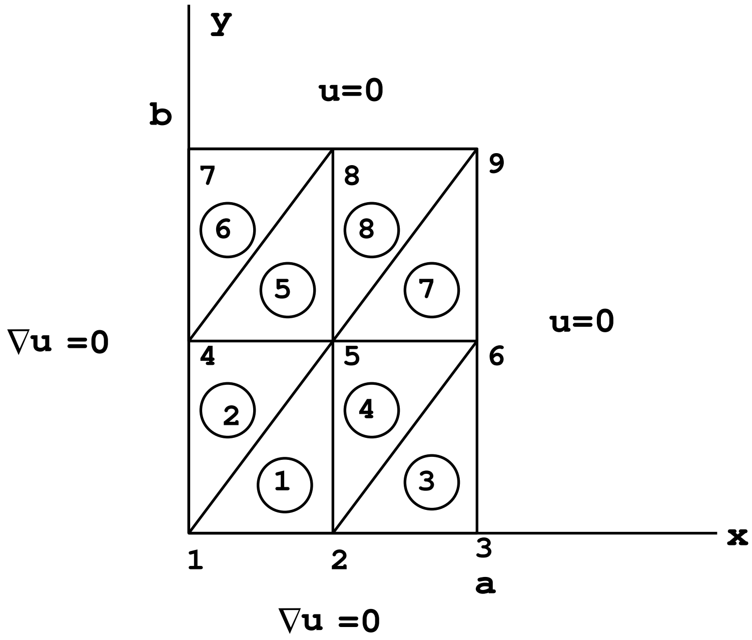

Figure 4.8: Rectangular domain discretized into eight elements.

We disretize the domain into eight triangular elements as shown in figure 4.8. We apply Dirichlet conditions on the top and right hand faces and a zero Neumann condition on the left and bottom faces. Once we've discretized the problem, we then need to construct the global system matrices. Since we are using first-order linear triangles, this means we need to construct

$$a_{ij}^{(n)} = \frac{1}{4A}(b_i^{(n)}b_j^{(n)} + c_i^{(n)}c_j^{(n)}) \tag{4.125}$$

and

$$f_i^{(n)} = \frac{Af}{3} \quad \text{for constant sources.} \tag{4.126}$$

Let's first just look at the elements 1 and 2. The contributions from each element yields

$$\begin{pmatrix} a_{11}^1 + a_{11}^2 & a_{12}^1 & 0 & a_{12}^2 & a_{13}^1 + a_{13}^2 \\ & a_{22}^1 & 0 & 0 & a_{23}^1 \\ & & 0 & 0 & 0 \\ & & & a_{22}^2 & a_{23}^2 \\ & S & & & a_{33}^1 + a_{33}^2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{pmatrix} = - \begin{pmatrix} f_1^1 + f_1^2 \\ f_2^1 \\ 0 \\ f_4^2 \\ f_5^1 + f_5^2 \end{pmatrix}. \tag{4.127}$$

If we look at the entire system for all eight elements (nine nodes), we obtain the following global matrices

$$\begin{pmatrix} (1+2) & (1) & 0 & (2) & (1+2) & 0 & 0 & 0 & 0 \\ & (1+3+4) & (3) & 0 & (1+4) & (3+4) & 0 & 0 & 0 \\ & & (3) & 0 & 0 & (3) & 0 & 0 & 0 \\ & & & (2+5+6) & (2+5) & 0 & (6) & (5+6) & 0 \\ & & & & (1+2+4+5+7+8) & (4+7) & 0 & (5+8) & (7+8) \\ & Sym. & & & & (3+4+7) & 0 & & (7) \\ & & & & & & (6) & (6) & 0 \\ & & & & & & & (5+6+8) & (8) \\ & & & & & & & & (7+8) \end{pmatrix} \tag{4.128}$$

and the right hand side is

$$-\begin{pmatrix} (1+2) \\ (1+4+3) \\ (3) \\ (2+5+6) \\ (1+2+4+5+7+8) \\ (3+4+7) \\ (7) \\ (5+6+8) \\ (7+8) \end{pmatrix} \tag{4.129}$$

where the contributions are denoted by element numbers. Now we need to apply the boundary conditions. We have the Dirichlet condition of $u = 0$ on nodes $3, 6, 7, 8, 9$ and the Neumann condition $\partial u / \partial n = 0$ on nodes $1, 2, 4$. As we noted before, the zero-flux boundary condition corresponding to the Neumann condition is automatically satisfied, so we don't have to do anything. Therefore we need to enforce the Dirichlet condition. We proceed as illustrated in the 3×3 example. Applying the boundary conditions, we obtain the following system

$$\begin{pmatrix} 5 & -4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 10 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ & & & 10 & -8 & 0 & 0 & 0 & 0 \\ & & & & 20 & 0 & 0 & 0 & 0 \\ & Sym. & & & & 1 & 0 & 0 & 0 \\ & & & & & & 1 & 0 & 0 \\ & & & & & & & 1 & 0 \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{pmatrix} = -f\frac{a^3}{3} \begin{pmatrix} 2 \\ 3 \\ 0 \\ 3 \\ 6 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{4.130}$$

where we have let $b = 2a$. Now we just have to solve the system of equations to obtain our solution vector u.

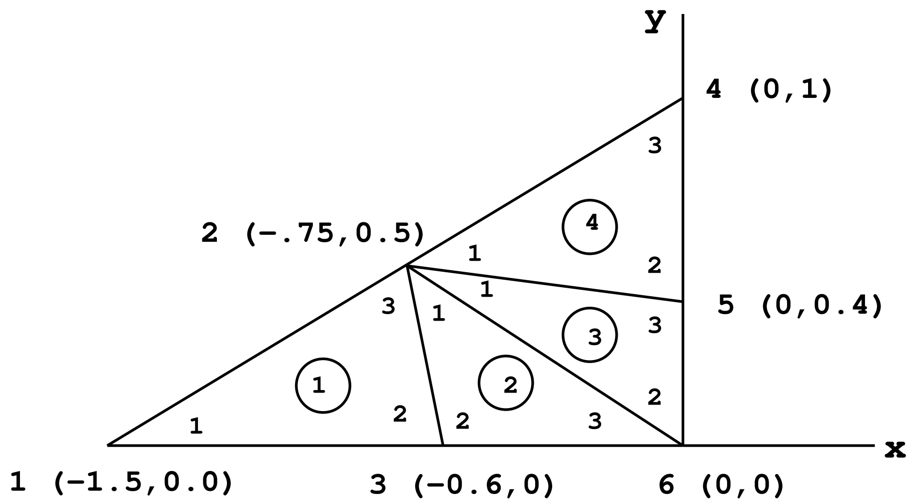

Figure 4.9: The Triangular Solution Domain.

4.8 Another 2D Example Using Triangular elements

Let's look at another example of solving Poisson's equation, but this time we'll work everything out in detail [27]. Consider following Poisson problem within the triangular region shown in figure 4.9 In the figure, the outside node numbers represent the global numbering scheme, while the inside numbers represent the local element numbering scheme. We wish to solve

$$\nabla^2 u = 2 \tag{4.131}$$

with the Dirichlet boundary condition given along the hypotenuse as

$$u = 5 - 1.5y + 2.5y^2 \tag{4.132}$$

such that at node 1, $u = 5$, at node 2, $u = 4.875$, and at node 4, $u = 6.0$. At the two faces, we'll impose the inhomogeneous Neumann boundary conditions

$$-\frac{\partial u}{\partial y} = -x \quad \text{at } y = 0 \tag{4.133}$$

and

$$\frac{\partial u}{\partial x} = y \quad \text{at } x = 0 \tag{4.134}$$

Note that the discretization shown in figure 4.9 is only for illustrative purposes - it is not necessarily the best element configuration. We now procede to calculate the element matrices. For element 1, the global node numbers $1, 3, 2$ correspond to the local element node numbers of $1, 2, 3$ respectively. Notice that we will always choose the local node numbers such that they are numbered in a counter-clockwise direction. The coordinates of our nodes are $(-1.5, 0)$,$(-0.6, 0)$, and $(-0.75, 0.5)$. To calculate the element matrices, we first need to calculate the geometric coefficients a_i, b_i, c_i. For element one we have:

$$\begin{array}{rrr} a_1 = x_2y_3 - x_3y_2 = -.3 & a_2 = x_3y_1 - x_1y_3 = .75 & a_3 = x_1y_2 - x_2y_1 = 0 \\ b_1 = y_2 - y_3 = -0.5 & b_2 = y_3 - y_1 = 0.5 & b_3 = y_1 - y_2 = 0 \\ c_1 = x_3 - x_2 = -0.15 & c_2 = x_1 - x_3 = -0.75 & c_3 = x_2 - x_1 = 0.90 \end{array} \tag{4.135}$$

Therefore, the area of element 1 is

$$A = \frac{1}{2}(a_1 + a_2 + a_3) = 0.225 \tag{4.136}$$

The element matrix for element 1 is therefore

$$a_{ij}^{(1)} = \begin{pmatrix} 0.30278 & -0.15278 & -0.15000 \\ & 0.90278 & -0.75000 \\ & & 0.90000 \end{pmatrix} \tag{4.137}$$

Since we have a non-zero Neumann condition, we have to add the condition onto the calculation of the right hand side

$$f_i^{(1)} = -\int_{A_1} 2\Phi_i(x, y)dxdy + \int_{-1.5}^{-0.6} (-x)\Phi(x, 0)dx \tag{4.138}$$

where A_1 is the area of element one. Integrating, we obtain

$$f_i^{(1)} = \begin{pmatrix} 0.3900 \\ 0.2550 \\ -0.1500 \end{pmatrix} \tag{4.139}$$

For element two, the local node numbers, $1, 2, 3$, correspond to the global nodes $2, 3, 6$. From the coordinate information, we obtain

$$\begin{array}{rrr} a_1 = x_2y_3 - x_3y_2 = 0.0 & a_2 = x_3y_1 - x_1y_3 = 0.0 & a_3 = x_1y_2 - x_2y_1 = .30 \\ b_1 = y_2 - y_3 = 0.0 & b_2 = y_3 - y_1 = -0.5 & b_3 = y_1 - y_2 = 0.5 \\ c_1 = x_3 - x_2 = 0.6 & c_2 = x_1 - x_3 = -0.75 & c_3 = x_2 - x_1 = 0.15 \end{array} \tag{4.140}$$

Therefore, the area of element 2 is

$$A = \frac{1}{2}(a_1 + a_2 + a_3) = 0.15 \tag{4.141}$$

The element matrix for element 2 is therefore

$$a_{ij}^{(2)} = \begin{pmatrix} 0.6000 & -0.7500 & 0.15000 \\ & 1.35417 & -0.60417 \\ & & 0.45417 \end{pmatrix} \tag{4.142}$$

Once again we have a non-zero Neumann condition and have to add the condition onto the calculation of the right hand side

$$f_i^{(2)} = -\int_{A_2} 2\Phi_i(x,y)dxdy + \int_{-0.6}^{0.0}(-x)\Phi(x,0)dx \tag{4.143}$$

where A_2 is the area of element two. Integrating, we obtain

$$f_i^{(2)} = \begin{pmatrix} -0.10 \\ 0.02 \\ 0.04 \end{pmatrix} \tag{4.144}$$

Following along the same lines, we obtain the local element matrix and vector for element three

$$a_{ij}^{(3)} = \begin{pmatrix} 0.26667 & 0.06667 & -0.33333 \\ & 0.95417 & -1.02083 \\ & & 1.35417 \end{pmatrix} \tag{4.145}$$

$$f_i^{(3)} = \begin{pmatrix} -0.10 \\ -0.0730 \\ -0.0470 \end{pmatrix} \tag{4.146}$$

and for element four

$$a_{ij}^{(4)} = \begin{pmatrix} 0.4000 & -.33333 & -0.06667 \\ & 0.90278 & -0.56944 \\ & & 0.63611 \end{pmatrix} \tag{4.147}$$

$$f_i^{(4)} = \begin{pmatrix} -0.15 \\ 0.03 \\ 0.09 \end{pmatrix} \tag{4.148}$$

Now we need to create the global matrix and vector by adding in the contributions from each of the elements. The global matrix will have contributions as follows

$$a_{ij}^{(n)} = \begin{pmatrix} a_{11}^1 & a_{13}^1 & a_{12}^1 & 0 & 0 & 0 \\ & a_{11}^2 + a_{33}^1 + a_{11}^3 + a_{11}^4 & a_{23}^1 + a_{12}^2 & a_{13}^4 & a_{13}^3 + a_{12}^4 & a_{13}^2 + a_{12}^3 \\ & & a_{22}^1 & 0 & 0 & a_{23}^2 \\ & & & a_{33}^4 & a_{23}^4 & 0 \\ & & & & a_{33}^3 + a_{22}^4 & a_{23}^3 \\ & & & & & a_{33}^2 + a_{22}^3 \end{pmatrix} \tag{4.149}$$

which yields

$$a_{ij}^{(n)} = \begin{pmatrix} 0.30278 & -.15000 & -0.15278 & 0 & 0 & 0 \\ & 2.1667 & -1.5000 & -0.667 & -0.6667 & 0.21667 \\ & & 2.25695 & 0 & 0 & -0.60417 \\ & & & 0.63611 & -0.56944 & 0 \\ & & & & 2.25695 & -1.02083 \\ & & & & & 1.40833 \end{pmatrix} \tag{4.150}$$

The contributions to the right hand side are

$$f_i^{(n)} = \begin{pmatrix} f_1^1 \\ f_3^1 + f_1^2 + f_1^3 + f_1^4 \\ f_2^1 + f_2^2 \\ f_3^4 \\ f_3^3 + f_2^4 \\ f_3^2 + f_3^3 \end{pmatrix} \tag{4.151}$$

which yields

$$f_i^{(n)} = \begin{pmatrix} 0.39000 \\ -0.50000 \\ 0.27200 \\ 0.09000 \\ -0.01700 \\ -0.11300 \end{pmatrix} \tag{4.152}$$

We now need to enforce the Dirichlet boundary conditions on nodes $1, 2, 4$ and then solve the system $Au = f$. Since this is such a small system, one way to proceed is to put the known values of u in for the nodes $1, 2, 4$ and then multiply out the system. We would be left with three equations with three unknowns (at nodes $3, 5, 6$). Another way to proceed is to apply the Dirichlet conditions directly and solving the augmented system. Still another way to proceed is to utilize the penalty method. Let's apply the boundary conditions directly. To impose the boundary condition at node 1, $u_1 = 5.0$ we need to modify the rhs as follows:

$$\hat{f}_i^{(n)} = \begin{pmatrix} 5.0 \\ -0.5 - (-0.15) \times 5.00 \\ 0.27200 - (-0.1528) \times 5.00 \\ 0.09000 - 0.0 \times 5.00 \\ -0.01700 - 0.0 \times 5.00 \\ -0.11300 - 0.00 \times 5.00 \end{pmatrix} \tag{4.153}$$

which yields

$$\hat{f}_i^{(n)} = \begin{pmatrix} 5.0 \\ 0.2500 \\ 1.0370 \\ 0.090 \\ -0.017 \\ -0.113 \end{pmatrix} \tag{4.154}$$

We then set the first row and the first column of A equal to 0, except for a_{11}, which is set equal to 1. We continue the process for the other two boundary conditions and end up with the following augmented system:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 0 \\ & & 2.257 & 0 & 0 & -0.6042 \\ & & & 1 & 0 & 0 \\ & Sym. & & & 2.257 & -1.021 \\ & & & & & 1.408 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{pmatrix} = \begin{pmatrix} 5.000 \\ 4.875 \\ 8.35 \\ 6.00 \\ 6.647 \\ -1.17 \end{pmatrix} \tag{4.155}$$

Now that we have applied the boundary conditions, we just need to solve our system. The solution vector from the FE approximation is

$$\mathbf{u} = \begin{pmatrix} 5.000 \\ 4.875 \\ 5.09 \\ 6.000 \\ 5.29 \\ 5.19 \end{pmatrix} \tag{4.156}$$

and the exact solution is

$$\mathbf{u} = \begin{pmatrix} 5.000 \\ 4.875 \\ 5.000 \\ 6.000 \\ 5.160 \\ 5.000 \end{pmatrix}. \tag{4.157}$$

Thus the finite element solution is quite close to the exact solution, even with such a course discretization.

4.9 Higher Order Finite Elements

Now that we have a feel for the finite element method and how it works for linear elements, let's discuss applying the Galerkin finite element method for higher order elements. While there are many different schemes for generating heirarchies of higher order elements, the simplist is based on the set of complete polynomials. The complete polynimals can be represented by Pascal's triangle as shown in figure 4.10.

The first three higher-order C^0 triangular elements are shown in figure 4.11. The trial functions can just be read off of the rows of Pasal's triangle. For example, the C^0 quadratic element in (a) has the form:

$$\tilde{u} = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 x^2 + \alpha_5 xy + \alpha_6 y^2, \tag{4.158}$$

while the C^0 cubic element in (b) has the form:

$$\tilde{u} = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 x^2 + \alpha_5 xy + \alpha_6 y^2 + \alpha_7 x^3 + \alpha_8 x^2 y + \alpha_9 xy^2 + \alpha_{10} y^3. \tag{4.159}$$

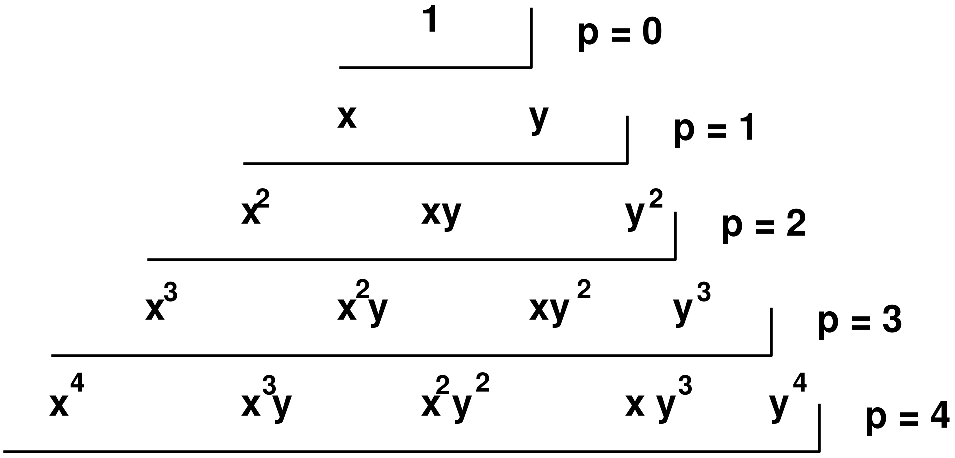

Figure 4.10: Pascal's triangle: A two-dimensional polynomical is complete to degree p if it contains all terms in the first p+1 rows of the triangle.

In the discussion of linear elements, we utilized the co-called direct approach in constructing the element basis functions. The direct approach is restricted to straight-sided elements. Since curved elements are useful in practice, principally for modeling curved boundaries, we will use a parametric mapping (in particular, an isoparametric mapping). Let's develop the shape (basis) functions and mappings for a C^0 isoparametric quadratic element.

Figure 4.12 shows the so-called parent element in ξ, η space next to the curved-sided real element in x, y space. What we need to do is develop the shape (basis) functions on the simple unit parent element, then we'll map these shape functions onto the real element. The sides of the parent element are straight, even though the real elements may be curved. While the parent element may have any triangular shape, choosing the parent element to be a unit right triangle makes life easier on us.

Let's develop the parent basis functions using the interpolation property, $\Phi_i(\xi_j), \eta_j = \delta_{ij}$. For a two-dimensional quadratic function, we would have for node 1:

$$\Phi_1(\xi, \eta) = \alpha_1 + \alpha_2\xi + \alpha_3\eta + \alpha_4\xi^2 + \alpha_5\xi\eta + \alpha_6\eta^2 \tag{4.160}$$

The interpolation property requires that

$$\Phi_1(\xi_1, \eta_1) = 1 \tag{4.161}$$

and

$$\Phi_1(\xi_j, \eta_j) = 0, \quad j = 2, 3, 4, 5, 6. \tag{4.162}$$

Apply these conditions results in a set of six equations with six unknowns:

$$\begin{array}{llllllr}
\alpha_1 & & & & & & = 1 \\
\alpha_1 & +\alpha_2 & & +\alpha_4 & & & = 0 \\
\alpha_1 & & \alpha_3 & & & +\alpha_6 & = 0 \\
\alpha_1 & +\frac{1}{2}\alpha_2 & & +\frac{1}{4}\alpha_4 & & & = 0 \\
\alpha_1 & +\frac{1}{2}\alpha_2 & +\frac{1}{2}\alpha_3 & +\frac{1}{4}\alpha_4 & +\frac{1}{4}\alpha_5 & +\frac{1}{4}\alpha_6 & = 0 \\
\alpha_1 & & \frac{1}{2}\alpha_3 & & & \frac{1}{4}\alpha_6 & = 0
\end{array} \tag{4.163}$$

Solving this system yields

$$\Phi_1(\xi, \eta) = [1 - (\xi + \eta)][1 - 2(\xi + \eta)]. \tag{4.164}$$

Similarly, we can find relationships for the other five nodes in the parent element:

$$\begin{aligned}
\Phi_2(\xi, \eta) &= \xi(2\xi - 1) \\
\Phi_3(\xi, \eta) &= \eta(2\eta - 1) \\
\Phi_4(\xi, \eta) &= 4\xi[1 - (\xi + \eta) \\
\Phi_5(\xi, \eta) &= 4\xi\eta \\
\Phi_6(\xi, \eta) &= 4\eta[1 - (\xi + \eta)
\end{aligned} \tag{4.165}$$

Now that we have a description of the basis functions for the parent element, we need to map them to the real element space. There are many ways to perform this mapping. One that is common for its simplicity and effectiveness is to use an isoparametric transformation. For an isoparametric transformation, the mapping functions are chosen to be the same as the shape functions. For the quadratic element, we have the mapping

$$x = \sum_{k=1}^{6} x_k^e \Phi_k(\xi, \eta) \tag{4.166}$$

and

$$y = \sum_{k=1}^{6} y_k^e \Phi_k(\xi, \eta) \tag{4.167}$$

where the $k = 1, \ldots, 6$ are the six nodes of the (e)th real element. The parent shape functions are transformed to each of the real elements, resulting in shape functions being defined on each real element.

While higher order elements have the property that they can conform to curved boundaries, we should be careful not to allow the elements to become *too* distorted. A general rule for an isoparametric element is that it will be acceptable if and only if its mapping from the parent element is 1 to 1, i.e. each point in the parent element maps onto one point in the real element and vice versa.

The analytical test for the acceptability of an element is that the determinant of the Jacobian is greater than zero everywhere in the element.

$$|J^{(e)}(\xi,\eta)| > 0 \ \text{ everywhere in the element} \tag{4.168}$$

where $|J^{(e)}(\xi,\eta)|$ is the Jacobian, which is the determinant of the Jacobian matrix. The Jacobian matrix is defined as:

$$J^{(e)}(\xi,\eta) = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \tag{4.169}$$

or

$$|J^{(e)}(\xi,\eta)| = \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \xi}\frac{\partial x}{\partial \eta} \tag{4.170}$$

The derivatives of the coordinate transformation are obtained via the isoparametric mapping, i.e.

$$J^{(e)}_{11}(\xi,\eta) = \frac{\partial x}{\partial \xi} = \sum_{i=1}^{6} x^{(e)}_k \frac{\partial \Phi_k(\xi,\eta)}{\partial \xi} \tag{4.171}$$

$$J^{(e)}_{12}(\xi,\eta) = \frac{\partial y}{\partial \xi} = \sum_{i=1}^{6} y^{(e)}_k \frac{\partial \Phi_k(\xi,\eta)}{\partial \xi} \tag{4.172}$$

$$J^{(e)}_{21}(\xi,\eta) = \frac{\partial x}{\partial \eta} = \sum_{i=1}^{6} x^{(e)}_k \frac{\partial \Phi_k(\xi,\eta)}{\partial \eta} \tag{4.173}$$

$$J^{(e)}_{22}(\xi,\eta) = \frac{\partial y}{\partial \eta} = \sum_{i=1}^{6} y^{(e)}_k \frac{\partial \Phi_k(\xi,\eta)}{\partial \eta} \tag{4.174}$$

Using the shape functions we derived for the parent quadratic element, one can take the derivaties, for example

$$\frac{\partial \Phi_1(\xi,\eta)}{\partial \xi} = -3 + 4\xi + 4\eta \tag{4.175}$$

and

$$\frac{\partial \Phi_6(\xi,\eta)}{\partial \eta} = (1 - \xi - 2\eta) \tag{4.176}$$

Before writing down expressions for the the element stiffness and load matrices, we need to make a choice as to how the nodes will be layed out on the element. The simplest choice is to have nodes at the vertices and the midpoints of the edges. This is not the only choice, but it is a convenient one and makes the mesh generation fairly straightforward (in that if you can create a mesh of linear triangles, it is easy to extend them to quadratic). If one uses this scheme, the one can show that the determinant of the Jacobian matrix is every great than zero and is in fact, equal to twice the area of the element.

Since the real shape functions may possibly be distorted, we cannot use closed form integral formulas and thus must use numerical approximations (via quadrature) instead. We first transform the integrals over the real element to integrals over the parent element, and then use a quadrature formula on the transformed integral. The element stiffness and load integrals are:

$$a_{ij}^{(e)} = \int_0^1 \int_0^{1-\eta} \frac{\partial \Phi_i^{(e)}}{\partial x} \frac{\partial \Phi_j^{(e)}}{\partial x} |J^{(e)}| \, d\xi d\eta + \int_0^1 \int_0^{1-\eta} \frac{\partial \Phi_i^{(e)}}{\partial y} \frac{\partial \Phi_j^{(e)}}{\partial y} |J^{(e)}| \, d\xi d\eta \tag{4.177}$$

and

$$f_{ij}^{(e)} = \int_0^1 \int_0^{1-\eta} f \Phi_i^{(e)} |J^{(e)}| \, d\xi d\eta \tag{4.178}$$

The quadrature formulas will require evaluating the integrands at various quadrature points (ξ_l, η_l). Table 4.9 shows a few examples of Gauss points and weights for triangles. For a more complete tabulation, see [28].

Order	Points	Gauss Points (ξ_{nl}, η_{nl})	Weights
Linear	1	$\xi_{11} = \frac{1}{3}$, $\eta_{11} = \frac{1}{3}$	1
Quadratic	3	$\xi_{31} = \frac{1}{6}$, $\xi_{31} = \frac{1}{6}$	1/3
		$\xi_{32} = \frac{2}{3}$, $\xi_{32} = \frac{1}{6}$	1/3
		$\xi_{33} = \frac{1}{6}$, $\xi_{33} = \frac{2}{3}$	1/3

Table 4.1: Gauss points and weights for Gauss quadrature rules on triangles.

The stiffness and load integrals can now be written as

$$a_{ij}^{(e)} = \frac{1}{2} \sum_1^n w_{nl} \left[\frac{\partial \Phi_i^{(e)}}{\partial x} \frac{\partial \Phi_j^{(e)}}{\partial x} |J^{(e)}| + \frac{\partial \Phi_i^{(e)}}{\partial y} \frac{\partial \Phi_j^{(e)}}{\partial y} |J^{(e)}| \right]_{(\xi_{nl}, \eta_{nl})} \tag{4.179}$$

and

$$f_{ij}^{(e)} = \frac{1}{2} \sum_1^n w_{nl} \left[f \Phi_i^{(e)} |J^{(e)}| \right]_{(\xi_{nl}, \eta_{nl})} \tag{4.180}$$

where all the terms inside the brackets can be evaluated at the Gauss points (ξ_{nl}, η_{nl}).

4.10 Renumbering and Storage Schemes

We've just discussed that the finite element approximation of an elliptic problem yields a symmetric, positive definite, sparse global matrix A. Now let's look at the issue of the effects of these properties when we try and solve the results global linear system, $A\xi = b$. Before doing any optimization what so ever, we have an $N \times N$ linear system to solve, where N represents the number of unknowns (the values at the nodes we are trying to determine). Let's say we have a system of 10,000 unknowns (which is pretty moderate for a 2D or 3D finite element calculation). If we store the entire system, we would need $10,000^2 \times (4 \text{ or } 8) = 400 \text{ or } 800Mb$. If we note that the system is symetric such that we would only store the upper or lower trianglular section, we save quite a bit, but it is still on the order of N^2, namely, $((10,000+1) \times 10,000)/2 =\approx 200 \text{ or } 400Mb$, which is still pretty large despite the savings over the full system. The *real* savings comes by exploiting the sparseness. In general, there are two basic schemes to reduce the necessary storage requirements. The two methods are *renumbering* and *sparse storage* methods. The former method, aimed primarily at direct methods, works by minimizing the bandwidth of the matrix such that it can be stored in an $N \times M$ rectangular matrix, where M is the size of the bandwidth. In sparse storage schemes, the bandwidth remains the same, but an algorithm is implemented that only stores nonzero elements along with pointers back to the positions of the elements.

To calculate the bandwidth, one can use the following algorithm:

Algorithm FE 4: Band(vert, N_{el}, Band_width)
Function: Computes the bandwidth of a matrix.
Output: Band_width
Inputs: N_{el}, the total number of finite elements; **vert** is the array of triangle connectivities. This only uses the element connectivities and can thus be done in the preprocessing stage.

Begin

```
    Band_width = 0
    For N ← 1 to N_el Do
        For i ← 1 to 3 Do
            vertI ← vertI (i,N)
            For k ← 1 to 3 Do
                vertJ ← vertJ(k,N)
                vertIJ ← vertI - vertJ +1
                If vertIJ > Band_width
                Then Band_width ← vertIJ
```

End

4.11 Renumbering Schemes

The basic idea is a simple one. Defining D_i as the largest distance between the diagonal term of row i and any other term j on that same row, then to minimize the storage, we would minimize,

$$B = \sum_i D_i \qquad \text{where } D_i = \max[i - j] \tag{4.181}$$

The most popular way of minimzing the bandwidth was put forth in the form of a simple and effective (thus its popularity) algorithm in a paper by Cuthill and McKee [29]. The algorithm is outlined in four steps:

1. Search for the connectivities between variables (nodes). This means to create a "table" which includes the number of connections each node has and also lists the connections to other variables (nodes).
2. Start with the least connected variable and call it 1, then the least connected of those connected to what you've called 1 is given the next value, 2. The other variable connected to what you've changed to 1 becomes 3 and so on.
3. Now continue this process for new node 2 and its connections.
4. Continue this process until all the nodes are renumbered.

4.12 Cuthill-McKee bandwidth reduction example

Let's assume we have the following global matrix from applying the finite element method to approximating an elliptic boundary value problem.

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 10 & 0 & 16 \\ & 3 & 4 & 0 & 7 & 0 & 0 & 17 \\ & & 5 & 0 & 8 & 0 & 13 & 0 \\ & & & 6 & 0 & 11 & 0 & 18 \\ & Sym. & & & 9 & 0 & 14 & 0 \\ & & & & & 12 & 0 & 19 \\ & & & & & & 15 & 0 \\ & & & & & & & 20 \end{pmatrix} \tag{4.182}$$

The (half) bandwidth for this matrix is 8 as we have an element at the furthest point from the diagonal. If we were to store the entire matrix (including the lower triangular), we would have 64 elements. Assuming double precision reals, this would require 512 bytes.

Let's now apply the Cuthill-McKee bandwidth reduction scheme. First we search for connectivities between variables - we look at both the number of connections and connections to other variables.

Node	Number of Connections	Connections to other nodes
1	3	2,6,8
2	4	1,3,5,8
3	3	2,5,7
4	2	6,8
5	3	2,3,7
6	3	1,4,8
7	2	3,5
8	4	1,2,4,6

We start with the least connected variable (4 or 7) and call it 1, then the least connected of those connected to node 4 (which is node 6) becomes 2. Variable 8, which is the other variable connected to 4, becomes the new node 3. Now we do the same thing to the new node 2 (old node 6). Of its connections $(1,4,8)$, both 4 and 8 are already renumbered. Therefore, the old node 1 takes the new number 4. We then continue until all the nodes are renumbered.

Node	Num. of Conn.	Conn. to other nodes	New number	New Conn.
1	3	2,6,8	4	5,2,3
2	4	1,3,5,8	5	4,6,7,3
3	3	2,5,7	6	5,7,8
4	2	6,8	1	2,3
5	3	2,3,7	7	5,6,8
6	3	1,4,8	2	4,1,3
7	2	3,5	8	6,7
8	4	1,2,4,6	3	4,5,1,2

Our matrix now has the following structure

$$\begin{pmatrix} 1 & 2 & 4 & 0 & 0 & 0 & 0 & 0 \\ & 3 & 5 & 7 & 0 & 0 & 0 & 0 \\ & & 6 & 8 & 10 & 0 & 0 & 0 \\ & & & 9 & 11 & 0 & 0 & 0 \\ & Sym. & & & 12 & 13 & 15 & 0 \\ & & & & & 14 & 16 & 18 \\ & & & & & & 17 & 19 \\ & & & & & & & 20 \end{pmatrix} \tag{4.183}$$

The (half) bandwidth of the matrix has been reduced to 3 instead of 8. Thus we now have to store only a 3×8 matrix instead of an 8×8 matrix such that it only takes 192 bytes instead of 512 bytes.

It was recognized by George [30] that if the Cuthill-McKee numbering is reversed, the bandwidth is often reduced beyond what is given by the direct Cuthill-McKee algorithm. Here reverse is used in the sense that element (i, j) moves to $(n-j+1, n-i+1)$. Lui and Sherman, [31] proved that the reverse Cuthill-McKee algorithm could not give a less efficient numbering and often gave a more efficient renumbering than the direct version.

In the Cuthill-McKee algorithm, one can recognize *level sets* within the renumbering. The first set is the starting node. The second set comprises those nodes connected directly to it. The third set consists of all nodes whose shortest route to the starting node involves just two links, etc. The resulting matrix can be specified as block tridiagonal because the nodes at any level are connected to nodes in no more than three adjacent levels. An optimal renumbering scheme will keep the blocks as small as possible. In graph theory, such a scheme is called a *breadth first search*.

The block structure of the Cuthill-McKee coefficient matrix is fixed by the choice of the starting node - thus the importance of picking a good one. While no one has found a way to find the optimal choice, it is found in practice that nodes in the highest level of a Cuthill-Mckee ordering make good choices for the starting node for a second renumbering. This implies that the algorithm should always be run twice, each time restarting with one of the last numbered nodes to achieve the best ordering.

As an aside (for now), one should calculate the cost of renumbering and balance this against the savings in storage space. It is interesting to note that while this isn't balanced for iterative solvers in a straightforward way, it can be proven that renumbering can save a signficant amount of time when using certain preconditioned conjugate gradient algorithms. This is especially true when the preconditioning methods are employed using approximate inverses having the same profile as the matrix to be solved [32].

Let's now return to the second method for reducing storage costs, that of *sparse storage* methods. As we saw before, using the fact that the matrix is symmetric still lead to a storage requirement which is proportional to N^2. With sparse storage techniques, we can bring down storage requirements down to be proportional to N (at least for first order finite element meshes). One of the first, and still widely used, techniques for efficiently storing global finite element matrices is called the Yale Sparse Storage Scheme. This is one of many row or column based storage techniques. Basically, the idea is to never store a zero and to efficiently point to the non-zero entries. The Yale sparse storage scheme breaks the global matrix into three vector arrays:

- $a(*)$ = a real valued array of non-zero elements. The array is $1 \times NZ$ in length, where NZ is the number of non-zero elements.

- $ja(*)$ = an integer valued array of column numbers. $ja(*)$ is the column number of $a(*)$. The vector is an integer array of length $1 \times NZ$.
- $ia(*)$ = a mapping vector to denote the starting locations of each of the blocks. ($ia(*)$ tells how many non-zero elements there are in each row. The vector is an integer array of length $N + 1$.

If we apply the Yale sparse storage technique to the matrix we used in the Cuthill-McKee example, we obtain the following arrays:

- $a(*) = [1, 2, 10, 16, 3, 4, 7, 17, 5, 8, 13, 6, 11, 18, 9, 14, 12, 19, 15, 20]$
- $ja(*) = [1, 2, 6, 8, 2, 3, 5, 8, 3, 5, 7, 4, 6, 8, 5, 7, 6, 8, 7, 8]$
- $ia(*) = [1, 5, 9, 12, 15, 17, 19, 20, 21]$

The $a(*)$ vector of reals will cost 160 bytes, while the two integer vectors will cost 40 and 18 bytes, giving us a total of 218 bytes. While this costs a little more than the bandwidth reduction scheme for this particular example, in general, the sparse storage methods yield a much greater savings for large system.

For argument sake, let's take first order linear triangles in which the unknowns are at the vertices of the triangles. For numerical reasons, we wish these elements to be as equilateral as possible, thus we could reasonable say there were six elements connected to each node (omitting the boundary points for now). Probabilistically, of these six, three will have smaller numbers than that node and therefore the corresponding coefficients will be three in number. Therefore, including the diagonal, we would probably average about four numbers be row to store. This yields a total of $4N$ numbers that need to be stored - plus numbers which correspond to their location. This would yield us with approximately $4N$ real numbers corresponding to the non-zero elements, $4N$ integers corresponding to their row location, and $4N$ integers corresponding to their column location. Therefore, in our example of $N = 10,000$ we would need $40,000$ real numbers and $80,000$ integers for a total of $480Kb$.

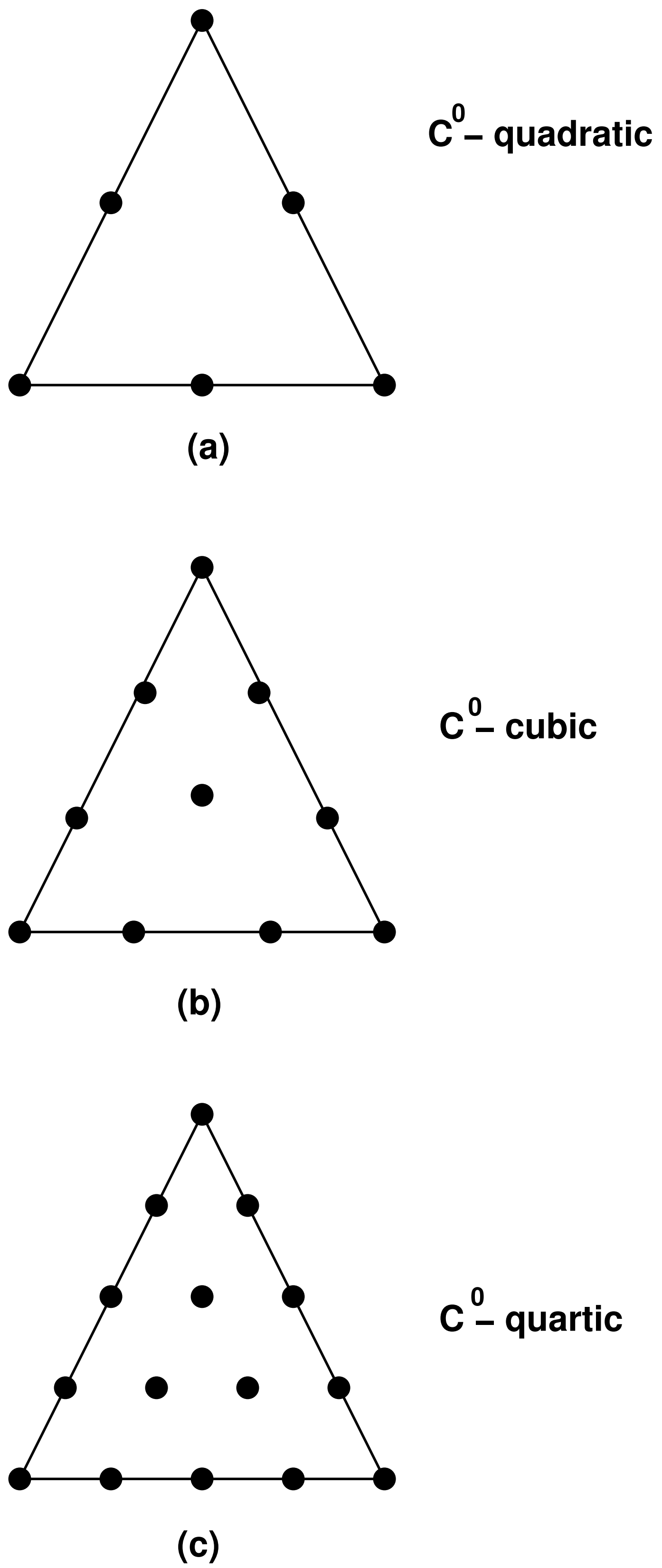

Figure 4.11: The first three higher-order C^0 triangular elements employing complete two-dimensional polynomials.

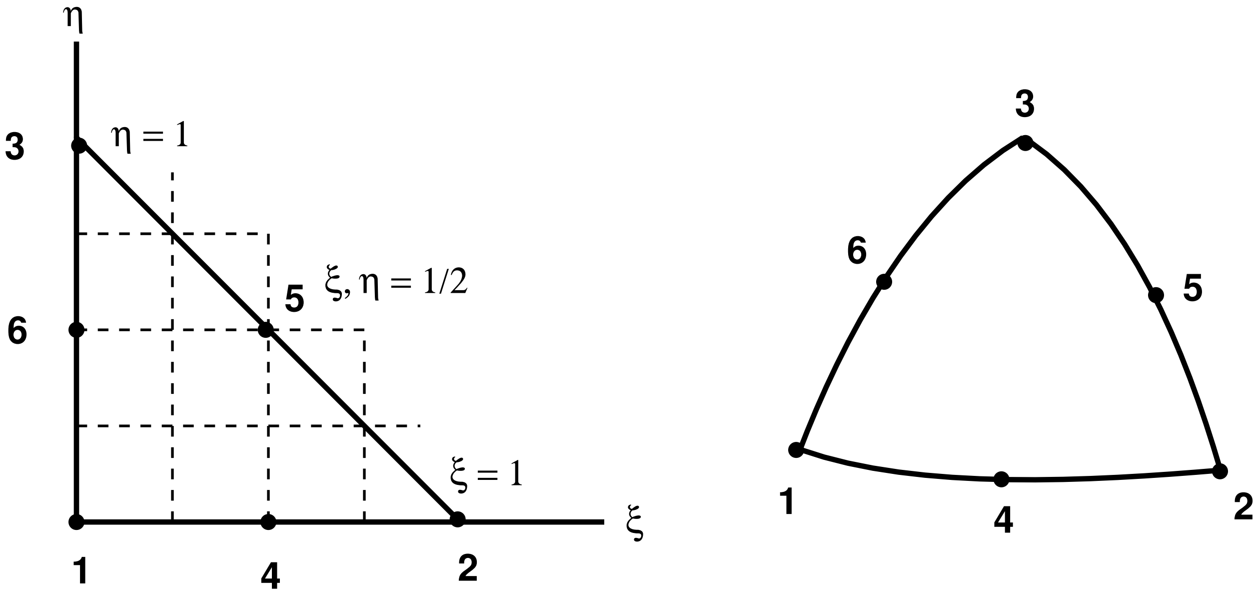

Figure 4.12: The parent element for the six-node quadratic isoparametric triangle and the curved sided real element.