

# 21

## FEM Program for Space Trusses

## TABLE OF CONTENTS

	Page
§21.1. <b>Introduction</b>	21-3
§21.2. <b>Analysis Stages</b>	21-3
§21.3. <b>Analysis Support Modules</b>	21-3
§21.3.1. Assembling the Master Stiffness . . . . .	21-4
§21.3.2. Defining Boundary Conditions . . . . .	21-5
§21.3.3. Modifying the Master Stiffness Equations . . . . .	21-6
§21.3.4. Displacement Solution and Reaction Recovery . . . . .	21-8
§21.3.5. Flattening and Partitioning Node-Freedom Vectors . . . . .	21-9
§21.3.6. Internal Force Recovery . . . . .	21-10
§21.3.7. The Solution Driver . . . . .	21-11
§21.4. <b>Utility Print Modules</b>	21-12
§21.5. <b>Utility Graphic Modules</b>	21-13
§21.5.1. Plot Module Calls . . . . .	21-13
§21.5.2. Plot View Specification . . . . .	21-15
§21.6. <b>Example 1: Bridge Plane Truss Example</b>	21-15
§21.7. <b>Example 2: An Orbiting Truss Structure</b>	21-17
§21. <b>Notes and Bibliography</b> . . . . .	21-19
§21. <b>Exercises</b> . . . . .	21-20

### §21.1. Introduction

This Chapter presents a complete FEM program for analysis of space trusses. Why not start with plane trusses? Three reasons. First, plane truss code already appeared in Chapter 4, although most components were tailored to the example truss of Chapters 2–3. Second, the difference between 2D and 3D implementation logic for truss analysis is insignificant. Finally, space trusses are more interesting in engineering applications, particularly for orbiting structures in aerospace.

The overall organization of the space truss analysis program is diagrammed in Figure 21.1

The description is done in “bottom up” fashion. That means the element level modules are presented first, followed by midlevel modules, ending with the driver program. This means traversing the diagram of Figure 21.1 left to right and bottom to top.

The program includes some simple minded graphics, including animation. The graphic modules are provided in the posted Notebook but not described in detail. Data structures are explained along the way as they arise.

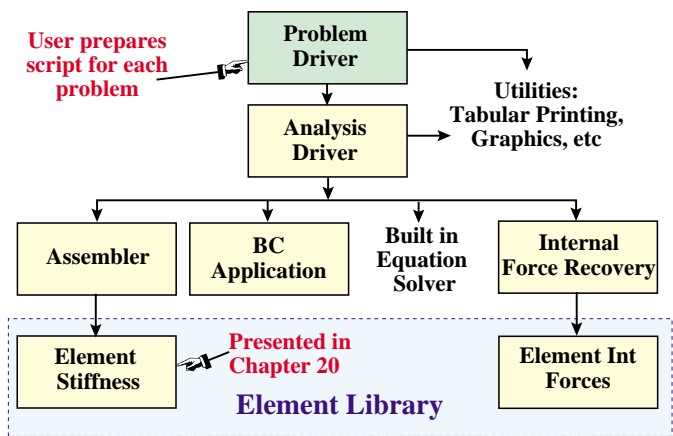


FIGURE 21.1. Overall organization of space truss analysis program.

### §21.2. Analysis Stages

The analysis of a structure by the Direct Stiffness Method involves three major stages: preprocessing or model definition, processing, and postprocessing. This is true for toy programs such as the one presented here, through huge commercial codes. Of course the stages here are very short.

The *preprocessing* portion of the space truss analysis is done by a driver script, which directly sets the data structures for the problem at hand.

The *processing* stage involves three steps:

- Assembly of the master stiffness matrix, with a subordinate element stiffness module.
- Modification of master stiffness matrix and node force vector for displacement boundary conditions.
- Solution of the modified equations for displacements. For the program presented here the built in *Mathematica* function `LinearSolve` is used.

Upon executing the processing steps, the displacements are available. The following *postprocessing* steps follow.

- Recovery of forces including reactions, done through a  $\mathbf{Ku}$  matrix multiplication.
- Computation of internal (axial) forces and stresses in truss members.
- Plotting deflected shapes and member stress levels.

```

SpaceTrussMasterStiffness[nodxyz_,elenod_,
  elemat_,elefab_,prcopt_]:=Module[
  {numele=Length[elenod],numnod=Length[nodxyz],neldof,
  e,eftab,ni,nj,i,j,ii,jj,ncoor,Em,A,options,Ke,K},
  K=Table[0,{3*numnod},{3*numnod}];
  For [e=1, e<=numele, e++, {ni,nj}=elenod[[e]];
  eftab={3*ni-2,3*ni-1,3*ni,3*nj-2,3*nj-1,3*nj};
  ncoor={nodxyz[[ni]],nodxyz[[nj]]};
  Em=elemat[[e]]; A=elefab[[e]]; options=prcopt;
  Ke=SpaceBar2Stiffness[ncoor,Em,A,options];
  neldof=Length[Ke];
  For [i=1, i<=neldof, i++, ii=eftab[[i]];
  For [j=i, j<=neldof, j++, jj=eftab[[j]];
  K[[jj,ii]]=K[[ii,jj]]+=Ke[[i,j]] ];
  ]; Return[K];
];
SpaceBar2Stiffness[ncoor_,Em_,A_,options_]:=Module[
  {x1,x2,y1,y2,z1,z2,x21,y21,z21,EA,numer,L,LL,LLL,Ke},
  {{x1,y1,z1},{x2,y2,z2}}=ncoor;{x21,y21,z21}={x2-x1,y2-y1,z2-z1};
  EA=Em*A; {numer}=options; LL=x21^2+y21^2+z21^2; L=Sqrt[LL];
  If [numer,{x21,y21,z21,EA,LL,L}=N[{x21,y21,z21,EA,LL,L}]];
  If [!numer, L=PowerExpand[L]]; LLL=Simplify[LL*L];
  Ke=(Em*A/LLL)*
  {{ x21*x21, x21*y21, x21*z21,-x21*x21,-x21*y21,-x21*z21},
  { y21*x21, y21*y21, y21*z21,-y21*x21,-y21*y21,-y21*z21},
  { z21*x21, z21*y21, z21*z21,-z21*x21,-z21*y21,-z21*z21},
  {-x21*x21,-x21*y21,-x21*z21, x21*x21, x21*y21, x21*z21},
  {-y21*x21,-y21*y21,-y21*z21, y21*x21, y21*y21, y21*z21},
  {-z21*x21,-z21*y21,-z21*z21, z21*x21, z21*y21, z21*z21}};
  Return[Ke]];

```

FIGURE 21.2. Master stiffness assembly module for a space truss. The element stiffness module SpaceBar2Stiffness, already discussed in Chapter 20, is listed for convenience.

### §21.3. Analysis Support Modules

We begin by listing here modules that support processing steps. These are put into separate cells for display and testing convenience.

#### §21.3.1. Assembling the Master Stiffness

Module SpaceTrussMasterStiffness, listed in Figure 21.2, assembles the master stiffness matrix of a space truss. It uses the element stiffness formation module SpaceBar2Stiffness discussed in the previous Chapter. That module is also listed in Figure 21.2 for convenience of the reader. The assembler is invoked by

$$K = \text{SpaceTrussMasterStiffness}[\text{nodxyz}, \text{elenod}, \text{elemat}, \text{elefab}, \text{prcopt}] \quad (21.1)$$

The arguments are

- nodxyz    Nodal coordinates placed in a node-by-node list  $\{\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \dots, \{x_n, y_n, z_n\}\}$ , where  $n$  is the total number of nodes of the truss.
- elenod    Element end nodes placed in an element-by-element list:  $\{\{i_1, j_1\}, \{i_2, j_2\}, \dots, \{i_e, j_e\}\}$ , where  $e$  is the total number of elements of the truss.

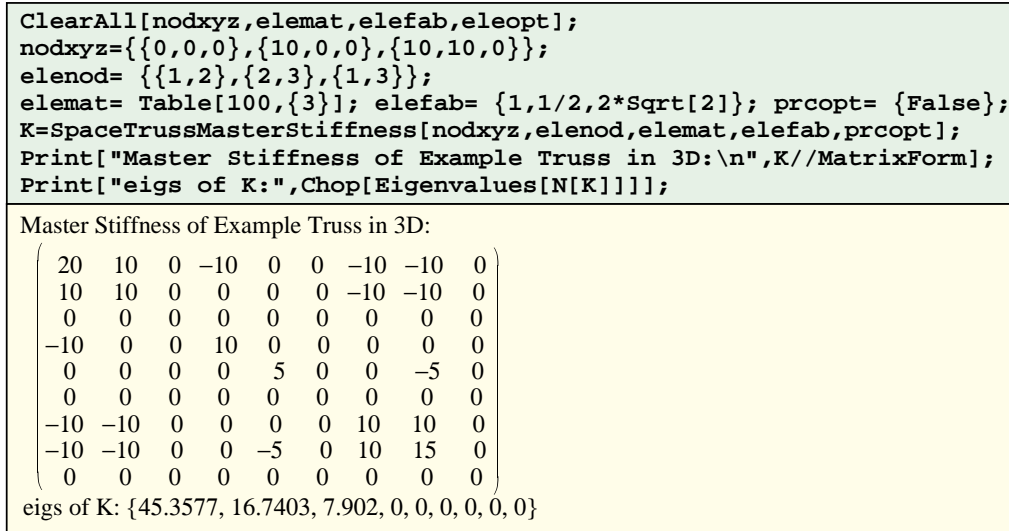


FIGURE 21.3. Testing the space truss assembler module.

- elemat** Element material properties. The only such property required for this analysis is the elastic modulus  $E$  of each element. These are put in an element-by-element list:  $\{E_1, E_2, \dots, E_e\}$ , where  $e$  is the total number of elements of the truss.
- elefab** Element fabrication properties. The only such property required for this analysis is the cross section area  $A$  of each element. These are put in an element-by-element list:  $\{A_1, A_2, \dots, A_e\}$ , where  $e$  is the total number of elements of the truss.
- prcopt** Processing option list. Only one option is required in the assembler and so **prcopt** is simply  $\{\text{numer}\}$ . Here **numer** is a logical flag set to  $\{\text{True}\}$  to tell the assembler to carry out element stiffness computations in floating-point arithmetic. Else set to  $\{\text{False}\}$  to keep computations in exact arithmetic or symbolic form.

The module returns the assembled stiffness, stored as a full  $3n \times 3n$  matrix, as function value.

Details of the assembly process are discussed in Chapter 25 for more general scenarios. Here we note that the module uses the freedom-pointer-table technique described in §3.4 for merging each element stiffness matrix into the master stiffness.

The assembler is tested by the script shown in the top cell of Figure 21.3. The script defines the example truss of Chapters 2–3 as a 3D structure with its three members placed in the  $\{x, y\}$  plane. See Figure 21.4. The axial rigidity values  $EA = 100, 50$  and  $200\sqrt{2}$  for elements 1, 2 and 3, respectively, have to be untangled because  $E$  is placed in **elemat** whereas  $A$  goes to **elefab**.

To split  $EA$  we take  $E = 100$  for the three elements. Thus **elemat** =  $\{100, 100, 100\} = \text{Table}[100, \{3\}]$  whereas **elefab** =  $\{1, 1/2, 2\sqrt{2}\}$ .

Running the assembler in exact arithmetic gives the  $9 \times 9$  master stiffness shown in the bottom cell of Figure 21.3. Taking its eigenvalues gives 6 zeros, which is the expected number in three dimensions.

### §21.3.2. Defining Boundary Conditions

The modification process described here refers to the application of displacement boundary condi-

tions on the master stiffness equations. These are assumed to be single-freedom constraints, either homogeneous such as  $u_{x3} = 0$ , or nonhomogeneous such as  $u_{z6} = -0.72$ .

Boundary condition data in FEM programs is usually specified in two levels: nodes at the first level, and freedoms assigned to that node at the second level. (The reason is that nodes are highly visible to casual users, whereas direct access to freedom numbers is difficult.) This space truss program is no exemption. This data is organized into two lists: node freedom tags and node freedom values. Their configuration is best specified through an example.

Consider again the example truss in 3D shown in Figure 21.4, which has 3 nodes and 9 freedoms. The node freedom tag list, internally called `nodtag`, is

$$\text{nodtag} = \{ \{1, 1, 1\}, \{0, 1, 1\}, \{0, 0, 1\} \} \quad (21.2)$$

Each first-level entry of this list pertains to a node. The second level is associated with freedoms: displacements in the  $x$ ,  $y$  and  $z$  directions. Freedom activity is marked by tag 0 or 1. A 1-tag means that the displacement is prescribed, whereas a 0-tag indicates that the force is known. Thus  $\{1, 1, 1\}$  for node 1 means that the node is fixed in the three directions.

The node freedom value list, internally called `nodval`, gives the prescribed value of the force or the displacement. For the example truss it is

$$\text{nodval} = \{ \{0, 0, 0\}, \{0, 0, 0\}, \{2, 1, 0\} \} \quad (21.3)$$

For node 1, the tag entry is  $\{1, 1, 1\}$  and the value entry is  $\{0, 0, 0\}$ . This says that  $u_{x1} = u_{y1} = u_{z1} = 0$ . For node 2 it says that  $f_{x2} = 0$  and  $u_{y2} = u_{z2} = 0$ . For node 3 it says that  $f_{x3} = 2$ ,  $f_{y3} = 1$  and  $u_{z3} = 0$ . The entries of `nodval` can be integers, floating point numbers, or symbols, whereas those in `nodtag` can only be 0 or 1.<sup>1</sup>

### §21.3.3. Modifying the Master Stiffness Equations

The modification of the master stiffness equations  $\mathbf{K}\mathbf{u} = \mathbf{f}$  for displacement BCs produces the modified system  $\hat{\mathbf{K}}\mathbf{u} = \hat{\mathbf{f}}$ . This is done by the two modules listed in Figure 21.5. These modules are not restricted to space trusses, and may in fact be used for more general problems.

The stiffness modifier is invoked by

$$\text{Kmod} = \text{ModifiedMasterStiffness}[\text{nodtag}, \mathbf{K}] \quad (21.4)$$

The arguments are:

- `nodtag`    A node by node list of freedom tags, as defined in the previous subsection.
- `K`         The master stiffness matrix  $\mathbf{K}$  produced by the assembler module.

<sup>1</sup> Other tag values may be implemented in more complicated programs to mark multifreedom constraints, for example.

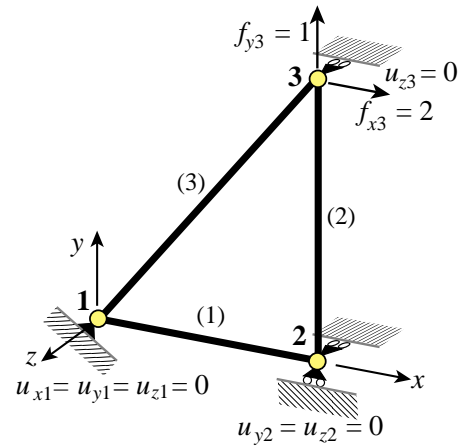


FIGURE 21.4. The example truss in three dimensions, used as module tester.

```

ModifiedMasterStiffness[nodtag_,K_] := Module[
  {i,j,k,n=Length[K],pdof,np,Kmod=K},
  pdof=PrescDisplacementDOFTags[nodtag]; np=Length[pdof];
  For [k=1,k<=np,k++, i=pdof[[k]];
    For [j=1,j<=n,j++, Kmod[[i,j]]=Kmod[[j,i]]=0];
    Kmod[[i,i]]=1];
  Return[Kmod]];

ModifiedNodeForces[nodtag_,nodval_,K_,f_] := Module[
  {i,j,k,n=Length[K],pdof,pval,np,d,c,fmod=f},
  pdof=PrescDisplacementDOFTags[nodtag]; np=Length[pdof];
  pval=PrescDisplacementDOFValues[nodtag,nodval]; c=Table[1,{n}];
  For [k=1,k<=np,k++, i=pdof[[k]]; c[[i]]=0];
  For [k=1,k<=np,k++, i=pdof[[k]]; d=pval[[k]];
    fmod[[i]]=d; If [d==0, Continue[]];
    For [j=1,j<=n,j++, fmod[[j]]-=K[[i,j]]*c[[j]]*d];
  ];
  Return[fmod]];

```

FIGURE 21.5. Modules to modify the master stiffness matrix and node force vector to apply the displacement boundary conditions.

The modified stiffness matrix, which has the same order as  $K$ , is returned as function value.

The force modifier is invoked by

$$\text{fmod} = \text{ModifiedNodeForces}[\text{pdof}, \text{pval}, K, \text{f}] \quad (21.5)$$

The arguments are:

- nodtag    The node freedom tag list defined in the previous subsection.
- nodval    The node freedom value list defined in the previous subsection.
- $K$         The master stiffness matrix  $\mathbf{K}$  produced by the assembler module (before modification). This is only used if at least one of the displacement BCs is non-homogeneous.
- $f$         The force vector before application of the displacement BCs.

The modified force vector, which has the same order as  $f$ , is returned as function value.

The modules are tested by the script listed in the top cell of Figure 21.6. It uses symbolic master stiffness equations of order 6. The test illustrates a not well known feature of *Mathematica*: use of Array function to generate subscripted symbolic arrays of one and two dimensions. The results, shown in the bottom cell of Figure 21.6, should be self explanatory.

**Remark 21.1.** On entry, the modification modules of Figure 21.5 build auxiliary lists `pdof` and `pval` to simplify the modification logic. `pdof` is a list of the prescribed degrees of freedom identified by their equation number in the master stiffness equations. For example, if freedoms 4, 7, 9 and 15 are specified, `pdof = {4,7,9,15}`. These indices are stored in ascending order. `pval` is a list of the prescribed displacement values listed in `pdof`. These lists are constructed by the modules listed in Figure 21.7. The calls are `pdof = PrescribedDOFTags[nodtag]` and `pval = PrescribedDOFValues[nodtag,nodval]`.

**Remark 21.2.** The logic of `ModifiedMasterStiffness` is straightforward. Construct `pdof`, then clear appropriate rows and columns of  $\mathbf{K}$  and place ones on the diagonal. Note the use of the *Mathematica* function `Length` to control loops: `np=Length[pdof]` sets `np` to the number of prescribed freedoms. Similarly

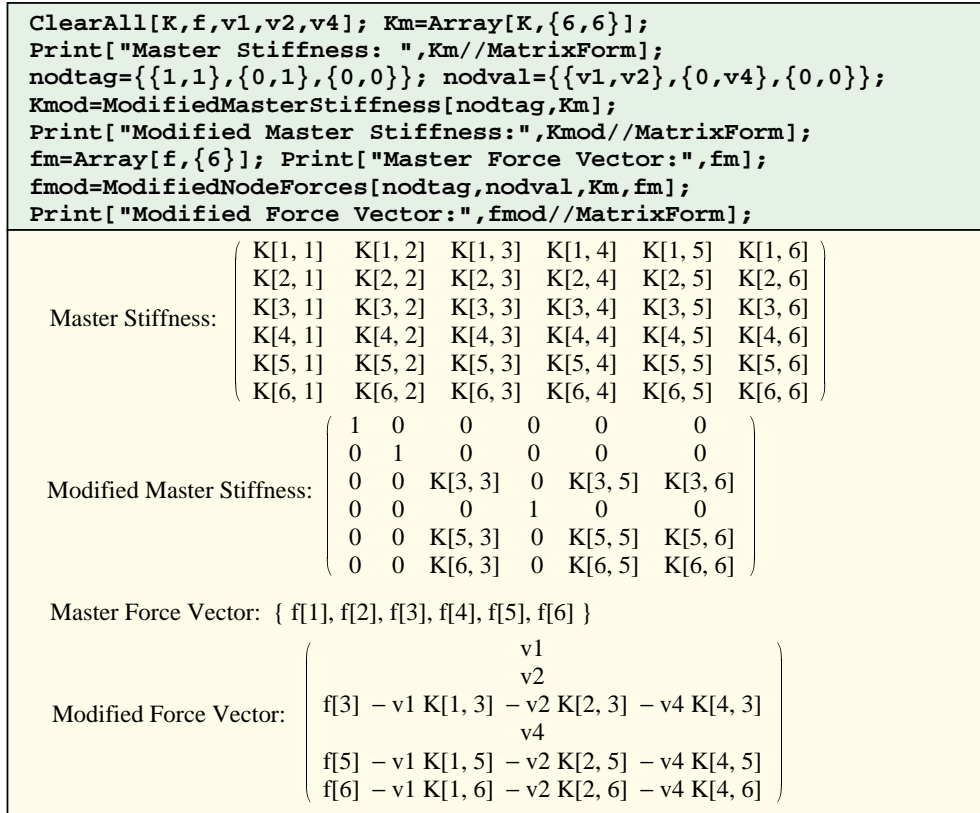


FIGURE 21.6. Testing the displacement boundary condition application modules.

`n=Length[K]` sets `n` to the order of the master stiffness matrix  $\mathbf{K}$ , which is used to bound the row and column clearing loop. These statements may be placed in the list that declares local variables.

**Remark 21.3.** `ModifiedNodalForces` has more complicated logic because it accounts for nonhomogeneous BCs. On entry it constructs `pdof` and `pval`. If nonzero values appear in `pval`, the original entries of `f` are modified as described in §4.1.2, and the end result is the effective force vector. Force vector entries corresponding to the prescribed displacement values are replaced by the latter in accordance with the prescription (4.13). If there are nonhomogeneous BCs it is important that the stiffness matrix provided as third argument be the master stiffness *before* modification by `ModifiedStiffnessMatrix`. This is because stiffness coefficients that are cleared by `ModifiedStiffnessMatrix` are needed for modifying the force vector.

#### §21.3.4. Displacement Solution and Reaction Recovery

The linear system  $\hat{\mathbf{K}} \mathbf{u} = \hat{\mathbf{f}}$ , where  $\hat{\mathbf{K}}$  and  $\hat{\mathbf{f}}$  are the modified stiffness and force matrices, respectively, is solved for displacements by a built-in linear algebraic solver. In Mathematica `LinearSolve` is available for this purpose:

$$\mathbf{u} = \text{LinearSolve}[\text{Kmod}, \text{fmod}] \quad (21.6)$$

At this point postprocessing begins. The node forces including reactions are obtained from  $\mathbf{f} = \mathbf{K} \mathbf{u}$ . This can be done simply as a matrix product:

$$\mathbf{f} = \mathbf{K} \cdot \mathbf{u} \quad (21.7)$$

```

PrescDisplacementDOFTags[nodtag_]:= Module [
  {j,n,numnod=Length[nodtag],pdof={},k=0,m},
  For [n=1,n<=numnod,n++, m=Length[nodtag[[n]]];
    For [j=1,j<=m,j++, If [nodtag[[n,j]]>0,
      AppendTo[pdof,k+j]];
    ]; k+=m;
  ]; Return[pdof]];
PrescDisplacementDOFValues[nodtag_,nodval_]:= Module [
  {j,n,numnod=Length[nodtag],pval={},k=0,m},
  For [n=1,n<=numnod,n++, m=Length[nodtag[[n]]];
    For [j=1,j<=m,j++, If [nodtag[[n,j]]>0,
      AppendTo[pval,nodval[[n,j]]]];
    ]; k+=m;
  ]; Return[pval]];

```

FIGURE 21.7. Modules to build auxiliary lists pdof and pval from node-by-node BC data.

where  $K$  is the original master stiffness matrix before modification, and  $u$  the displacement vector computed by `LinearSolve`.

### §21.3.5. Flattening and Partitioning Node-Freedom Vectors

In the analysis process one often needs displacement and force vectors in two different list formats. For example, the computed node displacement vector produced by (21.6) for the example truss in 3D is

$$u = \{0,0,0,0,0,0,0.4,-0.2,0\} \quad (21.8)$$

Following the terminology of `Mathematica` this will be called the *flat* form of the displacement vector. For postprocessing purposes (especially printing and plotting) it is convenient to rearrange to the node by node form

$$\text{noddis} = \{\{0,0,0\},\{0,0,0\},\{0.4,-0.2,0\}\} \quad (21.9)$$

This will be called the *node-partitioned* form of the displacement vector. Similar dual formats exist for the node force vector. In flat form this is called  $f$  and in node-partitioned form  $\text{nodfor}$ .

```

FlatNodePartVector[nv_]:=Flatten[nv];
NodePartFlatVector[nfc_,v_]:= Module [
  {i,k,m,n,nv={},numnod},
  If [Length[nfc]==0, nv=Partition[v,nfc]];
  If [Length[nfc]>0, numnod=Length[nfc]; m=0;
    nv=Table[0,{numnod}];
    For [n=1,n<=numnod,n++, k=nfc[[n]];
      nv[[n]]=Table[v[[m+i]],{i,1,k}];
      m+=k];
  ]; Return[nv]];

```

FIGURE 21.8. Utility modules to flatten and node-partition node-DOF vectors.

The utility modules listed in Figure 21.8 can be used to pass from one format to the other. To flatten the node-partitioned form of a vector, say  $nv$ , say

$$v = \text{FlatNodePartVector}[nv] \quad (21.10)$$

```

SpaceTrussIntForces[nodxyz_,elenod_,elemat_,elefab_,
noddis_,prcopt_]:= Module[{ numnod=Length[nodxyz],
numele=Length[elenod],e,ni,nj,ncoor,Em,A,options,ue,p},
p=Table[0,{numele}];
For [e=1, e<=numele, e++, {ni,nj}=elenod[[e]];
ncoor={nodxyz[[ni]],nodxyz[[nj]]};
ue=Flatten[{ noddis[[ni]],noddis[[nj]] }];
Em=elemat[[e]]; A=elefab[[e]]; options=prcopt;
p[[e]]=SpaceBar2IntForce[ncoor,Em,A,ue,options]
];
Return[p];

SpaceBar2IntForce[ncoor_,Em_,A_,ue_,options_]:= Module[
{x1,x2,y1,y2,z1,z2,x21,y21,z21,EA,numer,LL,pe},
{{x1,y1,z1},{x2,y2,z2}}=ncoor; {x21,y21,z21}={x2-x1,y2-y1,z2-z1};
EA=Em*A; {numer}=options; LL=x21^2+y21^2+z21^2;
If [numer,{x21,y21,z21,EA,LL}=N[{x21,y21,z21,EA,LL}]];
pe=(EA/LL)*(x21*(ue[[4]]-ue[[1]])+y21*(ue[[5]]-ue[[2]])+
+z21*(ue[[6]]-ue[[3]]));
Return[pe];

SpaceTrussStresses[elefab_,elefor_,prcopt_]:= Module[
{numele=Length[elefab],e,elesig}, elesig=Table[0,{numele}];
For [e=1, e<=numele, e++, elesig[[e]]=elefor[[e]]/elefab[[e]] ];
Return[elesig];

```

FIGURE 21.9. Modules to compute internal forces and stresses in a space truss.

To node-partition a flat vector  $v$  say

$$nv = \text{NodePartFlatVector}[nfc, v] \quad (21.11)$$

where  $nfc$  is the number of freedoms per node. For space trusses this is 3 so appropriate conversion calls are  $noddis = \text{NodePartFlatVector}[3, u]$  and  $nodfor = \text{NodePartFlatVector}[3, f]$ .

**Remark 21.4.** `FlatNodePartVector` can be directly done by the built-in function `Flatten` whereas `NodePartFlatVector` — for a fixed number of freedoms per node — can be done by `Partition`. The reason for the “wrappers” is to guide the conversion of *Mathematica* code to a lower level language such as C, where such built-in list functions are missing.

**Remark 21.5.** The additional code in `NodePartFlatVector` caters to the case where the number of freedoms can vary from node to node, in which case  $nfc$  is a list (not a number) called the *node freedom count*, hence the abbreviation. That facility is useful in more advanced courses.

### §21.3.6. Internal Force Recovery

The calculation of internal forces and stresses in a space truss involves computing axial forces in the bar elements. The modules that do those calculations given the displacement solution are listed in Figure 21.9. Module `SpaceTrussIntForces` computes the internal forces (axial forces) in all truss members. It is invoked by

$$\text{elefor} = \text{SpaceTrussIntForces}[\text{nodxyz}, \text{elenod}, \text{elemat}, \text{elefab}, \text{noddis}, \text{prcopt}] \quad (21.12)$$

```

ClearAll[nodxyz,elenod,elemat,elefab,noddis];
nodxyz={{0,0,0},{10,0,0},{10,10,0}}; elenod={{1,2},{2,3},{1,3}};
elemat= Table[100,{3}]; elefab= {1,1/2,2*Sqrt[2]};
noddis={{0,0,0}, {0,0,0}, {4/10,-2/10,0}}; prcopt={False};
elefor=SpaceTrussIntForces[nodxyz,elenod,elemat,elefab,noddis,prcopt];
Print["Int Forces of Example Truss:",elefor];
Print["Stresses:",SpaceTrussStresses[elefab,elefor,prcopt]];

```

```

Int Forces of Example Truss: { 0, -1, 2*Sqrt[2] }
Stresses: { 0, -2, 1 }

```

FIGURE 21.10. Test of the internal force recovery module.

Five of the arguments: `nodxyz`, `elenod`, `elemat`, `elefab` and `prcopt`, are the same used in the call (21.1) to the stiffness assembler. The additional argument, `noddis`, contains the computed node displacements arranged in node-partitioned form

$$\text{noddis} = \{ \{ \text{ux1}, \text{uy1}, \text{uz1} \}, \{ \text{ux2}, \text{uy2}, \text{uz2} \}, \dots \{ \text{uxn}, \text{uyn}, \text{uzn} \} \} \quad (21.13)$$

This form can be obtained from the computed displacement solution through the utility module (21.11). As function value `SpaceTrussIntForces` returns a list of element axial forces

$$\{ p_1, p_2 \dots p_e \} \quad (21.14)$$

`SpaceTrussIntForces` makes use of `SpaceBar2IntForce`, which computes the internal force in an individual bar element. This is invoked as `SpaceBar2IntForce`

$$p = \text{SpaceBar2IntForces}[\text{ncoor}, \text{Em}, \text{A}, \text{ue}, \text{options}] \quad (21.15)$$

Arguments `ncoor`, `Em`, `A` and `options` are the same as in the call to `SpaceBar2Stiffness` described in §20.2.2. The additional argument, `ue`, contains the flat list of the six element node displacements in the global system arranged as  $\{ \text{ux1}, \text{uy1}, \text{ux1}, \text{ux2}, \text{ux2}, \text{uz2} \}$ . The recovery equation is the subject of Exercise 21.4.

The last module in Figure 21.9 computes the member stresses simply by dividing the internal forces by the cross section areas. It is invoked as

$$\text{elesig} = \text{SpaceTrussStresses}[\text{elefab}, \text{elefor}, \text{prcopt}] \quad (21.16)$$

Here `elefab` and `prcopt` are as before, and `elefor` contains the element forces computed by `SpaceTrussIntForces`. The element axial stresses are returned as function value.

The statements of the top cell of Figure 21.9 exercise the internal force recovery for the example truss in 3D, requesting exact calculations. Array `p` is printed in the bottom cell. The axial forces of 0,  $-1$  and  $2\sqrt{2}$  agree with those determined in Chapter 3.

### §21.3.7. The Solution Driver

It is convenient to package the sequence of operations described in the previous subsections, namely assembly, modification, solution, force and stress recovery, into one module called the *solution driver*. This is listed in Figure 21.11. It is invoked by saying

$$\{ \text{noddis}, \text{nodfor}, \text{elefor}, \text{elesig} \} = \text{SpaceTrussSolution}[\text{nodxyz}, \text{elenod}, \text{elemat}, \text{elefab}, \text{nodtag}, \text{nodval}, \text{prcopt}] \quad (21.17)$$

All arguments: `nodxyz`, `elenod`, `elemat`, `elefab`, `nodtag`, `nodval` and `prcopt`, have been described in previous subsections. The module returns four lists:

- `noddis` Computed node displacement in node partitioned format.
- `nodfor` Recovered node forces including reactions in node partitioned format.
- `elefor` Element internal forces.
- `elesig` Element stresses.

Note that the listing of `SpaceTrussSolution` in Figure 21.11 has two commented out eigenvalue computations, one for the master stiffness `K` and one for the modified stiffness `Kmod`. Uncommenting those commands comes in handy when setting up and running a new problem if errors are detected.

```
SpaceTrussSolution[nodxyz_,elenod_,elemat_,elefab_,nodtag_,nodval_,
prcopt_] := Module[{K,Kmod,f,fmod,u,noddis,nodfor,elefor,elesig},
K=SpaceTrussMasterStiffness[nodxyz,elenod,elemat,elefab,prcopt];
(* Print["eigs of K=",Chop[Eigenvalues[N[K]]]]; *)
Kmod=ModifiedMasterStiffness[nodtag,K];
f=FlatNodePartVector[nodval];
fmod=ModifiedNodeForces[nodtag,nodval,K,f];
(* Print["eigs of Kmod=",Chop[Eigenvalues[N[Kmod]]]]; *)
u=LinearSolve[Kmod,fmod]; u=Chop[u]; f=Chop[K.u, 10.0^(-8)];
nodfor=NodePartFlatVector[3,f]; noddis=NodePartFlatVector[3,u];
elefor=Chop[SpaceTrussIntForces[nodxyz,elenod,elemat,elefab,
noddis,prcopt]];
elesig=SpaceTrussStresses[elefab,elefor,prcopt];
Return[{noddis,nodfor,elefor,elesig}];
];
```

FIGURE 21.11. The analysis driver module.

## §21.4. Utility Print Modules

Utility print modules are used to display input and output data in tabular form. The following six modules are provided in Cell 6 of the `SpaceTruss.nb` notebook.

To print the node coordinates in `nodxyz`:

$$\text{PrintSpaceTrussNodeCoordinates}[nodxyz, title, digits] \quad (21.18)$$

To print the element nodes in `elenod`, element materials in `elemat` and element fabrications in `elefab`:

$$\text{PrintSpaceTrussElementData}[elenod, elemat, elefab, title, digits] \quad (21.19)$$

To print the freedom activity data in `nodtag` and `nodval`:

$$\text{PrintSpaceTrussFreedomActivity}[nodtag, nodval, title, digits] \quad (21.20)$$

To print the node displacements in `noddis` (configured in node-partitioned form):

$$\text{PrintSpaceTrussNodeDisplacements}[noddis, title, digits] \quad (21.21)$$

To print the node forces in `nodfor` (configured in node-partitioned form):

$$\text{PrintSpaceTrussNodeForces}[nodfor, title, digits] \quad (21.22)$$

To print the element internal forces in `elefor` and element stresses in `elesig`:

```
PrintSpaceTrussElemForcesAndStresses[elefor,elesig,title,digits] (21.23)
```

In all cases, `title` is an optional character string to be printed as a title before the table; for example "Node coordinates of bridge truss". To eliminate the title, specify "" (two quote marks together).

The last argument of the print modules: `digits`, is optional. If set to `{d,f}` it specifies that floating point numbers are to be printed with room for at least `d` digits, with `f` digits after the decimal point.

If `digits` is specified as a void list: `{ }`, a preset default is used for `d` and `f`.

## §21.5. Utility Graphic Modules

Graphic modules that support preprocessing are placed in Cells 4 and 5 of the `SpaceTruss.nb` notebook. These display unlabeled elements, elements and nodes with labels, deformed shapes and element stress levels.

### §21.5.1. Plot Module Calls

To plot elements only:

```
PlotSpaceTrussElements[nodxyz,elenod,title,{view,aspect,{}}] (21.24)
```

To plot element and nodes with optional labeling:

```
PlotSpaceTrussElementsAndNodes[nodxyz,elenod,title,{view,aspect,labels}] (21.25)
```

To plot deformed shape of the truss under computed node displacements:

```
PlotSpaceTrussDeformedShape[nodxyz,elenod,noddis,amplif,box,title,
                             {view,aspect,colors}] (21.26)
```

To plot element axial stress levels using a coloring scheme:

```
PlotSpaceTrussStresses[nodxyz,elenod,elesig,sigfac,box,title,
                       {view,aspect,{}}] (21.27)
```

In the foregoing `nodxyz`, `elenod`, `noddis`, `noddfor` `elefor` and `elesig` have been described above. The other arguments are as follows.

- |                      |  |
|----------------------|--|
| <code>view</code>    | A list configured as a list of two 3D vectors: <code>{{Vhat1,Vhat2,Vhat3},{W1,W2,W3}}</code> . Vector <b>W</b> with global components <code>{W1,W2,W3}</code> specifies the view direction whereas vector $\hat{V}$ , with global components <code>{V1,V2,V3}</code> , specifies the up direction for the plot view, as discussed in §21.5.2. If a void list is provided for the argument, the default is <code>{{0,1,0},{0,0,1}}</code> ; this means that view direction is along the $-z$ axis whereas the up direction is the $y$ axis. |
| <code>aspect</code>  | Vertical-to-horizontal aspect ratio of the plot as it appears on a Notebook cell. Three possibilities. If set to $-1$ , the <i>Mathematica</i> default <code>Aspect-&gt;Automatic</code> is chosen. If set to zero, an aspect ratio is computed by the plot module as appropriate. If set to a positive number, the aspect ratio is set to that number.  |
| <code>labels</code>  | Only used in <code>PlotSpaceTrussElementsAndNodes</code> . A list with the following configuration: <code>{{nlabels,frn,fex,fey},{elabels,fre},{fntfam,fntsiz,fntwgt,fntslt}}</code> .   |
| <code>nlabels</code> | A logical flag. Set to <code>True</code> to get node labels in plot.   |
| <code>frn</code>     | Radius of circle drawn around each node expressed as percentage of plot size.  |

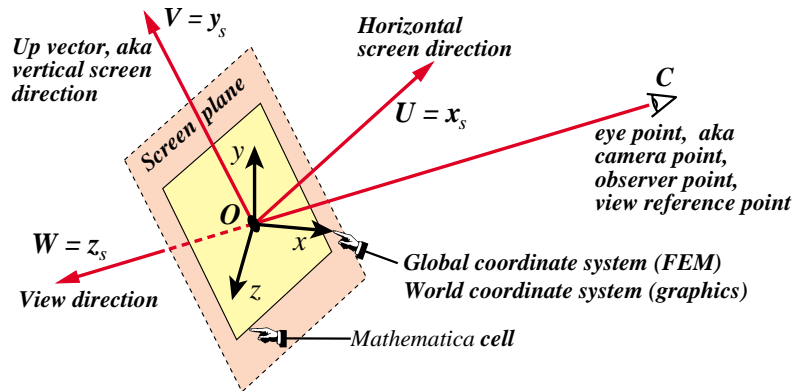


FIGURE 21.12. Plot view in 3D as mapping of world to screen coordinates.

- `fex` Horizontal eccentricity in points of node label from node location.
- `fey` Vertical eccentricity in points of node label from node location.
- `elabels` A logical flag. Set to `True` to get element labels in plot.
- `fre` Radius of circle drawn around each element number expressed as percentage of plot size. from node location.
- `fntfam` Font family used for labels, for example "Times"
- `fntsiz` Size in points of font used for labels; usually 10 through 14.
- `fntwgt` Font weight used for element labels. Typical settings are "Plain" or "Bold". Node labels are always drawn in boldface.
- `fntslt` Font slant used for element labels. Typical settings are "Plain", "Italics" or "Slanted". Node labels are always drawn in Plain.
- `amplif` The displacement amplification factor to be used by `PlotSpaceTrussDeformedShape`. Usually a value much larger than one (say 100 or 1000) is necessary to visualize displacements in actual structures. Becomes a list if the call is to draw several shapes (for example undeformed and deformed). For example setting `amplif` to `{ 0, 100 }` will draw two shapes: the undeformed configuration and one with magnification of 100. If more than one shape is to be drawn the `colors` specification comes in handy.
- `box` A list of points, specified by their  $\{x, y, z\}$  global coordinates, that forms a box that encloses the plot. Used in `PlotSpaceTrussDeformedShape` and `PlotSpaceTrussStresses`. The box is converted to a frame by the view projector. This is useful for various purposes, one being to do realistic animations by drawing a sequence of deformed shapes moving inside this box.
- `colors` Defines element colors to be used by `PlotSpaceTrussDeformedShape`, specified as lower case character string. Legal ones are "black", "red", "blue", "green" and "white". If the call is to draw several shapes (for example undeformed and deformed), this argument can be a list, such as `{ "black", "red" }`, in which case the colors are in one to one correspondence with the amplification values in `amplif`. If no color is specified, black is assumed.
- `sigfac` A stress scaling factor for `PlotSpaceTrussStresses`. Normally set to 1.

Because of the confusing way *Mathematica* handles plots, (some features do scale with plot size while others, such as font sizes, do not), some interactive experimentation with dimension specs seems inevitable.

### §21.5.2. Plot View Specification

Plotting 3D objects, such as the space trusses considered here, involves mapping the coordinates given in the FEM global system  $\{x, y, z\}$  into *screen coordinates*  $\{x_s, y_s, z_s\}$  in which  $z_s = 0$ . Objects are rendered using screen coordinates. Construction of this mapping is based on the view specification, which appears as argument of all plotting routines described in the foregoing subsection.

The viewing ingredients are shown in Figure 21.12. In computer graphics, the 3D space spanned by the FEM global system  $\{x, y, z\}$  is called the *world space* for obvious reasons. The screen coordinates  $\{x_s, y_s, z_s\}$  are defined by two vectors, which in computer graphics are typically identified as  $\mathbf{W}$  and  $\mathbf{V}$ : the *view direction* and *up direction*, respectively.

The *view direction* is the line defined by joining the eye position at  $C$  with the origin  $O$  of  $\{x, y, z\}$ . The *screen plane* passes through  $O$  and is normal to  $\mathbf{W}$ . Screen coordinates  $x_s$  and  $y_s$  are in the screen plane and going along the horizontal and vertical directions, respectively. In computer graphics the  $x_s$  and  $y_s$  directions are called  $\mathbf{U}$  and  $\mathbf{V}$ , respectively. When the plot is rendered in a *Mathematica* cell,  $\mathbf{U} \equiv x_s$  goes horizontally from left to right. Axes  $\{\mathbf{u} \equiv x_s, \mathbf{V} \equiv y_s, \mathbf{W} \equiv z_s\}$  form a RHS Cartesian coordinate system. Mappings from screen to pixel coordinates are handled by the plotting system, and need not be discussed here.

In the plot modules used here, the eye point  $C$  is assumed to be at infinity.<sup>2</sup> Thus only the view direction  $\mathbf{W}$ , as specified by three direction numbers, is used. For example, the specification  $\{1, 1, 1\}$  says that  $\mathbf{W}$  is the trisector of the  $\{x, y, z\}$  octant. To define the up direction  $\mathbf{V} \equiv y_s$  one has to enter a second “indication” vector:  $\hat{\mathbf{V}}$ , which must not be parallel to  $\mathbf{W}$ . Since  $\hat{\mathbf{V}}$  is not necessarily normal to  $\mathbf{W}$ ,  $\mathbf{V}$  is constructed by orthogonalization:  $\mathbf{V} = \hat{\mathbf{V}} - (\hat{\mathbf{V}}^T \mathbf{W}_n) \mathbf{W}_n$ , where  $\mathbf{W}_n$  is  $\mathbf{W}$  normalized to length one. For example if  $\hat{\mathbf{V}}$  and  $\mathbf{W}$  are specified by view argument  $\{\{0, 1, 0\}, \{2, 2, 1\}\}$ , then  $\mathbf{V} = [0, 1, 0] - ([0, 1, 0]^T [2/3, 2/3, 1/3] / [2/3, 2/3, 1/3]^T [2/3, 2/3, 1/3]) [2/3, 2/3, 1/3] = [0, 1, 0] - (2/3) [2/3, 2/3, 1/3] = [-4/9, 5/9, -2/9]$ . Note that  $\mathbf{V}^T \mathbf{W} = 0$ .

## §21.6. Example 1: Bridge Plane Truss Example

This example deals with the analysis of the 6-bay bridge truss problem defined in Figure 21.13. This truss has 12 nodes and 17 elements. It is contained in the  $\{x, y\}$  plane and can only move in that plane. It is fixed at node 1 and on rollers at node 12.

The driver is listed in Figure 21.14. Preprocessing statements appear on top, with light green background. These define the problem through specification of the following data structures.<sup>3</sup>

NodeCoordinates	Same configuration as nodxyz
ElemNodes	Same configuration as elenod
ElemMaterials	Same configuration as elemat
ElemFabrications	Same configuration as elefab. This list is built up from four repeating cross sectional areas: Abot, Atop, Abat and Adia, for the areas of bottom longerons, top longerons, battens and diagonals, respectively.

<sup>2</sup> Graphics where the eye point  $C$  is at a finite distance produce *perspective plots*. The Graphics3D system of *Mathematica* allows perspective plotting. However the plots described here use only the 2D graphics subset.

<sup>3</sup> Note the use of longer mnemonic names in the problem driver. For example NodeCoordinates instead of nodxyz. This simplifies the preparation of problem solving assignments since driver scripts are more self-documenting. It also helps grading returned assignments.

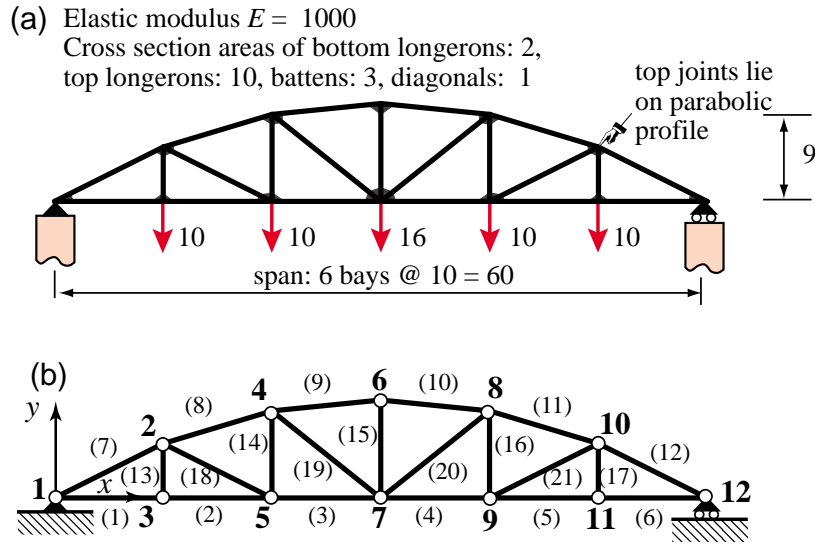


FIGURE 21.13. Six-bay bridge plane truss used as example problem: (a) truss structure showing supports and applied loads; (b) finite element idealization as pin-jointed truss.

- NodeDOFTags** Same configuration as `nodtag`. Initialized to  $\{0,0,1\}$  for all nodes on creation so as to fix all  $z$  displacements. Then the support conditions in the  $\{x, y\}$  plane are specified at supported nodes. For example, `NodeDOFTags[[1]] = {1,1,1}` says that the three node displacement components  $u_{x1}$ ,  $u_{y1}$  and  $u_{z1}$  of node 1 are specified.
- NodeDOFValues** Same configuration as `nodval`. Initialized to  $\{0,0,0\}$  on creation for all nodes. Then the value of nonzero applied loads is set at nodes 3, 5, 7, 9 and 11. For example `NodeDOFValues[[7]] = {0,-16,0}` specifies  $f_{x7} = 0$ ,  $f_{y7} = -16$  and  $u_{z7} = 0$ .

The input data structures can be shown in tabular form for convenient inspection using print utility modules. Printed tables are shown on the left of Figure 21.15.

Running the solution module returns computed displacements, node forces including reactions, internal forces and member stress. These are printed with utility modules. These results are shown on the right of Figure 21.15.

Output plot results are collected in Figure 21.16.

```

NodeCoordinates={ {0,0,0}, {10,5,0}, {10,0,0}, {20,8,0}, {20,0,0}, {30,9,0},
                  {30,0,0}, {40,8,0}, {40,0,0}, {50,5,0}, {50,0,0}, {60,0,0} };
ElemNodes= { {1,3}, {3,5}, {5,7}, {7,9}, {9,11}, {11,12},
              {1,2}, {2,4}, {4,6}, {6,8}, {8,10}, {10,12},
              {2,3}, {4,5}, {6,7}, {8,9}, {10,11},
              {2,5}, {4,7}, {7,8}, {9,10} };
PrintSpaceTrussNodeCoordinates[NodeCoordinates,"Node coordinates:",{}];
numnod=Length[NodeCoordinates]; numele=Length[ElemNodes];
Em=1000; Abot=2; Atop=10; Abat=3; Adia=1;
ElemMaterials= Table[Em,{numele}];
ElemFabrications= {Abot,Abot,Abot,Abot,Abot,Abot,Abot,Atop,Atop,Atop,Atop,
                   Atop,Atop,Abat,Abat,Abat,Abat,Abat,Abat,Adia,Adia,Adia,Adia};
PrintSpaceTrussElementData[ElemNodes,ElemMaterials,ElemFabrications,
                           "Element data:",{}];
ProcessOptions= {True};

view={ {0,1,0}, {0,0,1} };
labels= { {True,0.06,-1.5,1.5}, {True,0.12}, {"Times",11,"Roman"} };
PlotSpaceTrussElementsAndNodes[NodeCoordinates,ElemNodes,
                                "bridge mesh",{view,-1,labels}];

NodeDOFTags= Table[{0,0,1},{numnod}];
NodeDOFValues=Table[{0,0,0},{numnod}];
NodeDOFValues[[3]]={0,-10,0}; NodeDOFValues[[5]]={0,-10,0};
NodeDOFValues[[7]]={0,-16,0};
NodeDOFValues[[9]]={0,-10,0}; NodeDOFValues[[11]]={0,-10,0};
NodeDOFTags[[1]]={1,1,1}; (* fixed node 1 *)
NodeDOFTags[[numnod]]={0,1,1}; (* hroller @ node 12 *)
PrintSpaceTrussFreedomActivity[NodeDOFTags,NodeDOFValues,
                              "DOF Activity:",{}];

{NodeDisplacements,NodeForces,ElemForces,ElemStresses}=
SpaceTrussSolution[ NodeCoordinates,ElemNodes,ElemMaterials,
                   ElemFabrications,NodeDOFTags,NodeDOFValues,ProcessOptions ];

PrintSpaceTrussNodeDisplacements[NodeDisplacements,
                                 "Computed node displacements:",{}];
PrintSpaceTrussNodeForces[NodeForces,
                          "Node forces including reactions:",{}];
PrintSpaceTrussElemForcesAndStresses[ElemForces,ElemStresses,
                                      "Int Forces and Stresses:",{}];

view={ {0,1,0}, {0,0,1} }; box={ {0,-4,0}, {60,-4,0}, {60,10,0}, {0,10,0} };
PlotSpaceTrussDeformedShape[NodeCoordinates,ElemNodes,NodeDisplacements,
                             {0,1},box,"deformed shape (unit magnif)",{view,-1,{"black","blue"}}];
PlotSpaceTrussStresses[NodeCoordinates,ElemNodes,ElemStresses,1,box,
                       "axial stresses in truss members",{view,0,labels}];

```

FIGURE 21.14. Driver script for analysis of the 6-bay plane bridge truss. Preprocessing statements in light green background. Processing and postprocessing statements in light blue.

## §21.7. Example 2: An Orbiting Truss Structure

To be included in the final version of the Chapter.

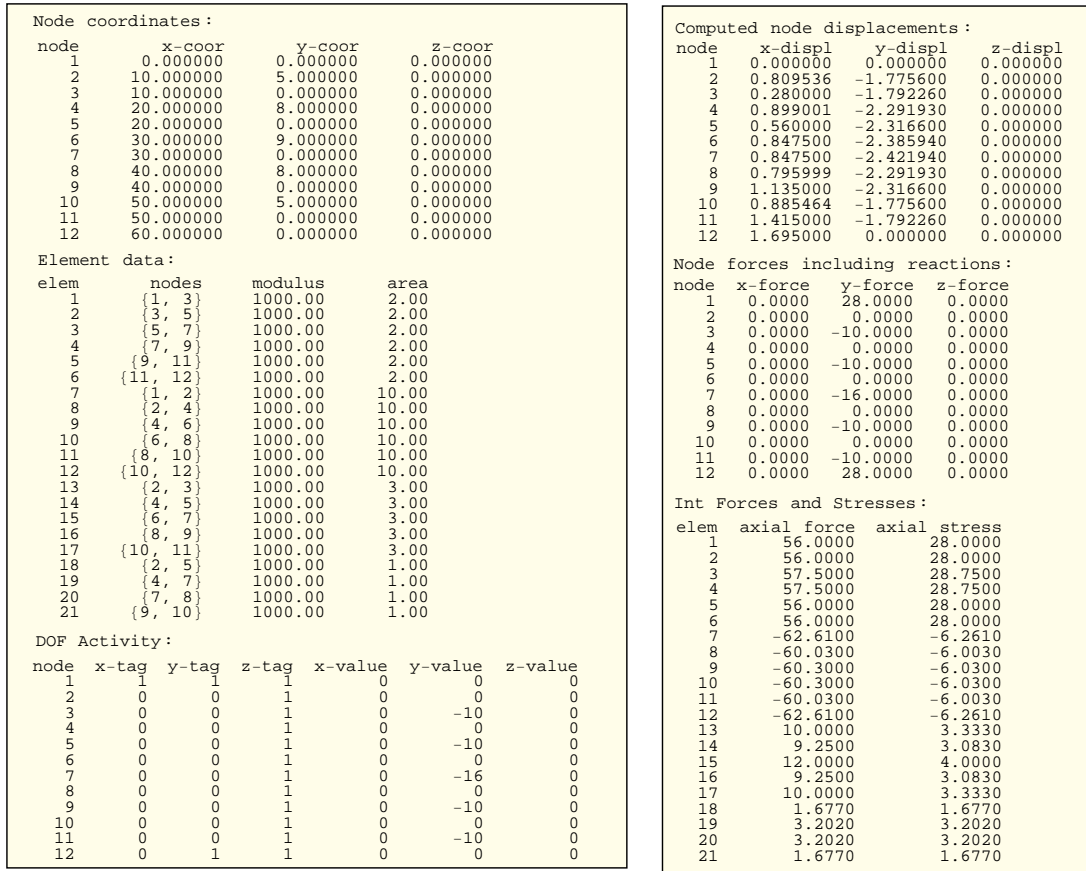


FIGURE 21.15. Bridge truss example: tabular printed output. On the left: node, element and freedom data. On the right: computed displacements, node forces, internal forces and member stresses.

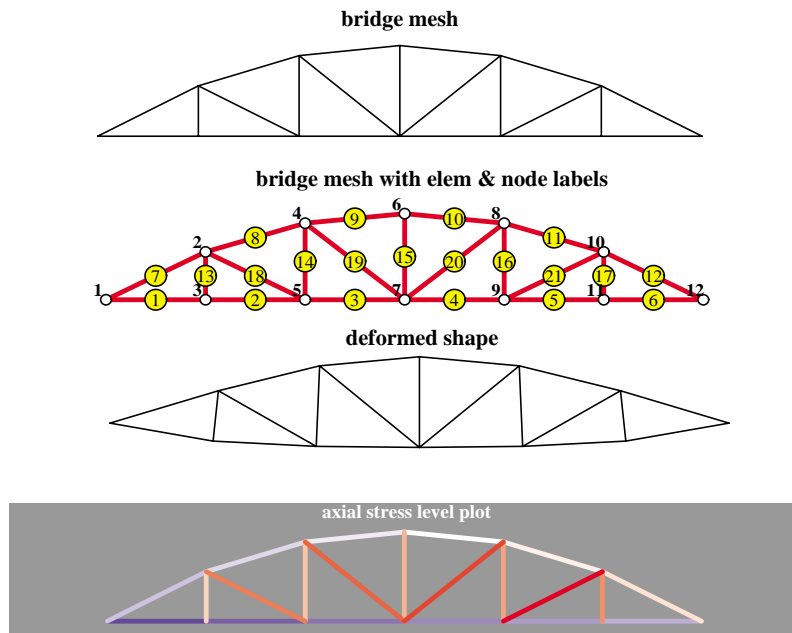


FIGURE 21.16. Bridge truss example: graphic output collected in one figure.

### Notes and Bibliography

The dominant philosophy in FEM implementation is to construct general purpose programs that can solve a wide range of problems. For example, static and dynamic response of arbitrary structures with linear or nonlinear behavior. This path was naturally taken once the Direct Stiffness Method became widely accepted in the mid 1960s. It is reflected in the current crop of commercial FEM programs. Their source code by now has reached into millions of lines.

These codes do have a place in undergraduate engineering education, starting at the junior level. At this level students should be taught rudiments of modeling and how to use those black box programs as tools. This exposure provides also basic knowledge for capstone senior projects that require finite element analysis.

At the graduate level, however, students should understand what goes on behind the scene. But access to innards of commercial programs is precluded (and even if it were, it would be difficult to follow given their complexity). The philosophy followed here is to use special purpose codes written in a high level language. These may be collectively called *grey level codes*. A high level language such as *Mathematica* conceals utility operations such as matrix products, linear solvers and graphics, but permits the application code logic to be seen and studied.

As a result a complete FEM program is tiny (typically a few hundreds lines), it can be built and debugged in a few hours, and may be understood as a whole by one person. On the down side of course these toy programs can do only very limited problems, but for instructional use simplicity outweighs generality.

### Homework Exercises for Chapter 21

#### FEM Program for Space Trusses

**EXERCISE 21.1** [D:10] The logic of `SpaceTrussMasterStiffness` cannot be used for structures other than space trusses. Justify this assertion.

**EXERCISE 21.2** [D:10] The logic of `ModifiedMasterStiffness` and `ModifiedNodeForces` is not restricted to space trusses, but can be used for any FEM program that stores  $\mathbf{K}$  and  $\mathbf{f}$  as full arrays. Justify this assertion.

**EXERCISE 21.3** [D:20] The logic of `PrescDisplacementDOFTags` and `PrescDisplacementDOFValues` is not restricted to a fixed number of DOF per node. Justify this assertion.

**EXERCISE 21.4** [A:15] Show that the longitudinal elongation of a space bar can be computed directly from the global displacements  $u_{x1}, u_{y1}, \dots, u_{z2}$  from

$$d = (x_{21} u_{x21} + y_{21} u_{y21} + z_{21} u_{z21})/\ell, \quad (\text{E21.1})$$

in which  $x_{21} = x_2 - x_1$ ,  $u_{x21} = u_{x2} - u_{x1}$ , etc, and  $\ell$  is the bar length. Hence justify the formula used in module `SpaceBar2IntForce` listed in Figure 21.9 to recover the axial force  $p = (EA/\ell)d$ .

**EXERCISE 21.5** [C:25] Analyze the structure shown in Figure E21.1. This is a pin-jointed truss model of a 200-in-high (5m) transmission tower originally proposed by Fox and Schmit in 1964 [96] as a test for early automated-synthesis codes based on FEM. It became a standard benchmark for structural optimization.

The truss has 10 joints (nodes) and 25 members (elements). The truss geometry and node numbering are defined in Figure E21.1(a). Joints 1 and 2 at the top of the tower lie on the  $\{x, z\}$  plane. The truss (but not the loads) is symmetric about the  $\{y, z\}$  and  $\{x, z\}$  planes. Figure E21.1(b) gives the element numbers.

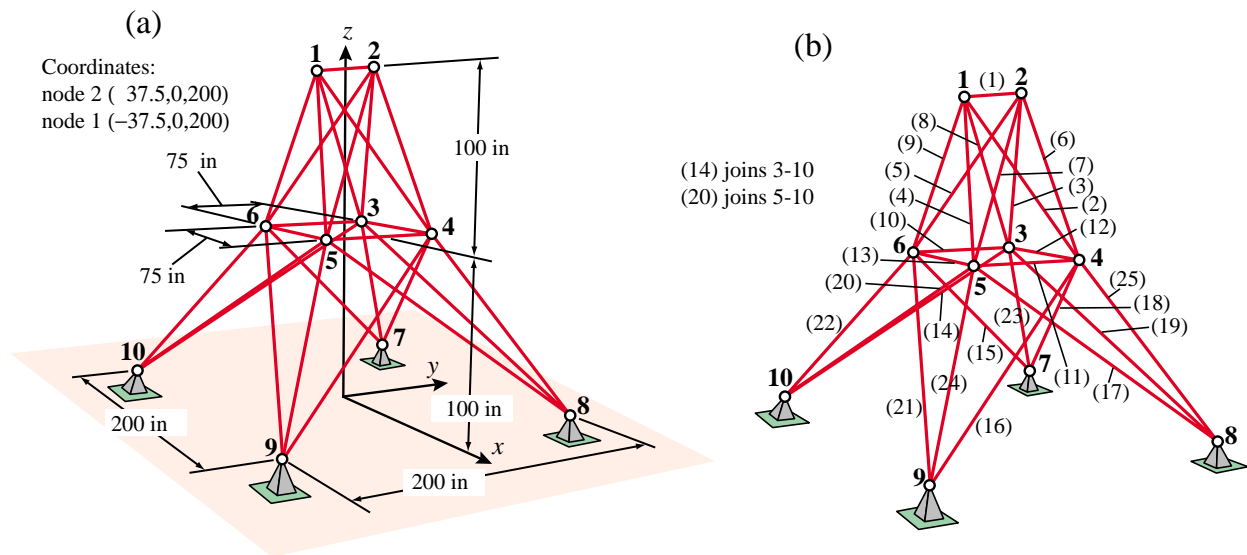


FIGURE E21.1. 25-member space truss model of a transmission tower. (a): Geometry definition and node numbers; (b) element numbers. For member properties and loads see Tables E21.1 and E21.2.

**Table E21.1** Cross section areas of transmission tower members

Element	A (sq in)	Element	A (sq in)	Element	A (sq in)
1	0.033	10	0.010	19	1.760
2	2.015	11	0.010	20	1.760
3	2.015	12	0.014	21	1.760
4	2.015	13	0.014	22	2.440
5	2.015	14	0.980	23	2.440
6	2.823	15	0.980	24	2.440
7	2.823	16	0.980	25	2.440
8	2.823	17	0.980		
9	2.823	18	1.760		

**Table E21.2** Applied load case for transmission tower

Node	x-load (lb)	y-load (lb)	z-load (lb)
1	1000	10000	−5000
2	0	10000	−5000
3	500	0	0
6	500	0	0

Applied forces at all other nodes are zero.  
Own-weight loads not considered.

The members are aluminum tubes with the cross sections listed in Table E21.1.<sup>4</sup> The modulus of elasticity is  $E = 10^7$  psi for all members. The specific weight is  $0.1 \text{ lb/in}^3$ . The applied load case to be studied is given in Table E21.2.

Analyze the transmission tower using the program provided in the `SpaceTruss.nb` Notebook (downloadable from Chapter 21 Index). Results to report: driver program cell, node displacements and element stresses. (More details on HW assignment sheet.) Note: as a quick check on model preparation, the total weight of the tower should be 555.18 lb.

<sup>4</sup> Data taken from an optimal design reported by Venkayya, Khot and Reddy in 1968 [253].