Chapter 10
Computing the load vector

The basic algorithm for computing the load vector $F$ is similar to that for assembling the stiffness matrix $K$, although it is complicated by the need to handle inhomogeneous boundary conditions. I will begin by completing the description of Algorithm 6.3 (see page 108), which applies to the following BVP (with homogeneous boundary conditions):

$$-\nabla \cdot (\kappa(x,y)\nabla u) = f(x,y) \quad \text{in } \Omega, \quad (10.1)$$

$$u = 0 \quad \text{on } \Gamma_1, \quad (10.2)$$

$$\kappa \frac{\partial u}{\partial n} = 0 \quad \text{on } \Gamma_2. \quad (10.3)$$

The integrals

$$\int_{T_h} f \phi_i$$

will be computed using the one-point quadrature rule

$$\int_{T_h} f \phi_i \approx A f(\bar{x},\bar{y}) \phi_i(\bar{x},\bar{y}),$$

where $A$ is the area and $(\bar{x},\bar{y})$ the centroid of $T_h$. Moreover, since all three basis functions have value 1/3 at $(\bar{x},\bar{y})$, this estimate is the same for all three basis functions that are nonzero over $T_h$:

$$\int_{T_h} f \phi_i \approx \frac{1}{3} A f(\bar{x},\bar{y}).$$

The details (getting the coordinates of the vertices of $T_h$, manipulating the pointers, computing the area of $T_h$, and so forth) are the same as for assembling the stiffness matrix. The complete algorithm for assembling $F$, under homogeneous boundary conditions, is given in Algorithm 10.1.
Initialize \( F \) to the zero vector
for \( k = 1, 2, \ldots, N_f \)
Call `getNodes` to get coords and `ptrs`
Compute the area \( A \) and the centroid \((\overline{x}, \overline{y})\) of \( T_k \)
Compute \( I = A f(\overline{x}, \overline{y})/3 \)
for \( r = 1, 2, 3 \)
if \( \text{ptr}(r) > 0 \)
Add \( I \) to \( F(\text{ptr}(r)) \)

| Table 10.1. The algorithm for assembling the load vector in the case of homogeneous boundary conditions. |

### 10.1 Inhomogeneous Dirichlet conditions

It is not difficult to incorporate, into the above algorithm, code to handle inhomogeneous Dirichlet boundary conditions. In place of

\[
F_i = \int_{\Omega} f \phi_i, \quad i = 1, 2, \ldots, N_f,
\]

it is necessary to compute

\[
F_i = \int_{\Omega} f \phi_i - \int_{\Omega} \kappa \nabla G \cdot \nabla \phi_i, \quad i = 1, 2, \ldots, N_f.
\]

The function \( G \) is the continuous piecewise linear function defined by

\[
G(v_n) = \begin{cases} 
0, & v_n \notin \Gamma_1, \\
g(v_n), & v_n \in \Gamma_1.
\end{cases}
\]

By definition, \( G \) is zero over every triangle except those having at least one constrained vertex. When looping over the triangles of the mesh, a contribution to \( F \) (from the inhomogeneous Dirichlet condition) must be computed whenever the triangle \( T_k \) contains at least one constrained node and at least one free node. These contributions have the form

\[
-\int_{T_k} \kappa \nabla G \cdot \nabla \phi_i.
\]

Since both \( G \) and \( \phi_i \) have constant gradients, it follows that

\[
-\int_{T_k} \kappa \nabla G \cdot \nabla \phi_i = -\nabla G \cdot \nabla \phi_i \int_{T_k} \kappa.
\]

Moreover, \( \nabla G \) is easily computed. If \( w_1, w_2, w_3 \) are the nodal values of \( G \) at the three nodes of \( T_k \), then, on \( T_k \),

\[
G = \sum_{i=1}^{3} w_i \phi_i \Rightarrow \nabla G = \sum_{i=1}^{3} w_i \nabla \phi_i.
\]
(here I am using local indices for the basis functions that are nonzero on \( T_k \)).

Algorithm 10.2 computes the load vector, taking into account the influence of the right-hand-side function \( f \) and the Dirichlet data \( g \) (but still ignoring any nonzero Neumann data). This algorithm assumes that the nonzero Dirichlet data is given in a \( N_c \times 1 \) array.

```markdown
Initialize \( F \) to the zero vector
for \( k = 1, 2, \ldots, N_t \)
  Call `getNodes` to get coords and ptrs
  Compute the area \( A \) and the centroid \((x, y)\) of \( T_k \)
  if \( f \) is nonzero
    Compute \( I = Af(x, y)/3 \)
    for \( r = 1, 2, 3 \)
      if \( \text{ptr}(r) > 0 \)
        Add \( I \) to \( F(\text{ptr}(r)) \)
  if \( g \) is nonzero and at least one \( \text{ptr}(r) \) is nonzero
    Form the matrix \( M \) and its inverse \( C \)
    Extract the nodal values of \( G \) on \( T_k \)
    Compute the gradient of \( G \) on \( T_k \)
    Compute \( I = Ak(x, y) \)
    for \( r = 1, 2, 3 \)
      if \( \text{ptr}(r) > 0 \)
        Add \( \nabla G \cdot \nabla \phi_r I \) to \( F(\text{ptr}(r)) \)
```

**Table 10.2.** The algorithm for assembling the load vector in the case of a nonzero right-hand side \( f \) and/or nonzero Dirichlet data \( g \).

### 10.2 Inhomogeneous Neumann conditions

If the problem involves inhomogeneous Neumann conditions, the resulting contribution to the load vector can also be computed while looping over the triangles of the mesh. It can be determined from the array `EdgeEls` whether each edge is an interior edge, a constrained boundary edge, or a free boundary edge. To be precise, `T.EdgeEls(j, 2)` is positive if edge \( e_j \) is an interior edge, zero if \( e_j \) is a constrained boundary edge, and \(-i\) is \( e_j \) is the \( i \)th free boundary edge.

The formula for the load vector, taking into account both Dirichlet and Neumann data, is

\[
F_i = \int_{\Omega} f \phi_i - \int_{\Omega} \kappa \nabla G \cdot \nabla \phi_i + \int_{\Gamma_z} h \phi_i, \quad i = 1, 2, \ldots, N_f.
\]

If node \( v_i \) is a free node that lies on \( \partial \Omega \), then it is the endpoint of two free boundary edges. Therefore, if \( i = p_m \) (this notation means that \( v_i \) is the \( i \)th free node) and
$v_n$ is an endpoint of free edges $e_{j_1}$ and $e_{j_2}$, then

$$
\int_{\Gamma_2} h\phi_i = \int_{e_{j_1}} h\phi_i + \int_{e_{j_2}} h\phi_i.
$$

These integrals can be computed while looping over the triangles. The reader should notice that $e_{j_1}$ and $e_{j_2}$ may or may not belong to the same triangle, but, as in the case of the integrals defining $K_{ij}$, it is not necessary that the two integrals be computed together.

I assume that the nodal values of the Neumann data $h$, at the endpoints of the free edges, are provided in an $N_b \times 2$ array. The function $h$ is then approximated by its piecewise linear interpolant on each edge. Each integral of the form

$$
\int_{e} h\phi_i
$$

is then estimated by the one-point Gaussian quadrature rule (the midpoint rule) for one-dimensional integrals.

The reader may notice that the array $FBndyEdges$, which identifies the free boundary edges, is not used while computing the load vector. Its only purpose is to identify the free boundary edges in a convenient way so that the needed Neumann data can be determined in a convenient way.

Algorithm 10.3 is the complete algorithm for computing the load vector.


10.2. Inhomogeneous Neumann conditions

Initialize $F$ to the zero vector
for $k = 1, 2, \ldots, N_t$

Call getNodes to get coords and ptrs
if $f$ is nonzero

Compute the area $A$ and the centroid $(\bar{x}, \bar{y})$ of $T_k$
Compute $I = A f(\bar{x}, \bar{y}) / 3$
for $r = 1, 2, 3$

if ptr($r$) > 0

Add $I$ to $F$(ptr($r$))
if $g$ is nonzero and at least one ptr($r$) is nonzero
if not already done

Compute the area $A$ and the centroid $(\bar{x}, \bar{y})$ of $T_k$
Form the matrix $M$ and its inverse $C$
Extract the nodal values of $G$ on $T_k$
Compute the gradient of $G$ on $T_k$
Compute $I = A \kappa(\bar{x}, \bar{y})$
for $r = 1, 2, 3$

if ptr($r$) > 0

Add $\nabla G \cdot \nabla \phi, I$ to $F$(ptr($r$))
if $h$ is nonzero
for $j = 1, 2, 3$

if edge $j$ is a free boundary edge

Extract the coordinates of the endpoints of the edge
Extract the corresponding pointers from NodePtrs
Compute the length $L$ of the edge
Evaluate the boundary data at the midpoint of the edge to get $h(m)$
Evaluate $I = 0.5 L h(m)$
if ptr(1) > 0

Add $I$ to $F$(ptr(1))
if ptr(2) > 0

Add $I$ to $F$(ptr(2))

Table 10.3. The complete algorithm for assembling the load vector.