

8

MultiFreedom Constraints I

TABLE OF CONTENTS

	Page
§8.1. Classification of Constraint Conditions	8-3
§8.1.1. MultiFreedom Constraints	8-3
§8.1.2. Methods for Imposing Multifreedom Constraints	8-4
§8.1.3. *MFC Matrix Forms	8-5
§8.2. The Example Structure	8-5
§8.3. The Master-Slave Method	8-6
§8.3.1. A One-Constraint Example	8-7
§8.3.2. Several Homogeneous MFCs	8-8
§8.3.3. Nonhomogeneous MFCs	8-9
§8.3.4. *The General Case	8-9
§8.3.5. *Retaining the Original Freedoms	8-10
§8.3.6. Model Reduction by Kinematic Constraints	8-10
§8.3.7. Assessment of the Master-Slave Method	8-13
§8. Notes and Bibliography	8-14
§8. References	8-14
§8. Exercises	8-15

§8.1. Classification of Constraint Conditions

In previous Chapters we have considered structural support conditions that are mathematically expressible as constraints on individual degrees of freedom:

$$\boxed{\text{nodal displacement component} = \text{prescribed value.}} \quad (8.1)$$

These are called *single-freedom constraints*. Chapter 3 explains how to incorporate constraints of this form into the master stiffness equations, using hand- or computer-oriented techniques. The displacement boundary conditions studied in Chapter 7, including the modeling of symmetry and antisymmetry, lead to constraints of this form.

For example:

$$u_{x4} = 0, \quad u_{y9} = 0.6. \quad (8.2)$$

These are two single-freedom constraints. The first one is homogeneous while the second one is non-homogeneous. These attributes are defined below.

§8.1.1. MultiFreedom Constraints

The next step up in complexity involves *multifreedom equality constraints*, or *multifreedom constraints* for short, the last name being acronymed to MFC. These are functional equations that connect *two or more* displacement components:

$$\boxed{F(\text{nodal displacement components}) = \text{prescribed value,}} \quad (8.3)$$

where function F vanishes if all its nodal displacement arguments do. Equation (8.3), where all displacement components are in the left-hand side, is called the *canonical form* of the constraint.

An MFC of this form is called *multipoint* or *multinode* if it involves displacement components at different nodes. The constraint is called *linear* if all displacement components appear linearly on the left-hand-side, and *nonlinear* otherwise.

The constraint is called *homogeneous* if, upon transferring all terms that depend on displacement components to the left-hand side, the right-hand side — the “prescribed value” in (8.3) — is zero. It is called *non-homogeneous* otherwise.

In this and next Chapter only linear constraints will be studied. Furthermore more attention is devoted to the homogeneous case, because it arises more frequently in practice.

Remark 8.1. The most general constraint class is that of inequality constraints, such as $u_{y5} - 2u_{x2} \geq 0.5$. These constraints are relatively infrequent in linear structural analysis, except in problems that involve contact conditions. They are of paramount importance, however, in other fields such as optimization and control.

Example 8.1. Here are three instances of MFCs:

$$u_{x2} = \frac{1}{2}u_{y2}, \quad u_{x2} - 2u_{x4} + u_{x6} = \frac{1}{4}, \quad (x_5 + u_{x5} - x_3 - u_{x3})^2 + (y_5 + u_{y5} - y_3 - u_{y3})^2 = 0. \quad (8.4)$$

The first one is linear and homogeneous. It is not a multipoint constraint because it involves the displacement components of one node: 2.

The second one is multipoint because it involves three nodes: 2, 4 and 6. It is linear and non-homogeneous.

The last one is multipoint, nonlinear and homogeneous. Geometrically it expresses that the distance between nodes 3 and 5 in two-dimensional motions on the $\{x, y\}$ plane remains constant. This kind of constraint appears in geometrically nonlinear analysis of structures, which is a topic beyond the scope of this book.

§8.1.2. Methods for Imposing Multifreedom Constraints

Accounting for multifreedom constraints is done — at least conceptually — by changing the assembled master stiffness equations to produce a *modified* system of equations:

$$\mathbf{K}\mathbf{u} = \mathbf{f} \xrightarrow{\text{MFC}} \hat{\mathbf{K}}\hat{\mathbf{u}} = \hat{\mathbf{f}}. \quad (8.5)$$

The modification process (8.5) is also called *constraint application* or *constraint imposition*. The modified system is that submitted to the equation solver, which returns $\hat{\mathbf{u}}$.

The procedure is flowcharted in Figure 8.1. The sequence of operations sketched therein applies to all methods outlined below.

Three methods for treating MFCs are discussed in this and the next Chapter:

1. *Master-Slave Elimination.* The degrees of freedom involved in each MFC are separated into master and slave freedoms. The slave freedoms are then explicitly eliminated. The modified equations do not contain the slave freedoms.
2. *Penalty Augmentation.* Also called the *penalty function method*. Each MFC is viewed as the presence of a fictitious elastic structural element called *penalty element* that enforces it approximately. This element is parametrized by a numerical *weight*. The exact constraint is recovered if the weight goes to infinity. The MFCs are imposed by augmenting the finite element model with the penalty elements.
3. *Lagrange Multiplier Adjunction.* For each MFC an additional unknown is adjoined to the master stiffness equations. Physically this set of unknowns represent *constraint forces* that would enforce the constraints exactly should they be applied to the unconstrained system.

For each method the exposition tries to give first the basic flavor by working out the same example for each method. The general technique is subsequently presented in matrix form for completeness but is considered an advanced topic.

Conceptually, imposing MFCs is not different from the procedure discussed in Chapter 3 for single-freedom constraints. The master stiffness equations are assembled ignoring all constraints. Then

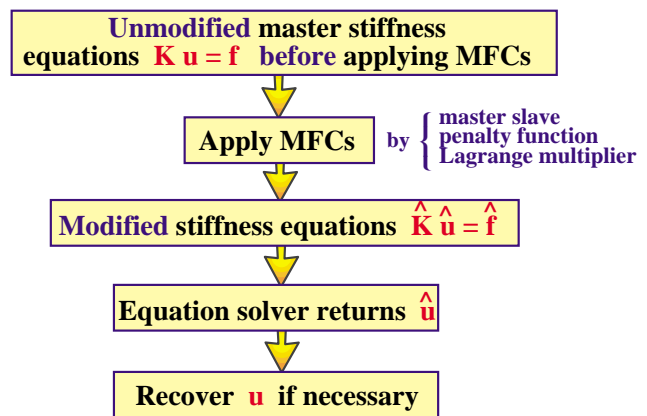


FIGURE 8.1. Schematics of MFC application.

the MFCs are imposed by appropriate modification of those equations. There are, however, two important practical differences:

1. The modification process is not unique because there are alternative constraint imposition methods, namely those listed above. These methods offer tradeoffs in generality, programming implementation complexity, computational effort, numerical accuracy and stability.
2. In the implementation of some of these methods — notably penalty augmentation — constraint imposition and assembly are carried out simultaneously. In that case the framework “first assemble, then modify,” is not strictly respected in the actual implementation.

Remark 8.2. The three methods are also applicable, as can be expected, to the simpler case of a single-freedom constraint such as (8.2). For most situations, however, the generality afforded by the penalty function and Lagrange multiplier methods are not warranted. The hand-oriented reduction process discussed in Chapters 3 and 4 is in fact a special case of the master-slave elimination method in which “there is no master.”

Remark 8.3. Often both multifreedom and single-freedom constraints are prescribed. The modification process then involves two stages: apply multifreedom constraints and apply single freedom constraints. The order in which these are carried out is implementation dependent. Most implementations do the MFCs first, either after the assembly is completed or during the assembly process. The reason is practical: single-freedom constraints are often automatically taken care of by the equation solver itself.

§8.1.3. *MFC Matrix Forms

Matrix forms of linear MFCs are often convenient for compact notation. An individual constraint such as the second one in (8.4) may be written

$$[1 \quad -2 \quad 1] \begin{bmatrix} u_{x2} \\ u_{x4} \\ u_{x6} \end{bmatrix} = 0.25. \quad (8.6)$$

In direct matrix notation:

$$\bar{\mathbf{a}}_i \bar{\mathbf{u}}_i = g_i, \quad (\text{no sum on } i) \quad (8.7)$$

in which index i ($i = 1, 2, \dots$) identifies the constraint, $\bar{\mathbf{a}}_i$ is a row vector, $\bar{\mathbf{u}}_i$ collects the set of degrees of freedom that participate in the constraint, and g_i is the right hand side scalar (0.25 in the foregoing example). The bars over \mathbf{a} and \mathbf{u} distinguishes (8.7) from the expanded form (8.9) discussed below.

For method description and general proofs it is often convenient to expand matrix forms so that they embody *all* degrees of freedom. For example, if (8.6) is part of a two-dimensional finite element model with 12 freedoms: $u_{x1}, u_{y1}, \dots, u_{y6}$, the left-hand side row vector may be expanded with nine zeros as follows

$$[0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad -2 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0] \begin{bmatrix} u_{x1} \\ u_{y1} \\ u_{x2} \\ \vdots \\ u_{y6} \end{bmatrix} = 0.25, \quad (8.8)$$

in which case the matrix notation

$$\mathbf{a}_i \mathbf{u} = g_i \quad (8.9)$$

is used. Finally, all multifreedom constraints expressed as (8.9) may be collected into a single matrix relation:

$$\mathbf{A} \mathbf{u} = \mathbf{g}, \quad (8.10)$$

where rectangular matrix \mathbf{A} is formed by stacking the \mathbf{a}_i 's as rows and column vector \mathbf{g} is formed by stacking the g_i s as entries. If there are 12 degrees of freedom in \mathbf{u} and 5 multifreedom constraints then \mathbf{A} will be 5×12 .

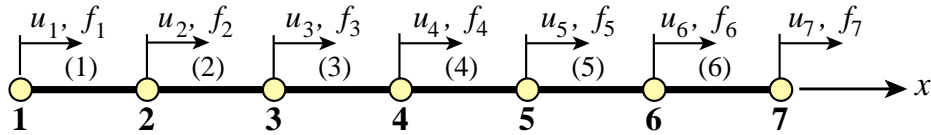


FIGURE 8.2. A one-dimensional problem discretized with six bar finite elements. The seven nodes may move only along the x direction. Subscript x is omitted from the u 's and f 's to reduce clutter.

§8.2. The Example Structure

The one-dimensional finite element discretization shown in Figure 8.2 will be used throughout Chapters 8 and 9 to illustrate the three MFC application methods. This structure consists of six bar elements connected by seven nodes that can only displace in the x direction.

Before imposing various multifreedom constraints discussed below, the master stiffness equations for this problem are assumed to be

$$\begin{bmatrix} K_{11} & K_{12} & 0 & 0 & 0 & 0 & 0 \\ K_{12} & K_{22} & K_{23} & 0 & 0 & 0 & 0 \\ 0 & K_{23} & K_{33} & K_{34} & 0 & 0 & 0 \\ 0 & 0 & K_{34} & K_{44} & K_{45} & 0 & 0 \\ 0 & 0 & 0 & K_{45} & K_{55} & K_{56} & 0 \\ 0 & 0 & 0 & 0 & K_{56} & K_{66} & K_{67} \\ 0 & 0 & 0 & 0 & 0 & K_{67} & K_{77} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix}, \quad (8.11)$$

or

$$\mathbf{Ku} = \mathbf{f}. \quad (8.12)$$

The nonzero stiffness coefficients K_{ij} in (8.11) depend on the bar rigidity properties. For example, if $E^e A^e / L^e = 100$ for each element $e = 1, \dots, 6$, then $K_{11} = K_{77} = 100$, $K_{22} = \dots = K_{66} = 200$, $K_{12} = K_{23} = \dots = K_{67} = -100$. However, for the purposes of the following treatment the coefficients may be kept arbitrary. The component index x in the nodal displacements u and nodal forces f has been omitted for brevity.

Now let us specify a multifreedom constraint that states that nodes 2 and 6 are to move by the same amount:

$$u_2 = u_6. \quad (8.13)$$

Passing all node displacements to the right hand side gives the canonical form:

$$\boxed{u_2 - u_6 = 0}. \quad (8.14)$$

Constraint conditions of this type are sometimes called *rigid links* because they can be mechanically interpreted as forcing node points 2 and 6 to move together as if they were tied by a rigid member.¹

We now study the imposition of constraint (8.14) on the master equations (8.11) by the methods mentioned above. In this Chapter the master-slave method is treated. The other two methods: penalty augmentation and Lagrange multiplier adjunction, are discussed in the following Chapter.

¹ This physical interpretation is exploited in the penalty method described in the next Chapter. In two and three dimensions rigid link constraints are more complicated.

§8.3. The Master-Slave Method

To apply this method *by hand*, the MFCs are taken one at a time. For each constraint a *slave* degree of freedom is chosen. The freedoms remaining in that constraint are labeled *master*. A new set of degrees of freedom $\hat{\mathbf{u}}$ is established by removing all slave freedoms from \mathbf{u} . This new vector contains master freedoms as well as those that do not appear in the MFCs. A matrix transformation equation that relates \mathbf{u} to $\hat{\mathbf{u}}$ is generated. This equation is used to apply a congruent transformation to the master stiffness equations. This procedure yields a set of modified stiffness equations that are expressed in terms of the new freedom set $\hat{\mathbf{u}}$. Because the modified system does not contain the slave freedoms, these have been effectively eliminated.

§8.3.1. A One-Constraint Example

The mechanics of the process is best seen by going through an example. To impose (8.14) pick u_6 as slave and u_2 as master. Relate the original unknowns u_1, \dots, u_7 to the new set in which u_6 is missing:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_7 \end{bmatrix}, \quad (8.15)$$

This is the required transformation relation. In compact form:

$$\boxed{\mathbf{u} = \mathbf{T}\hat{\mathbf{u}}}. \quad (8.16)$$

Replacing (8.15) into (8.12) and premultiplying by \mathbf{T}^T yields the modified system

$$\boxed{\hat{\mathbf{K}}\hat{\mathbf{u}} = \hat{\mathbf{f}}, \quad \text{in which} \quad \hat{\mathbf{K}} = \mathbf{T}^T \mathbf{K} \mathbf{T}, \quad \hat{\mathbf{f}} = \mathbf{T}^T \mathbf{f}. \quad (8.17)}$$

Carrying out the indicated matrix multiplications yields

$$\begin{bmatrix} K_{11} & K_{12} & 0 & 0 & 0 & 0 \\ K_{12} & K_{22} + K_{66} & K_{23} & 0 & K_{56} & K_{67} \\ 0 & K_{23} & K_{33} & K_{34} & 0 & 0 \\ 0 & 0 & K_{34} & K_{44} & K_{45} & 0 \\ 0 & K_{56} & 0 & K_{45} & K_{55} & 0 \\ 0 & K_{67} & 0 & 0 & 0 & K_{77} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_7 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 + f_6 \\ f_3 \\ f_4 \\ f_5 \\ f_7 \end{bmatrix}, \quad (8.18)$$

Equation (8.18) is a new linear system containing 6 equations in the remaining 6 unknowns: u_1, u_2, u_3, u_4, u_5 and u_7 . Upon solving it, u_6 is recovered from the constraint (8.13).

Remark 8.4. The form of modified system (8.17) can be remembered by a simple mnemonic rule: premultiply both sides of $\mathbf{T}\hat{\mathbf{u}} = \mathbf{u}$ by $\mathbf{T}^T \mathbf{K}$, and replace $\mathbf{K}\mathbf{u}$ by \mathbf{f} on the right hand side.

Remark 8.5. For a simple freedom constraint such as $u_4 = 0$ the only possible choice of slave is of course u_4 and there is no master. The congruent transformation is then nothing more than the elimination of u_4 by striking out rows and columns from the master stiffness equations.

Remark 8.6. For a simple MFC such as $u_2 = u_6$, it does not matter which degree of freedom is chosen as master or unknown. Choosing u_2 as slave produces a system of equations in which now u_2 is missing:

$$\begin{bmatrix} K_{11} & 0 & 0 & 0 & K_{12} & 0 \\ 0 & K_{33} & K_{34} & 0 & K_{23} & 0 \\ 0 & K_{34} & K_{44} & K_{45} & 0 & 0 \\ 0 & 0 & K_{45} & K_{55} & K_{56} & 0 \\ K_{12} & K_{23} & 0 & K_{56} & K_{22} + K_{66} & K_{67} \\ 0 & 0 & 0 & 0 & K_{67} & K_{77} \end{bmatrix} \begin{bmatrix} u_1 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_3 \\ f_4 \\ f_5 \\ f_2 + f_6 \\ f_7 \end{bmatrix}. \quad (8.19)$$

Although (8.18) and (8.19) are algebraically equivalent, the latter would be processed faster if a skyline solver (Part III of course) is used for the modified equations.

§8.3.2. Several Homogeneous MFCs

The matrix equation (8.17) in fact holds for the general case of multiple homogeneous linear constraints. Direct establishment of the transformation equation, however, is more complicated if slave freedoms in one constraint appear as masters in another. To illustrate this point, suppose that for the example system we have three homogeneous multifreedom constraints:

$$u_2 - u_6 = 0, \quad u_1 + 4u_4 = 0, \quad 2u_3 + u_4 + u_5 = 0, \quad (8.20)$$

Picking as slave freedoms u_6 , u_4 and u_3 from the first, second and third constraint, respectively, we can solve for them as

$$u_6 = u_2, \quad u_4 = -\frac{1}{4}u_1, \quad u_3 = -\frac{1}{2}(u_4 + u_5) = \frac{1}{8}u_1 - \frac{1}{2}u_5. \quad (8.21)$$

Observe that solving for u_3 from the third constraint brings u_4 to the right-hand side. But because u_4 is also a slave freedom (it was chosen as such for the second constraint) it is replaced in favor of u_1 using $u_4 = -\frac{1}{4}u_1$. The matrix form of the transformation (8.21) is

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{8} & 0 & -\frac{1}{2} & 0 \\ -\frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_5 \\ u_7 \end{bmatrix}, \quad (8.22)$$

The modified system is now formed through the congruent transformation (8.17). Note that the slave freedoms selected from each constraint must be distinct; for example the choice u_6 , u_4 , u_4 would be inadmissible as long as the constraints are independent. This rule is easy to enforce when slave freedoms are chosen by hand, but can lead to implementation and numerical difficulties when it is programmed as an automated procedure, as further discussed later.

Remark 8.7. The three MFCs (8.20) with u_6, u_4 and u_2 chosen as slaves and u_1, u_2 and u_5 chosen as masters, may be presented in the partitioned matrix form:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 4 & 0 \\ 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_3 \\ u_4 \\ u_6 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_5 \end{bmatrix} \quad (8.23)$$

This may be compactly written $\mathbf{A}_s \mathbf{u}_s + \mathbf{A}_m \mathbf{u}_m = \mathbf{0}$. Solving for the slave freedoms gives $\mathbf{u}_s = -\mathbf{A}_s^{-1} \mathbf{A}_m \mathbf{u}_m$. Expanding with zeros to fill out \mathbf{u} and $\hat{\mathbf{u}}$ produces (8.22). This general matrix form is considered in §8.4.4. Note that non-singularity of \mathbf{A}_s is essential for this method to work.

§8.3.3. Nonhomogeneous MFCs

Extension to non-homogeneous constraints is immediate. In this case the transformation equation becomes non-homogeneous. For example suppose that (8.14) has a nonzero prescribed value:

$$\boxed{u_2 - u_6 = 0.2} \quad (8.24)$$

Nonzero RHS values such as 0.2 in (8.24) may be often interpreted physically as “gaps” (thus the use of the symbol \mathbf{g} in the matrix form). Chose u_6 again as slave: $u_6 = u_2 - 0.2$, and build the transformation

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.2 \\ 0 \end{bmatrix}. \quad (8.25)$$

In compact matrix notation,

$$\boxed{\mathbf{u} = \mathbf{T} \hat{\mathbf{u}} + \mathbf{g}.} \quad (8.26)$$

Here the constraint gap vector \mathbf{g} is nonzero and \mathbf{T} is the same as before. To get the modified system applying the shortcut rule of Remark 8.4, premultiply both sides of (8.26) by $\mathbf{T}^T \mathbf{K}$, replace $\mathbf{K} \mathbf{u}$ by \mathbf{f} , and pass the data to the RHS:

$$\boxed{\hat{\mathbf{K}} \hat{\mathbf{u}} = \hat{\mathbf{f}}, \quad \text{in which} \quad \hat{\mathbf{K}} = \mathbf{T}^T \mathbf{K} \mathbf{T}, \quad \hat{\mathbf{f}} = \mathbf{T}^T (\mathbf{f} - \mathbf{K} \mathbf{g}).} \quad (8.27)$$

Upon solving (8.27) for $\hat{\mathbf{u}}$, the complete displacement vector is recovered from (8.26). For the MFC (8.24) this technique gives the system

$$\begin{bmatrix} K_{11} & K_{12} & 0 & 0 & 0 & 0 \\ K_{12} & K_{22} + K_{66} & K_{23} & 0 & K_{56} & K_{67} \\ 0 & K_{23} & K_{33} & K_{34} & 0 & 0 \\ 0 & 0 & K_{34} & K_{44} & K_{45} & 0 \\ 0 & K_{56} & 0 & K_{45} & K_{55} & 0 \\ 0 & K_{67} & 0 & 0 & 0 & K_{77} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_7 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 + f_6 - 0.2K_{66} \\ f_3 \\ f_4 \\ f_5 - 0.2K_{56} \\ f_7 - 0.2K_{67} \end{bmatrix}. \quad (8.28)$$

See Exercise 8.2 for multiple non-homogeneous MFCs.

§8.3.4. *The General Case

For implementation in general-purpose programs the master-slave method can be described as follows. The degrees of freedoms in \mathbf{u} are classified into three types: independent or uncommitted, masters and slaves. (The uncommitted freedoms are those that do not appear in any MFC.) Label these sets as \mathbf{u}_u , \mathbf{u}_m and \mathbf{u}_s , respectively, and partition the stiffness equations accordingly:

$$\begin{bmatrix} \mathbf{K}_{uu} & \mathbf{K}_{um} & \mathbf{K}_{us} \\ \mathbf{K}_{um}^T & \mathbf{K}_{mm} & \mathbf{K}_{ms} \\ \mathbf{K}_{us}^T & \mathbf{K}_{ms}^T & \mathbf{K}_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{u}_u \\ \mathbf{u}_m \\ \mathbf{u}_s \end{bmatrix} = \begin{bmatrix} \mathbf{f}_u \\ \mathbf{f}_m \\ \mathbf{f}_s \end{bmatrix} \quad (8.29)$$

The MFCs may be written in matrix form as

$$\mathbf{A}_m \mathbf{u}_m + \mathbf{A}_s \mathbf{u}_s = \mathbf{g}_A, \quad (8.30)$$

where \mathbf{A}_s is assumed square and nonsingular. If so we can solve for the slave freedoms:

$$\mathbf{u}_s = -\mathbf{A}_s^{-1} \mathbf{A}_m \mathbf{u}_m + \mathbf{A}_s^{-1} \mathbf{g}_A \stackrel{\text{def}}{=} \mathbf{T} \mathbf{u}_m + \mathbf{g}, \quad (8.31)$$

Inserting into the partitioned stiffness matrix and symmetrizing

$$\begin{bmatrix} \mathbf{K}_{uu} & \mathbf{K}_{um} \mathbf{T} \\ \mathbf{T}^T \mathbf{K}_{um}^T & \mathbf{T}^T \mathbf{K}_{mm} \mathbf{T} \end{bmatrix} \begin{bmatrix} \mathbf{u}_u \\ \mathbf{u}_m \end{bmatrix} = \begin{bmatrix} \mathbf{f}_u - \mathbf{K}_{us} \mathbf{g} \\ \mathbf{f}_m - \mathbf{K}_{ms} \mathbf{g} \end{bmatrix} \quad (8.32)$$

It is seen that the misleading simplicity of the handworked examples is gone.

§8.3.5. *Retaining the Original Freedoms

A potential disadvantage of the master-slave method in computer work is that it requires a rearrangement of the original stiffness equations because $\hat{\mathbf{u}}$ is a subset of \mathbf{u} . The disadvantage can be annoying when sparse matrix storage schemes are used for the stiffness matrix, and becomes intolerable if secondary storage is used for that purpose.

With a bit of trickery it is possible to maintain the original freedom ordering. Let us display it for the example problem under (8.14). Instead of (8.15), use the *square* transformation

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ \tilde{u}_6 \\ u_7 \end{bmatrix}, \quad (8.33)$$

where \tilde{u}_6 is a *placeholder* for the slave freedom u_6 . The modified equations are

$$\begin{bmatrix} K_{11} & K_{12} & 0 & 0 & 0 & 0 & 0 \\ K_{12} & K_{22} + K_{66} & K_{23} & 0 & 0 & K_{56} & 0 \\ 0 & K_{23} & K_{33} & K_{34} & 0 & 0 & 0 \\ 0 & 0 & K_{34} & K_{44} & K_{45} & 0 & 0 \\ 0 & K_{56} & 0 & K_{45} & K_{55} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{67} & 0 & 0 & 0 & 0 & K_{77} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ \tilde{u}_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 + f_6 \\ f_3 \\ f_4 \\ f_5 \\ 0 \\ f_7 \end{bmatrix}, \quad (8.34)$$

which are submitted to the equation solver. If the solver is not trained to skip zero rows and columns, a one should be placed in the diagonal entry for the \tilde{u}_6 (sixth) equation. The solver will return $\tilde{u}_6 = 0$, and this placeholder value is replaced by u_2 . Note several points in common with the computer-oriented placeholder technique described in §3.4 to handle single-freedom constraints.

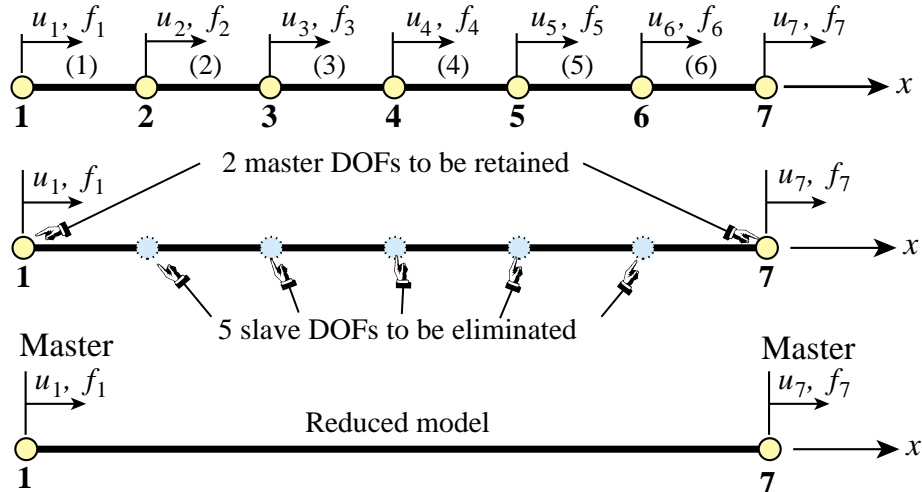


FIGURE 8.3. Model reduction of the example structure of Figure 8.2 to the end freedoms.

§8.3.6. Model Reduction by Kinematic Constraints

The congruent transformation equations (8.17) and (8.27) have additional applications beyond the master-slave method. An important one is *model reduction by kinematic constraints*. Through this procedure the number of DOF of a static or dynamic FEM model is reduced by a significant number, typically to 1% – 10% of the original number. This is done by taking a lot of slaves and a few masters. Only the masters are left after the transformation. The reduced model is commonly used in subsequent calculations as component of a larger system, particularly during design or in parameter identification.

Example 8.2. Consider the bar assembly of Figure 8.2. Assume that the only masters are the end motions u_1 and u_7 , as illustrated in Figure 8.2, and interpolate all freedoms linearly:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 5/6 & 1/6 \\ 4/6 & 2/6 \\ 3/6 & 3/6 \\ 2/6 & 4/6 \\ 1/6 & 5/6 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_7 \end{bmatrix}, \quad \text{or } \mathbf{u} = \mathbf{T}\hat{\mathbf{u}}. \quad (8.35)$$

The reduced-model equations are $\hat{\mathbf{K}}\hat{\mathbf{u}} = \mathbf{T}^T \mathbf{K} \mathbf{T} \hat{\mathbf{u}} = \mathbf{T}^T \mathbf{f} = \hat{\mathbf{f}}$, or in detail

$$\begin{bmatrix} \hat{K}_{11} & \hat{K}_{17} \\ \hat{K}_{17} & \hat{K}_{77} \end{bmatrix} \begin{bmatrix} u_1 \\ u_7 \end{bmatrix} = \begin{bmatrix} \hat{f}_1 \\ \hat{f}_7 \end{bmatrix}, \quad (8.36)$$

where

$$\begin{aligned} \hat{K}_{11} &= \frac{1}{36}(36K_{11} + 60K_{12} + 25K_{22} + 40K_{23} + 16K_{33} + 24K_{34} + 9K_{44} + 12K_{45} + 4K_{55} + 4K_{56} + K_{66}), \\ \hat{K}_{17} &= \frac{1}{36}(6K_{12} + 5K_{22} + 14K_{23} + 8K_{33} + 18K_{34} + 9K_{44} + 18K_{45} + 8K_{55} + 14K_{56} + 5K_{66} + 6K_{67}), \\ \hat{K}_{77} &= \frac{1}{36}(K_{22} + 4K_{23} + 4K_{33} + 12K_{34} + 9K_{44} + 24K_{45} + 16K_{55} + 40K_{56} + 25K_{66} + 60K_{67} + 36K_{77}), \\ \hat{f}_1 &= \frac{1}{6}(6f_1 + 5f_2 + 4f_3 + 3f_4 + 2f_5 + f_6), \quad \hat{f}_7 = \frac{1}{6}(f_2 + 2f_3 + 3f_4 + 4f_5 + 5f_6 + 6f_7). \end{aligned} \quad (8.37)$$

```
(* Model Reduction Example *)
ClearAll[K11,K12,K22,K23,K33,K34,K44,K45,K55,K56,K66,
  f1,f2,f3,f4,f5,f6];
K={{K11,K12,0,0,0,0,0},{K12,K22,K23,0,0,0,0},
  {0,K23,K33,K34,0,0,0},{0,0,K34,K44,K45,0,0},
  {0,0,0,K45,K55,K56,0},{0,0,0,0,K56,K66,K67},
  {0,0,0,0,0,K67,K77}}; Print["K=",K//MatrixForm];
f={f1,f2,f3,f4,f5,f6,f7}; Print["f=",f];
T={{6,0},{5,1},{4,2},{3,3},{2,4},{1,5},{0,6}}/6;
Print["T (transposed to save space)=",Transpose[T]//MatrixForm];
Khat=Simplify[Transpose[T].K.T];
fhat=Simplify[Transpose[T].f];
Print["Modified Stiffness:"];
Print["Khat(1,1)=",Khat[[1,1]],"\nKhat(1,2)=",Khat[[1,2]],
  "\nKhat(2,2)=",Khat[[2,2]] ];
Print["Modified Force:"];
Print["fhat(1)=",fhat[[1]]," fhat(2)=",fhat[[2]] ];
```

$$K = \begin{pmatrix} K11 & K12 & 0 & 0 & 0 & 0 & 0 \\ K12 & K22 & K23 & 0 & 0 & 0 & 0 \\ 0 & K23 & K33 & K34 & 0 & 0 & 0 \\ 0 & 0 & K34 & K44 & K45 & 0 & 0 \\ 0 & 0 & 0 & K45 & K55 & K56 & 0 \\ 0 & 0 & 0 & 0 & K56 & K66 & K67 \\ 0 & 0 & 0 & 0 & 0 & K67 & K77 \end{pmatrix}$$

$$f = \{f1, f2, f3, f4, f5, f6, f7\}$$

$$T \text{ (transposed to save space)} = \begin{pmatrix} 1 & \frac{5}{6} & \frac{2}{3} & \frac{1}{2} & \frac{1}{3} & \frac{1}{6} & 0 \\ 0 & \frac{1}{6} & \frac{1}{3} & \frac{1}{2} & \frac{2}{3} & \frac{5}{6} & 1 \end{pmatrix}$$

Modified Stiffness:

$$Khat(1,1) = \frac{1}{36} (36 K11 + 60 K12 + 25 K22 + 40 K23 + 16 K33 + 24 K34 + 9 K44 + 12 K45 + 4 K55 + 4 K56 + K66)$$

$$Khat(1,2) = \frac{1}{36} (6 K12 + 5 K22 + 14 K23 + 8 K33 + 18 K34 + 9 K44 + 18 K45 + 8 K55 + 14 K56 + 5 K66 + 6 K67)$$

$$Khat(2,2) = \frac{1}{36} (K22 + 4 K23 + 4 K33 + 12 K34 + 9 K44 + 24 K45 + 16 K55 + 40 K56 + 25 K66 + 60 K67 + 36 K77)$$

Modified Force:

$$fhat(1) = \frac{1}{6} (6 f1 + 5 f2 + 4 f3 + 3 f4 + 2 f5 + f6) \quad fhat(2) = \frac{1}{6} (f2 + 2 f3 + 3 f4 + 4 f5 + 5 f6 + 6 f7)$$
FIGURE 8.4. *Mathematica* script for the model reduction example of Figure 8.3.

This reduces the order of the FEM model from 7 to 2. A *Mathematica* script to do the reduction is shown in Figure 8.4. The key feature is that the masters are picked *a priori*, as the freedoms to be retained in the model for further use.

Remark 8.8. Model reduction can also be done by the static condensation method explained in Chapter 10. As its name indicates, condensation is restricted to static analysis. On the other hand, for such problems it is exact whereas model reduction by kinematic constraints generally introduces approximations.

§8.3.7. Assessment of the Master-Slave Method

What are the good and bad points of this constraint application method? It enjoys the advantage of being exact (except for inevitable solution errors) and of reducing the number of unknowns. The concept is also easy to explain and learn. The main implementation drawback is the complexity of the general case as can be seen by studying (8.29) through (8.32). The complexity is due to three factors:

1. The equations may have to be rearranged because of the disappearance of the slave freedoms. This drawback can be alleviated, however, through the placeholder trick outlined in §8.3.5.
2. An auxiliary linear system, namely (8.31), has to be assembled and solved to produce the transformation matrix \mathbf{T} and vector \mathbf{g} .
3. The transformation process may generate many additional matrix terms. If a sparse matrix storage scheme is used for \mathbf{K} , the logic for allocating memory and storing these entries can be difficult and expensive.

The level of complexity depends on the generality allowed as well as on programming decisions. At one extreme, if \mathbf{K} is stored as full matrix and slave freedom coupling in the MFCs is disallowed the logic is simple.² At the other extreme, if arbitrary couplings are permitted and \mathbf{K} is placed in secondary (disk) storage according to some sparse scheme, the complexity can become overwhelming.

Another, more subtle, drawback of this method is that it requires decisions as to which degrees of freedom are to be treated as slaves. This can lead to implementation and numerical stability problems. Although for disjointed constraints the process can be programmed in reliable form, in more general cases of coupled constraint equations it can lead to incorrect decisions. For example, suppose that in the example problem you have the following two MFCs:

$$\frac{1}{6}u_2 + \frac{1}{2}u_4 = u_6, \quad u_3 + 6u_6 = u_7. \quad (8.38)$$

For numerical stability reasons it is usually better to pick as slaves the freedoms with larger coefficients. If this is done, the program would select u_6 as slave freedoms from both constraints. This leads to a contradiction because having two constraints we must eliminate two slave degrees of freedom, not just one. The resulting modified system would in fact be inconsistent. Although this defect can be easily fixed by the program logic in this case, one can imagine the complexity burden if faced with hundreds or thousands of MFCs.

Serious numerical problems can arise if the MFCs are not independent. For example:

$$\frac{1}{6}u_2 = u_6, \quad \frac{1}{5}u_3 + 6u_6 = u_7, \quad u_2 + u_3 - u_7 = 0. \quad (8.39)$$

The last constraint is an exact linear combination of the first two. If the program blindly chooses u_2 , u_3 and u_7 as slaves, the modified system is incorrect because we eliminate three equations when in fact there are only two independent constraints.

Exact linear dependence, as in (8.39), can be recognized by a rank analysis of the \mathbf{A}_s matrix defined in (8.30). In inexact floating-point arithmetic, however, such detection may fail.³

² This is the case in model reduction, since each slave freedom appears in one and only one MFC.

³ The safest technique to identify dependencies is to do a singular-value decomposition (SVD) of \mathbf{A}_s . This can be, however, prohibitively expensive if one is dealing with hundreds or thousands of constraints.

The complexity of slave selection is in fact equivalent to that of automatically selecting kinematic redundancies in the Force Method of structural analysis. It has led implementors of programs that use this method to require masters and slaves be prescribed in the input data, thus transferring the burden to users.

The method is not generally extendible to nonlinear constraints without case by case programming. In conclusion, the master-slave method is useful when a few simple linear constraints are imposed by hand. As a general purpose technique for finite element analysis it suffers from complexity and lack of robustness. It is worth learning, however, because of the great importance of congruent transformations in *model reduction* for static and dynamic problems.

Notes and Bibliography

Multifreedom constraints are treated in several of the FEM books recommended in §1.7.5, notably Zienkiewicz and Taylor [279]. The master-slave method was incorporated to treat MFCs as part of the DSM developed at Boeing during the 1950s. It is first summarily described in the DSM-overview by Turner, Martin and Weikel [257, p. 212]. The implementation differs, however, from the one described here because the relation of FEM to energy methods was not clear at the time.

The master-slave method became popular through its adoption by the general-purpose NASTRAN code developed in the late 1960s [7] and early assessments of its potential [249]. The implementation unfortunately relied on user inputs to identify slave DOFs. Through this serious blunder the method gained a reputation for unreliability that persists to the present day.

The important application of master-slave to model reduction, which by itself justifies teaching the method, is rarely mentioned in FEM textbooks.

References

Referenced items have been moved to Appendix R.

Homework Exercises for Chapter 8

MultiFreedom Constraints I

EXERCISE 8.1 [C+N:20] The example structure of Figure 8.1 has $E^e A^e / L^e = 100$ for each element $e = 1, \dots, 6$. Consequently $K_{11} = K_{77} = 100$, $K_{22} = \dots = K_{66} = 200$, $K_{12} = K_{23} = \dots = K_{67} = -100$. The applied node forces are taken to be $f_1 = 1$, $f_2 = 2$, $f_3 = 3$, $f_4 = 4$, $f_5 = 5$, $f_6 = 6$ and $f_7 = 7$, which are easy to remember. The structure is subjected to one support condition: $u_1 = 0$ (a fixed left end), and to one MFC: $u_2 - u_6 = 1/5$.

Solve this problem using the master-slave method to process the MFC, taking u_6 as slave. Upon forming the modified system (8.27) apply the left-end support $u_1 = 0$ using the placeholder method of §3.4. Solve the equations and verify that the displacement solution and the recovered node forces including reactions are

$$\mathbf{u} = [0 \quad 0.270 \quad 0.275 \quad 0.250 \quad 0.185 \quad 0.070 \quad 0.140]^T$$

$$\mathbf{K}\mathbf{u} = [-27 \quad 26.5 \quad 3 \quad 4 \quad 5 \quad -18.5 \quad 7]^T \quad (\text{E8.1})$$

Use *Mathematica* or *Matlab* to do the algebra is recommended. For example, the *Mathematica* script of Figure E8.1 solves this Exercise.

```
(* Exercise 8.1 - Master-Slave Method *)
(* MFC: u2-u6 = 1/5 - slave: u6 *)
MasterStiffnessOfSixElementBar[kbar_]:=Module[
  {K=Table[0,{7},{7}], K[[1,1]]=K[[7,7]]=kbar;
  For [i=2,i<=6,i++,K[[i,i]]=2*kbar];
  For [i=1,i<=6,i++,K[[i,i+1]]=K[[i+1,i]]=-kbar];
  Return[K]];
FixLeftEndOfSixElementBar[Khat_,fhat_]:=Module[
  {Kmod=Khat,fmod=fhat}, fmod[[1]]=0; Kmod[[1,1]]=1;
  Kmod[[1,2]]=Kmod[[2,1]]=0; Return[{Kmod,fmod}]];

K=MasterStiffnessOfSixElementBar[100];
Print["Stiffness K=",K//MatrixForm];
f={1,2,3,4,5,6,7}; Print["Applied forces=",f];
T={{1,0,0,0,0,0},{0,1,0,0,0,0},{0,0,1,0,0,0},
  {0,0,0,1,0,0},{0,0,0,0,1,0},{0,1,0,0,0,0},
  {0,0,0,0,0,1}};
Print["Transformation matrix T=",T//MatrixForm];
g={0,0,0,0,0,-1/5,0};
Print["Constraint gap vector g=",g];
Khat=Simplify[Transpose[T].K.T]; fhat=Simplify[Transpose[T].(f-K.g)];
{Kmod,fmod}=FixLeftEndOfSixElementBar[Khat,fhat]; (* fix left end *)
Print["Modified Stiffness upon fixing node 1:",Kmod//MatrixForm];
Print["Modified RHS upon fixing node 1:",fmod];
umod=LinearSolve[Kmod,fmod];
Print["Computed umod (lacks slave u6)=",umod];
u=T.umod+g; Print["Complete solution u=",u];
Print["Numerical u=",N[u]];
fu=K.u; Print["Recovered forces K.u with reactions=",fu];
Print["Numerical K.u=",N[fu]];
```

FIGURE E8.1. Script for solving Exercise 8.1

EXERCISE 8.2 [C+N:25] As in the previous Exercise but applying the following three MFCs, two of which are non-homogeneous:

$$u_2 - u_6 = 1/5, \quad u_3 + 2u_4 = -2/3, \quad 2u_3 - u_4 + u_5 = 0. \quad (\text{E8.2})$$

Hints. Chose u_4 , u_5 and u_6 as slaves. Much of the script shown for Exercise 8.1 can be reused. The main changes are in the formation of \mathbf{T} and \mathbf{g} . If you are a *Mathematica* wizard (or willing to be one) those can be automatically formed by the statements listed in Figure E8.2.

```
sol=Simplify[Solve[{u2-u6==1/5, u3+2*u4==-2/3, 2*u3-u4+u5==0},{u4,u5,u6}]];
ums={u1,u2,u3,u4,u5,u6,u7}/.sol[[1]]; um={u1,u2,u3,u7};
T=Table[Coefficient[ums[[i]],um[[j]]],{i,1,7},{j,1,4}];
g=ums/.{u1->0,u2->0,u3->0,u4->0,u5->0,u6->0,u7->0};
Print["Transformation matrix T=",T//MatrixForm];
Print["Gap vector g=",g];
```

FIGURE E8.2. Script for partially solving Exercise 8.2.

If you do this, explain what it does and why it works. Otherwise form and enter \mathbf{T} and \mathbf{g} by hand. The numerical results (shown to 5 places) should be

$$\mathbf{u} = [0. \quad 0.043072 \quad -0.075033 \quad -0.29582 \quad -0.14575 \quad -0.15693 \quad -0.086928]^T, \quad (\text{E8.3})$$

$$\mathbf{Ku} = [-4.3072 \quad 16.118 \quad 10.268 \quad -37.085 \quad 16.124 \quad -8.1176 \quad 7.]^T.$$

EXERCISE 8.3 [A:25] Can the MFCs be pre-processed to make sure that no slave freedom in a MFC appears as master in another?

EXERCISE 8.4 [A:25] In the general case discussed in §8.4.4, under which condition is the matrix \mathbf{A}_s of (8.30) diagonal and thus trivially invertible?

EXERCISE 8.5 [A:25] Work out the general technique by which the unknowns need not be rearranged, that is, \mathbf{u} and $\hat{\mathbf{u}}$ are the same. Use “placeholders” for the slave freedoms. (Hint: use ideas of §3.4).

EXERCISE 8.6 [A/C:35] Is it possible to establish a slave selection strategy that makes \mathbf{A}_s diagonal or triangular? (This requires knowledge of matrix techniques such as pivoting.)

EXERCISE 8.7 [A/C:40] Work out a strategy that produces a well conditioned \mathbf{A}_s by selecting new slaves as linear combinations of finite element freedoms if necessary. (Requires background in numerical analysis and advanced programming experience in matrix algebra).