Once more, we defined the test space $\mathcal{V}_h$ as the direction of $\mathcal{S}_h$. Notice, that $\mathcal{V}_h = \tilde{\mathcal{V}}_h$. A basis for $\mathcal{V}_h$ is obtained by considering all basis functions in $\mathcal{W}_h$ that are 0 on $\partial\Omega_D$. To see this, notice that

$$w_h(x) = 0 \text{ for } all\ x \in \partial\Omega_D \iff w^a = 0 \text{ whenever } \mathbf{X}^a \in \partial\Omega_D.$$

Hence, if vertex $\mathbf{X}^a \in \partial\Omega_D$, then $N_a \notin \mathcal{V}_h$. Instead, if $\mathbf{X}^a \notin \partial\Omega_D$, then $N_a \in \mathcal{V}_h$. Once again, we should stop and reflect on the ease by which it is possible to identify a subset of the finite element basis for $\mathcal{W}_h$ to serve as basis for $\mathcal{V}_h$. This is not trivial or even possible with many other bases, as Example 2.7 illustrates.

So, the set of active and constrained indices is

$$\eta_a = \{a \in \{1,\dots,n_V\} \mid \mathbf{X}^a \notin \partial\Omega_D\}$$
$$\eta_g = \{a \in \{1,\dots,n_V\} \mid \mathbf{X}^a \in \partial\Omega_D\}.$$

This definition of $\mathcal{S}_h$ automatically identifies a function $\overline{u}_h \in \mathcal{S}_h$ as

$$\overline{u}_h = \sum_{a\in\eta_g} g(\mathbf{X}^a) N_a.$$

However, in this case the equations that constrained indices should satisfy are already included in the definition of $\mathcal{S}_h$, namely,

$$u^a = \overline{u}^a = g(\mathbf{X}^a) \qquad a \in \eta_g. \tag{2.54}$$

We can assume that a **list of boundary nodes**, those with indices in η_g, is provided by the way the mesh is constructed. The **list of boundary values**, GG, should be constructed from (2.54).

Finally, at this stage we can approximate k and f as piecewise constant over each triangle, so that we count with values k_e and f_e in each triangle.

2.4.3.1 Element Contributions

We can build a function that computes the element stiffness matrix and element load vector, for example:

```
function [Ke, Fe]=elementKandF(xe,ke,fe)
  dN=[xe(2,2)-xe(2,3),xe(2,3)-xe(2,1),xe(2,1)-xe(2,2);...
      xe(1,3)-xe(1,2),xe(1,1)-xe(1,3),xe(1,2)-xe(1,1)];
  Ae2=dN(2,3)*dN(1,2)-dN(1,3)*dN(2,2);
  dN=dN/Ae2;
  Ke=Ae2/2*ke*dN'*dN;
  Fe=Ae2*fe*ones(3,1)/6;
end
```

2.4.3.2 Assembly

It only remains to **assemble** the element contributions and impose the Dirichlet boundary conditions to end up with the global stiffness matrix and load vector.

Here we will use a trick that simplifies the coding: Instead of taking care of the boundary conditions during the assembly procedure, we will *assemble the global matrix and load vector as if there were no boundary conditions, and then correct the lines that correspond to unknowns with imposed value*. This is a convenient alternative to the algorithm in §1.4.4 because Octave/MATLAB can efficiently insert/add a submatrix into a larger matrix.

If this trick is adopted, the assembly procedure of finite element stiffness matrices and load vectors is quite intuitive and straightforward to code. In Octave/MATLAB it reads:

```
LG=LV; nod=size(X,2); nel=size(LG,2); npe=size(LG,1);
K=zeros(nod,nod); F=zeros(nod,1);
for iel=1:nel
%% setting the local data
  lge=LG(:,iel);
  xe(1:dd,1:npe)=X(1:dd,lge(1:npe));
  ke=difcoeff(iel);
  fe=source(iel);
%% computing element K and F
  [Ke Fe]=elementKandF(xe,ke,fe);
%% assembly, from local to global
  K(lge,lge)=K(lge,lge)+Ke;
  F(lge)=F(lge)+Fe;
end
```

Here, arrays `difcoeff` and `source` contain the element-wise values k_e and f_e of k and f, respectively. Array `lge` is an index array that contains the numbering of the three nodes of the element. In this way, the matrix `K(lge,lge)` is the submatrix (or "slice") of `K` consisting of just the rows and columns present in `lge`. The key assembly operations

```
  K(lge,lge)=K(lge,lge)+Ke;
  F(lge)=F(lge)+Fe;
```

are thus just a shorter way of coding

```
for j=1:3
   for k=1:3
      K(lge(j),lge(k))=K(lge(j),lge(k))+Ke(j,k);
   end
   F(lge(j))=F(lge(j))+Fe(j);
end
```

which is the lengthier code we introduced originally. For example, if `lge=[7 4 9]`, then the operation

```
K(lge,lge)=K(lge,lge)+Ke;
```

is equivalent to

```
K(7,7)=K(7,7)+Ke(1,1); K(7,4)=K(7,4)+Ke(1,2); K(7,9)=K(7,9)+Ke(1,3);
K(4,7)=K(4,7)+Ke(2,1); K(4,4)=K(4,4)+Ke(2,2); K(4,9)=K(4,9)+Ke(2,3);
K(9,7)=K(9,7)+Ke(3,1); K(9,4)=K(9,4)+Ke(3,2); K(9,9)=K(9,9)+Ke(3,3);
```

It only remains to fix the lines corresponding to the boundary nodes and we will be in a position to solve our first two-dimensional problem with finite elements! Remember what we need to do: If node A is a Dirichlet node with

imposed value g, we must modify the linear system (i.e., the arrays K and F) so that the A-th equation reads, simply, $\mathsf{U}_A = g$. This is easily coded as follows:

```
ng=length(EtaG); II=eye(nod);
for ig=1:ng
 K(EtaG(ig),:)=II(EtaG(ig),:);
 F(EtaG(ig))=GG(ig);
end
```

The code is now complete, we can solve for the unknown vector U which contains the nodal values of u_h and plot the solution.

```
U=K\F;
trisurf(LG',X(1,:),X(2,:),U)
```

The command `trisurf` plots functions defined on arbitrary conforming triangulations.

Example 2.10 A uniformly heated rod of arbitrary polygonal shape.

Consider the geometry shown in Fig. 2.12, only that now we will work with the much finer discretization shown in Fig. 2.16. Let us assume that the rod is homogeneous, with diffusion coefficient equal to 1 and heat source f equal to 10, and that the surface temperature is $g = 20$. We aim to compute the temperature distribution inside the rod as given by the finite element method with a continuous P_1 finite element space.

The whole code needed for this purpose is provided as `octavefemp1a.m` in the accompanying material and reads as shown in Table 2.1. After running it, we have computed our first 2D finite element solution! The function u_h can be seen in Figure 2.17. It provides a good estimation of the exact temperature distribution, since the mesh is quite refined. It can be used to estimate, for example, the maximum temperature in the rod, which yields

$$\max_{x\in\Omega} u_h(x) = 43.077\,.$$

Example 2.11 (The uniformly heated square rod revisited with P_1 elements) By simply changing the geometry of the previous example we can revisit the square-rod problem discussed in Example 2.7.

The results obtained with different meshes are plotted in Figure 2.18. The maximum of u_h is 20.719, 20.733 and 20.737, respectively, for the meshes in part (a), (b) and (c) of the figure. The exact maximum is 20.737. The corresponding number of elements is 68, 242 and 1054, and the number of nodes is 45, 142 and 568. The maximum absolute value of the error $u - u_h$ for each mesh is 0.0193, 0.00557 and 0.00134. We observe that the numerical solution remains stable as the mesh is refined, and converges at all points of the domain to the exact solution.

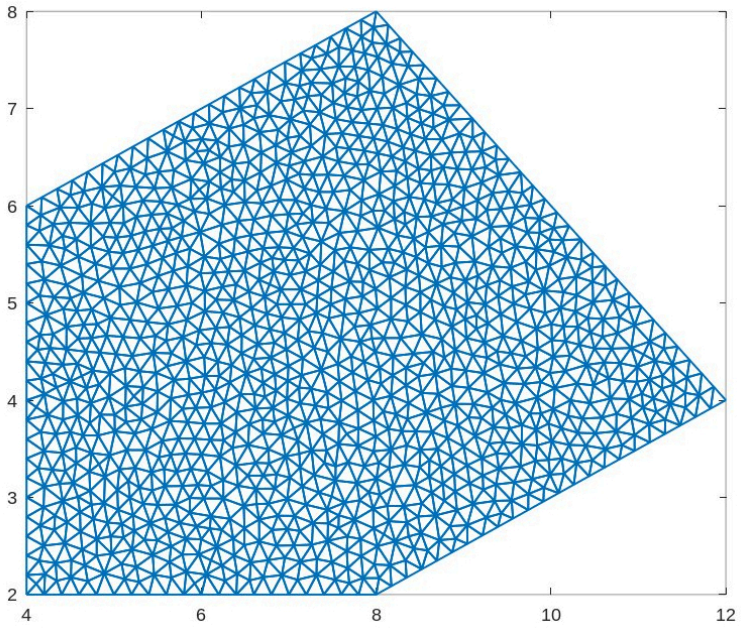

Figure 2.16 Refined triangulation of a polygonal domain.

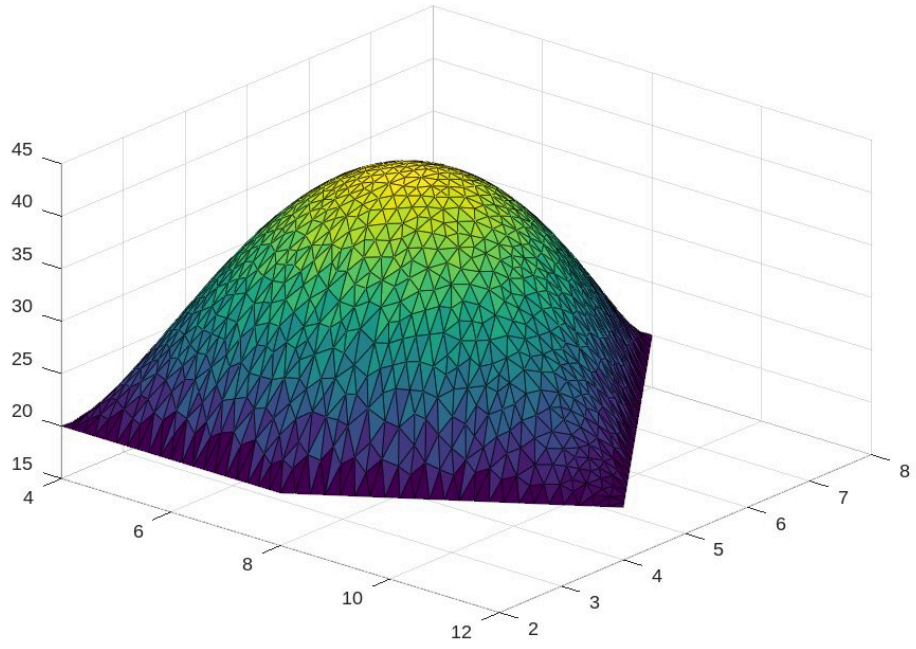

Figure 2.17 Finite element solution u_h of Example 2.10.

```
function [Ke Fe]=elementKandF(xe,ke,fe)
  dN=[xe(2,2)-xe(2,3),xe(2,3)-xe(2,1),xe(2,1)-xe(2,2);...
      xe(1,3)-xe(1,2),xe(1,1)-xe(1,3),xe(1,2)-xe(1,1)];
  Ae2=dN(2,3)*dN(1,2)-dN(1,3)*dN(2,2);
  dN=dN/Ae2;
  Ke=Ae2/2*ke*dN'*dN;
  Fe=Ae2*fe*ones(3,1)/6;
end
%% finite element solver begins (X, LV, EtaG and GG are given)
LG=LV; nod=size(X,2); nel=size(LG,2); npe=size(LG,1);
difcoeff=ones(1,nel);
source=10*ones(1,nel);
GG=20*ones(1,length(EtaG));
K=zeros(nod,nod); F=zeros(nod,1);
for iel=1:nel
%% setting the local data
  lge=LG(:,iel);
  xe(1:dd,1:npe)=X(1:dd,lge(1:npe));
  ke=difcoeff(iel);
  fe=source(iel);
%% computing element K and F
  [Ke Fe]=elementKandF(xe,ke,fe);
%% assembly, from local to global
  K(lge,lge)=K(lge,lge)+Ke;
  F(lge)=F(lge)+Fe;
end
%% boundary nodes
ng=length(EtaG); II=eye(nod);
for ig=1:ng
 K(EtaG(ig),:)=II(EtaG(ig),:);
 F(EtaG(ig))=GG(ig);
end
%% solve algebraic system
U=K\F;
%% plot
trisurf(LG',X(1,:),X(2,:),U)
```

Table 2.1 Code **octavefemp1a.m**. It solves Example 2.10.

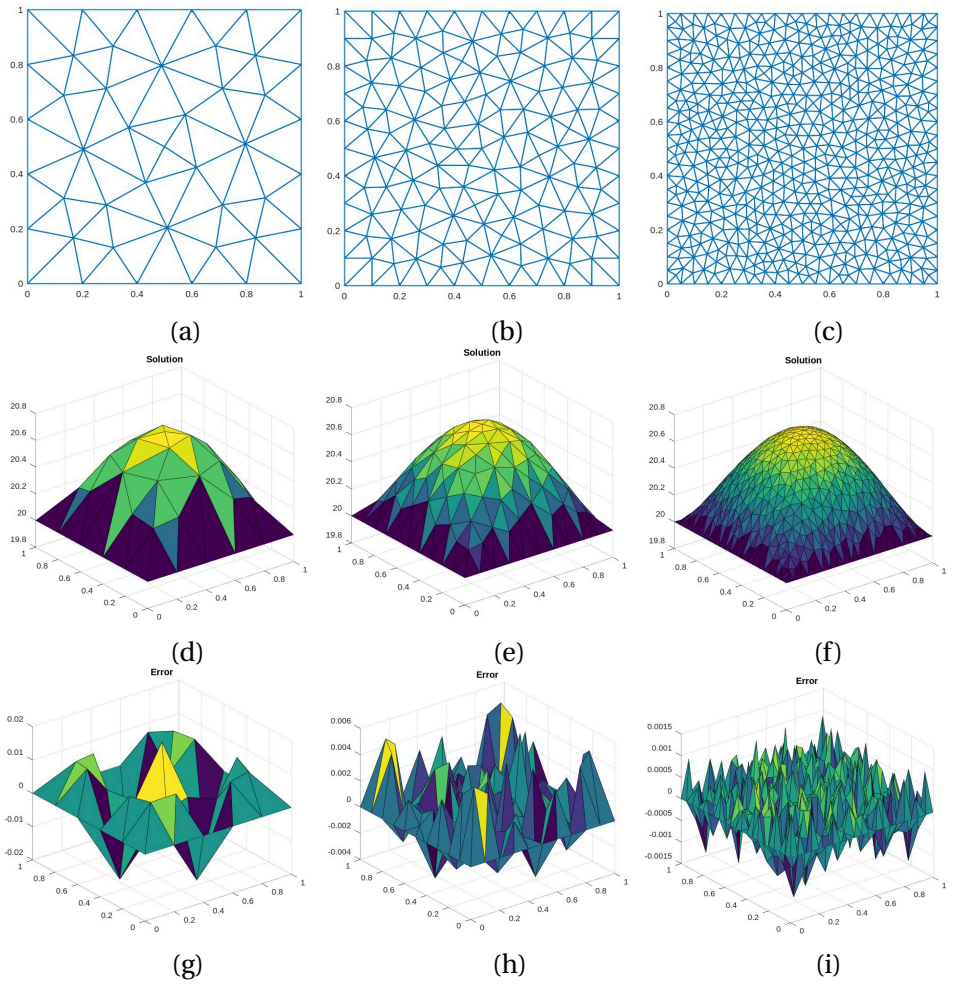

Figure 2.18 The P_1 finite element solutions of the uniformly heated square-rod example. Each column corresponds to a different mesh, which is plotted at the top of the column. The second row shows the finite element solution u_h and the third row shows the error $u - u_h$.

> For comparison purposes we report also the results from a variational method obtained, for this specific case, with the space $\mathcal{W}_h$ of quartic polynomials used in Example 2.7. The maximum of the discrete solution is 20.781 and the maximum error is 0.0445.

2.4.4 Solving Problems with Neumann Boundaries

We have presented a method that approximates the solution of the diffusion equation in any 2D polygonal domain with P_1 finite elements, but just when the solution value is known at the whole boundary (i.e., when the Neumann boundary $\partial\Omega_N$ is empty).

Frequently, however, we have information of the value of the normal flux $H = k\nabla u \cdot \check{n}$ on some parts of the boundary (where $\check{n}$ is the outward normal to $\partial\Omega$). In such cases, the solution is specified on the Dirichlet boundary as before, and the trial and test finite element spaces are still given by (2.53), but there will remain a part $\partial\Omega_N$ where there exist basis functions of $\mathcal{V}_h$ that are not identically zero.

Assuming that node A belongs to $\partial\Omega_N$, the A-th component of the load vector involves two terms,

$$\mathsf{F}_A = \int_\Omega f\, N_A\, d\Omega + \int_{\partial\Omega_N} H\, N_A\, d\Gamma\,,$$

while the corresponding line of the stiffness matrix remains the same.

Symmetry lines and homogeneous Neumann conditions. If the domain and data of the problem are symmetric with respect to some line that crosses the domain, then we can expect the solution to also be. Then one can solve the corresponding equation on a *reduced domain*, consisting of just one half of the original one, and extend the solution found in the reduced domain to the original one by symmetry. The symmetry line is thus a part of the boundary of the reduced domain. What boundary condition should be imposed there? It can be shown that the correct boundary condition is a *Neumann condition* with $H = 0$, because the normal derivative of u must be zero there. These are called **homogeneous Neumann conditions**, and also apply at any isolated (or zero-flux) boundary.

If all parts of the boundary that do not have Dirichlet conditions have homogeneous Neumann conditions, then the code in Table 2.1 works perfectly well as is. It was built considering that the Dirichlet boundary occupies the whole of $\partial\Omega$ and thus the load vector F lacks the contribution

$$\int_{\partial\Omega_N} H\, N_i\, d\Gamma,$$

but, if $H = 0$ all over $\partial\Omega_N$, **there is nothing to be added**. The variational method automatically imposes homogeneous Neumann conditions all over the part of the boundary where the solution value is not imposed. This is because homogeneous Neumann conditions are sometimes called "do-nothing boundary conditions" for this problem.

Example 2.12 (Exploiting symmetries) The case of the uniformly heated square rod exhibits several symmetries. The central horizontal and vertical lines are lines of symmetry, as well as both diagonals. By exploiting those symmetries, we end up with the reduced domain defined as the triangle ABC shown in Fig. 2.19(a). Along the edge BC the Dirichlet condition $u = g$ is applied. The other two edges (AB and CA) must satisfy **homogeneous Neumann boundary conditions** for the solutions in the original and reduced domains to coincide. This is quite useful, since meshing the reduced domain requires significantly less elements of any given size than those required by the original domain (approximately 1/8).

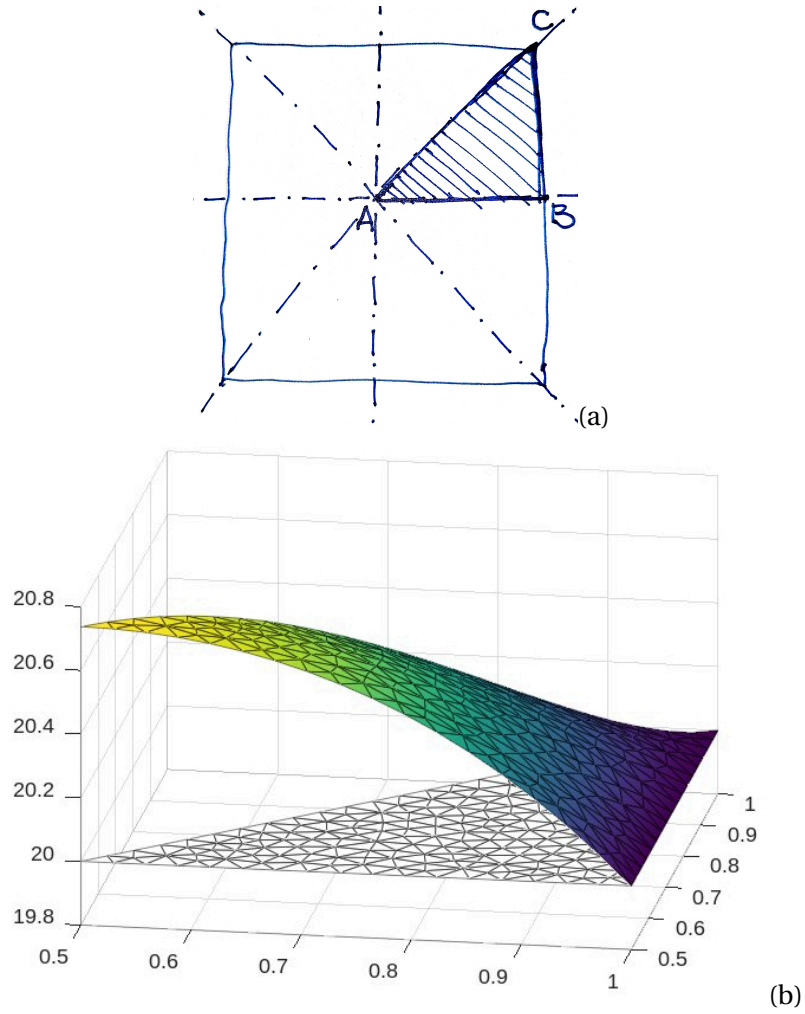

Figure 2.19 (a) Original domain of the square rod problem, and reduced domain (the triangle ABC after exploiting the several symmetries. (b) Finite element solution in the reduced domain, with homogeneous Neumann conditions at symmetry boundaries.

> We modified the previous code to set the triangle ABC as domain and only set the nodes in the edge BC as Dirichlet nodes, with imposed value $g = 20$. Nothing special was programmed for the other boundary nodes, they were treated just like internal nodes. The other constants are as in Example 2.11, i.e., $k = 1$ and $f = 10$. The result obtained with a mesh of 380 elements and 220 nodes is shown in Fig. 2.19(b). The maximum error of the discrete solution is 0.00064, smaller than the error attained in the whole domain with a mesh of 1054 elements and 568 nodes.

Non-homogeneous Neumann boundary conditions. When $H \neq 0$ one needs to build the complete load vector, which, as said, if A is a non-Dirichlet node

reads

$$\mathsf{F}_A = \int_\Omega f\, N_A\, d\Omega + \int_{\partial\Omega_N} H\, N_A\, d\Gamma\,.$$

The second integral is our focus of attention now. The differential $d\Gamma$ is a differential of length, because $\partial\Omega$ is one-dimensional. Our first task is to describe the **data structure** with which we handle the definition of $\partial\Omega_N$ and of the function H inside the code.

The **mesh generator** builds the domain boundary by joining together ℓ individual lines provided by the user, of which some (or all) belong to the Neumann boundary. Let us assume that the produced triangulation has n_{be} boundary edges. By construction, each boundary edge belongs to one and only one of these lines. The generator provides an **array of boundary edges** BE of dimension $2 \times n_{\text{be}}$. Each column of BE contains three numbers. The first two indicate the two nodes of the corresponding edge. The third one indicates the line to which the edge belongs.

To keep things simple, we can assume that H is constant over each edge. Then it can be provided by an array H of n_{be} entries, as shown in the example below.

Example 2.13 (Boundary arrays of a triangulation) In Figure 2.20 we show a small triangulation of a polygon, defined by points 1-5 and lines 1-5. For this triangulation $n_{\text{be}} = 11$, and BE is as follows:

$$\mathsf{BE} = \begin{pmatrix} 1 & 6 & 2 & 7 & 5 & 8 & 9 & 4 & 10 & 3 & 11 \\ 6 & 2 & 7 & 5 & 8 & 9 & 4 & 10 & 3 & 11 & 1 \\ 1 & 1 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 5 & 5 \end{pmatrix}$$

Now suppose that along **line 2** the imposed heat flux H is constant and equal to $\frac{2}{3}$, while along **line 3** it is also constant but equal to $\frac{1}{2}$. Along lines 1, 4 and 5 the Dirichlet condition $u = 20$ is imposed. This is specified in the array H that provides the value of H over each edge, assuming it is constant at each edge.

$$\mathsf{H} = \begin{pmatrix} 0 & 0 & \frac{2}{3} & \frac{2}{3} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \end{pmatrix}$$

The construction of H is easily coded as

```
for k=1:nbe
   if (BE(3,k)==2)
      HH(k)=2./3;
   end
   if (BE(3,k)==3)
      HH(k)=1./2;
   end
end
```

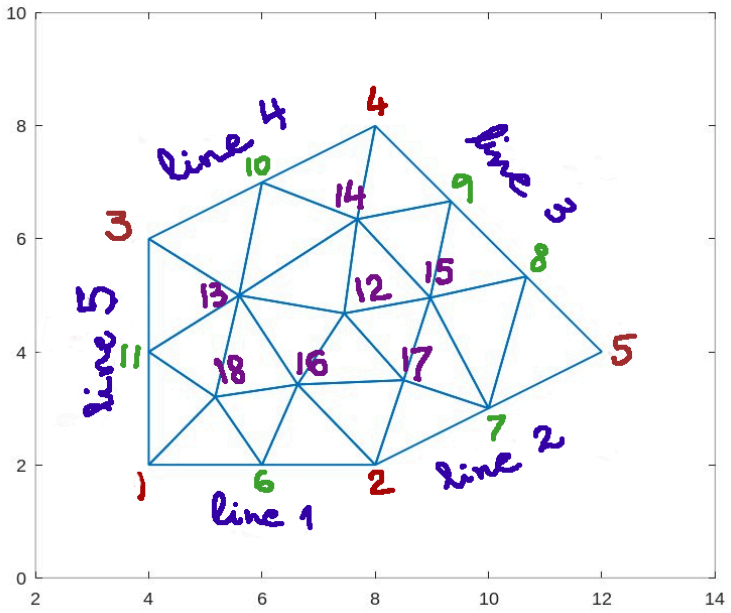

Figure 2.20 A small triangulation, showing the numbering of the defining points (in red), of the defining lines (in blue), and of the rest of the vertices.

It should be clear by now that a P_1 function restricted to an edge of the triangulation is *affine*. The function interpolates linearly along the edge between the two nodal values at its ends. This implies that, if the k-th boundary edge (let us denote it by E_k) joins the nodes $A_1 = \mathsf{BE}(1,k)$ and $A_2 = \mathsf{BE}(2,k)$, then the only two basis functions that are different from zero along the edge are N_{A_1} and N_{A_2}. Further, these two basis functions go affinely from 0 to 1 over the edge, so that their average value is $\frac{1}{2}$. We can thus compute the integral over E_k as

$$\int_{E_k} H\, N_{A_1}\, d\Gamma = \int_{E_k} H\, N_{A_2}\, d\Gamma = \frac{1}{2}\,\mathsf{H}(k)\,|E_k|\,,$$

where $|E_k|$ is the length of the edge,

$$|E_k| = \|X^{A_1} - X^{A_2}\|\,.$$

We can implement the addition of the Neumann part of the load vector as an assembly procedure.

```
%% Neumann contributions
for ied=1:nbe
  lged=BE(1:2,ied);
  xed(1:dd,1:2)=X(1:dd,lged);
  Led=norm(xed(:,1)-xed(:,2));
  Hed=HH(ied);
  F(lged)=F(lged)+Hed*Led/2;
end
```

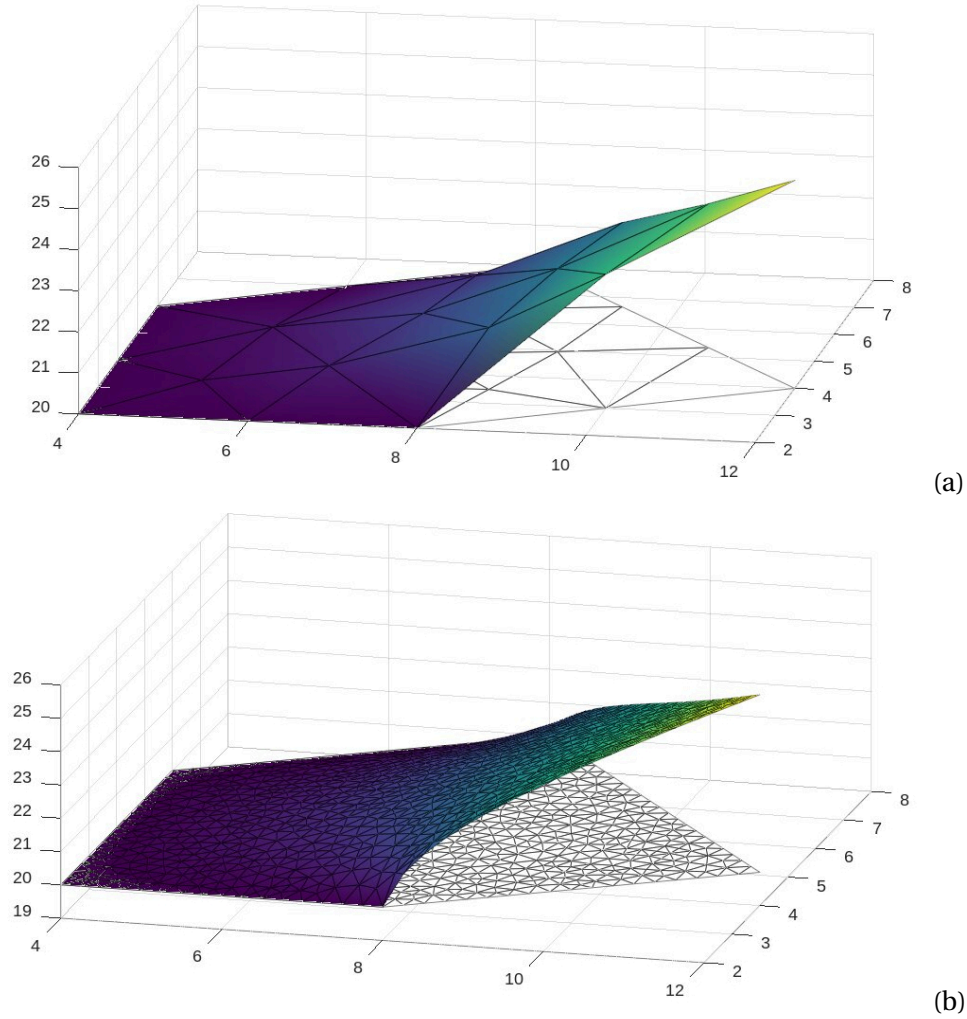

Figure 2.21 Solution to Example 2.14 with two meshes with different resolutions.

Remark: Of course, the contribution of each edge will depend on the specific function H. Piecewise polynomial functions of higher degree can easily be implemented.

Example 2.14 (Solution of the Neumann problem of Example 2.13) The procedure above has been implemented in **octavefemp1b.m**. The source f was set to zero and the diffusion coefficient to $q = 1$. The results obtained for the mesh of Figure 2.20 and for a finer mesh on the same geometry are shown in Figure 2.21.