30

Linearized Prebuckling: Implementation

TABLE OF CONTENTS

		Page
§30.1	**Introduction**	30–3
§30.2	**Analysis Modules**	30–3
§30.2.1	Finite Element Library	30–4
§30.2.2	PFabule Master Stiffness Assembler	30–5
§30.2.3	PFabule Boundary Condition Applicators	30–7
§30.2.4	Initial Stress Computation	30–8
§30.2.5	Stability Eigenmatrices	30–9
§30.2.6	Stability Eigensystem Solution	30–10
§30.2.7	Analysis Driver	30–12
§30.3	**Print Utilities**	30–12
§30.4	**Plot Utilities**	30–16
§30.4.1	Get Display Channel	30–16
§30.4.2	Color Name Mapping	30–16
§30.4.3	Plot Cubic Shape	30–17
§30.4.4	Cubic Shape Minimum Bounding Frame	30–17
§30.4.5	Plot Rectilinear Spring	30–18
§30.4.6	Plot Circular Spring	30–20
§30.4.7	Plot FEM Model	30–21
§30.4.8	Plot Deflected Shape	30–23

§30.1. Introduction

This Chapter presents the `PFabule` program. This is an acronym for *Plane Frame Analysis of Buckling Using Linearized Eigenproblem*. As is name indicates, its main objective is to carry out finite element buckling analysis of plane frame structures using the LPB simplified model.

The basic element is a plane beam-column with 3 degrees of freeodm (DOF) per node. The program also include plane bar elements with 2 DOF per node, as well as extensional and torsional linear springs that can be attached to a node. As a result the code can do plane trusses as well as spring-propped columns, the latter allowing direct comparision with results of the equilibrium analysis carried out by hand in Chapter 29.

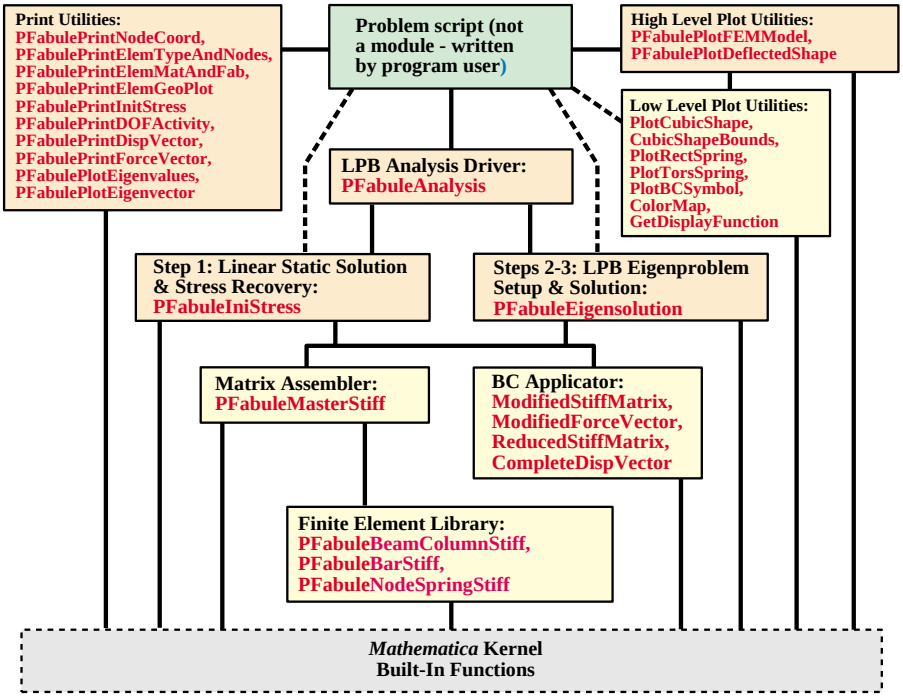

FIGURE 30.1. Configuration of the LPB frame analysis program `PFabule`. Module names in red. Green box: problem script written by user. Orange boxes: modules that may be called from the script. Yellow boxes: lower level modules (usually) not accessed from problem script. Full and dashed lines indicate normal and potential reference calls, respectively. Naming convention: modules whose name starts with `PFabule` are exclusive to this program; others are "universal" in the sense that they can be used elsewhere.

§30.2. Analysis Modules

The configuration of `Pfabule` is shown in Figure 30.1. The figure legend specifies color and line-drawing conventions. The program includes three finite element types:

(1) A two-node plane beam-column with 3 DOF per node. A prismatic Bernoulli-Euler (BE) beam model combined with a standard two-node bar. Its geometric stiffness is derived in a previous Chapter.

(2) A two-node prismatic bar with 2 DOF per node. Its geometric stiffness is derived in Chapter 8.

(3) A node spring element that can be used to specify extensional as well as torsional (rotational) springs, which are attached to structural nodes. This element has null geometric stiffness.

```
PFabuleBarStiff[enodXY_,Em_,A0_,s0_,options_]:=Module[{X1,Y1,X2,Y2,
 X21,Y21,LL,L,B,BB,m=Length[options],numer=False,srule={},KeM,KeG},
  If [m>=1,numer=options[[1]]]; If [m>=2,srule=options[[2]]];
 {{X1,Y1},{X2,Y2}}=enodXY; {X21,Y21}={X2-X1,Y2-Y1};
  LL=X21^2+Y21^2; L=Sqrt[LL];
  If [!numer,{L,LL}=Simplify[{L,LL},srule]];
  B={-X21,-Y21,X21,Y21}; KeM=(Em*A0/L)*Outer[Times,B,B]/LL;
  KeG=(s0*A0/L)*{{1,0,-1,0},{0,1,0,-1},{-1,0,1,0},{0,-1,0,1}};
  If [numer,{KeM,KeG}=N[{KeM,KeG}],
           {KeM,KeG}=Simplify[{KeM,KeG},srule]];
  Return[{KeM,KeG}]];

PFabuleBeamColStiff[enodXY_,Em_,{A0_,Izz0_},s0_,options_]:=
   Module[{X1,X2,Y1,Y2,X21,Y21,XX,YY,XY,EA,EI,P=s0*A0,L,LL,LLL,
   LLLL,LLLLL,EALL,EILL,EIALL,EAXIY,EIXAY,
   m=Length[options],numer=False,srule={},KeM,KeG},
   If [m>=1,numer=options[[1]]]; If [m>=2,srule=options[[2]]];
  {{X1,Y1},{X2,Y2}}=enodXY; {X21,Y21}={X2-X1,Y2-Y1};
   EA=Em*A0; EI=Em*Izz0; LL=X21^2+Y21^2; L=Sqrt[LL];
   If [!numer,{L,LL}=Simplify[{L,LL},srule]];
   XX=X21^2; XY=X21*Y21; YY=Y21^2; LLL=LL*L; LLLL=LL*LL;
   EALL=EA*LL; EILL=EI*LL; EIALL=12*EI-EALL; LLLLL=LLL*LL;
   EAXIY=EALL*XX+12*EI*YY; EIXAY=12*EI*XX+EALL*YY;
   KeM={{EAXIY,-EIALL*XY,-6*EILL*Y21,-EAXIY,EIALL*XY,-6*EILL*Y21},
   {-EIALL*XY,EIXAY,6*EILL*X21,EIALL*XY,-EIXAY,6*EILL*X21},
   {-6*EILL*Y21,6*EILL*X21,4*EILL*LL,6*EILL*Y21,-6*EILL*X21,2*EILL*LL},
   {-EAXIY,EIALL*XY,6*EILL*Y21,EAXIY,-EIALL*XY,6*EILL*Y21},
   {EIALL*XY,-EIXAY,-6*EILL*X21,-EIALL*XY,EIXAY,-6*EILL*X21},
   {-6*EILL*Y21,6*EILL*X21,2*EILL*LL,6*EILL*Y21,-6*EILL*X21,
    4*EILL*LL}}/LLLLL;
   KeG={{36*YY,-36*XY,-3*LL*Y21,-36*YY,36*XY,-3*LL*Y21},
   {-36*XY,36*XX,3*LL*X21,36*XY,-36*XX,3*LL*X21},
   {-3*LL*Y21,3*LL*X21,4*LLLL,3*LL*Y21,-3*LL*X21,-LLLL},
   {-36*YY,36*XY,3*LL*Y21,36*YY,-36*XY,3*LL*Y21},
   {36*XY,-36*XX,-3*LL*X21,-36*XY,36*XX,-3*LL*X21},
   {-3*LL*Y21,3*LL*X21,-LLLL,3*LL*Y21,-3*LL*X21,4*LLLL}}*P/(30*LLL);
   Return[{KeM,KeG}]];

PFabuleNodeSpringStiff[type_,k_]:=Module[{KeM,KeG=Table[0,{3},{3}]},
   If [type=="SpringX", KeM=DiagonalMatrix[{k,0,0}]];
   If [type=="SpringY", KeM=DiagonalMatrix[{0,k,0}]];
   If [type=="SpringT", KeM=DiagonalMatrix[{0,0,k}]];
   Return[{KeM,KeG}]];
```

FIGURE 30.2. `Pfabule` program: finite element library modules.

The program carries three DOF (two X, Y translations and one rotation about Z) at each node. If only bar elements and/or extensional springs concur at a node, the rotational DOF must be removed through a zero-rotation kinematic constraint. As a consequence, the program can also do plane truss structures (for example, the Mises truss treated in Chapter 9).

Interfaces to the analysis modules listed in Figure 30.1 are described next, in bottom up fashion. Utility modules that support printing and plotting are covered in §30.3 and §30.4.

§30.2.1. Finite Element Library

The three finite element types: beam-column, bar and node spring, are implemented in modules `LPBPlaneBarStiff`, `LPBPlaneBeamColumnStiff`, and `LPBNodeSpringStiff`, respectively.

Their coding is listed in Figure 30.2.

The element stiffness modules are invoked by the following calling sequences:

```
{KeM,KeG}=LPBPlaneBarStiff[enodXY,Em,A0,s0,options];                    (30.1)
{KeM,KeG}=LPBPlaneBeamColStiff[enodXY,Em,{A0,Izz0},s0,options];         (30.2)
{KeM,KeG}=LPBNodeSpringStiff[{kuX,kuY,kθZ}];                           (30.3)
```

The arguments are

- `enodXY` For beam-column and bar elements: list of X, Y coordinates of element end nodes in reference configuration, configured as $\{\{X_1,Y_1\},\{X_2,Y_2\}\}$
- `Em` For beam-column and bar elements: elastic modulus E.
- `A0` For beam-column and bar elements: cross section area A_0.
- `Izz0` For beam-column element: second moment of inertia I_{zz0} of beam cross section about neutral axis.
- `s0` For beam-column and bar elements: PK2 axial stress s_0 in reference configuration.
- `kuX,kuY` For node spring element: stiffness of extensional springs along X and Y axes, respectively.
- `kθZ` For node spring element: stiffness of torsional spring about Z axis.
- `options` A list of options, which may be supplied as {`numer,srule`}, {`numer`}, or { }.
  - `numer` Logical flag: set to `True` for floating-point work, and to `False` for symbolic or numerically exact work. If omitted, `False` is assumed.
  - `srule` Rule to assist symbolic simplifications if `numer` is `False`. Example: if the length of the element is a function of a reference length `L`, set `srule` to `L>0` so that `Sqrt[L^2]`, say, simplifies to `L`. If omitted, `srule={ }` is assumed so no rule applies. If `numer` is `True` this item is ignored.

All modules return a list of two matrices:

- `KeM` Element material stiffness matrix in reference configuration.
- `KeG` Element geometric stiffness matrix in reference configuration produced by the initial stress `s0`. For the node spring element, this matrix is null

§30.2.2. PFabule Master Stiffness Assembler

The module `PlaneFrameMasterStiff`, listed in Figure 30.3, assembles either the master material stiffness $\mathbf{K}_M$ or the master geometric stiffness $\mathbf{K}_G$, the choice specified through an argument flag. In what follows the number of nodes is denoted by N whereas the number of elements is N^e.

The assembler is invoked by the calling sequence

```
K=PFabuleMasterStiff[nodXYZ,eletyp,elenod,elemat,elefab,
                                  elestr,options,KMorKG];                (30.4)
```

The arguments in the calling sequence are

- `nodXYZ` Reference node coordinate list, configured as $\{\{X_1,Y_1\}\{X_2,Y_2\},\ldots\{X_N,Y_N\}\}$. The Z coordinate is assumed zero and need not be given.

```
PFabuleMasterStiff[nodXYZ_,eletyp_,elenod_,elemat_,elefab_,elestr_,
  options_,KMorKG_]:=Module[{numele=Length[eletyp],numnod=Length[nodXYZ],
  neldof,eftab,enodXY,ni,nj,i,j,ii,jj,type,type3,Em,fab,s0,numer,
  modname="PFabuleMasterStiff:",KeM,KeG,Ke,K},
  If [KMorKG!="M"&&KMorKG!="G", Print [modname," KMorKG not M or G"];
      Return[Null]]; K=Table[0,{3*numnod},{3*numnod}];
  For [e=1, e<=numele, e++, type=eletyp[[e]]; type3=StringTake[type,3];
      If [!MemberQ[{"Beam","Bar","SpringX","SpringY","SpringT"},type],
         Print [modname, " Illegal element type: ",type]; Return[Null]];
      If [type3=="Spr",    {ni}=elenod[[e]];
          eftab={3*ni-2,3*ni-1,3*ni}; fab=elefab[[e]];
         {KeM,KeG}=PFabuleNodeSpringStiff[type,fab[[1]]]];
      If [type=="Bar",
         {ni,nj}=elenod[[e]]; enodXY={nodXYZ[[ni]],nodXYZ[[nj]]};
          Em=elemat[[e]]; fab=elefab[[e]]; s0=elestr[[e]];
          eftab={3*ni-2,3*ni-1,3*nj-2,3*nj-1};
         {KeM,KeG}=PFabuleBarStiff[enodXY,Em,fab,s0,options]];
      If [type=="Beam",
         {ni,nj}=elenod[[e]]; enodXY={nodXYZ[[ni]],nodXYZ[[nj]]};
          Em=elemat[[e]]; fab=elefab[[e]]; s0=elestr[[e]];
          eftab={3*ni-2,3*ni-1,3*ni,3*nj-2,3*nj-1,3*nj};
         {KeM,KeG}=PFabuleBeamColStiff[enodXY,Em,fab,s0,options]];
      If [KMorKG=="M", Ke=KeM, Ke=KeG];  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]]]];
      ];  If [!numer, K=Simplify[K]];
  Return[K]];
```

FIGURE 30.3. LPB plane frame analysis program: master stiffness matrix assembler.

`eletyp` A list of element types configured as $\{\,\texttt{type}^{(1)},\texttt{type}^{(2)},\ldots,\texttt{type}^{N_e}\,\}$, in which $\texttt{type}^e$ is a character string that specifies the element type as `"Beam"`, `"Bar"`, or `"Spring"`, for beam-column, bar and node spring, respectively. If an illegal type is detected, the assembler prints an error message and returns `Null`.

`elenod` A list of element node lists, configured as $\{\,\texttt{enl}^{(1)},\texttt{enl}^{(2)},\ldots,\texttt{enl}^{N_e}\,\}$. Two cases:

(1) If the e^{th} element is a beam-column or bar with end nodes n_1 and n_2, in which $n_1 < n_2$, then $\texttt{enl}^e$ is $\{\,n_1, n_2\,\}$

(2) If the e^{th} element is a node spring attached to node n, then $\texttt{enl}^e$ is $\{\,n\,\}$.

`elemat` Material parameters: configured as $\{\,E^{(1)}, E^{(2)},\ldots,E^{N_e}\,\}$, in which E^e is the axial elastic modulus of the e^{th} element if it is a beam-column or bar. For a node spring element, enter 0.

`elefab` Fabrication parameters, configured as $\{\,\texttt{fab}^{(1)},\texttt{fab}^{(2)},\ldots,\texttt{fab}^{N_e}\,\}$. Three cases:

(1) If the e^{th} element is a beam-column with cross section A_0 and bending moment of inertia I_{zz0}, then $\texttt{fab}^e$ is $\{\,A_0, I_{zz0}\,\}$

(2) If the e^{th} element is a bar with cross section A_0 then $\texttt{fab}^e$ is $\{\,A_0\,\}$

(3) If the e^{th} element is a node spring with extensional stiffnesses k_{uX} and k_{uY} along X and Y, and rotational stiffness $k_{\theta Z}$ about Z, then $\texttt{fab}^e$ is $\{\,k_{uX}, k_{uY}, k_{\theta Z}\,\}$.

`elestr` Initial stresses: configured as $\{\,s_0^{(1)}, s_0^{(2)},\ldots s_0^{N_e}\,\}$, in which s_0^e is the initial axial stress of the e^{th} bar element in the reference configuration if element is a beam-column or a bar. For a node spring element, enter 0.

`options` See description in §30.2.1.

```
ModifiedStiffMatrix[nodtag_,K_]:=Module[{numdof,tags,
  fixdof,i,j,k,d,Kmod=K}, tags=Flatten[nodtag];
  numdof=Length[tags]; fixdof=Flatten[Position[tags,_?(#>0 &)]];
  For [k=1,k<=Length[fixdof],k++, i=fixdof[[k]];
      For [j=1,j<=numdof,j++, Kmod[[i,j]]=Kmod[[j,i]]=0];
      Kmod[[i,i]]=1;
      ];  ClearAll[tags,vals,fixdof];
  Return[Kmod]];

ModifiedForceVector[nodtag_,nodval_,K_,f_]:=Module[{numdof,tags,vals,
  fixdof,i,j,k,d,fmod=f}, tags=Flatten[nodtag]; vals=Flatten[nodval];
  numdof=Length[tags]; fixdof=Flatten[Position[tags,_?(#>0 &)]];
  For [k=1,k<=Length[fixdof],k++, i=fixdof[[k]]; d=vals[[i]];
      fmod[[i]]=d; If [d==0, Continue[]];
      For [j=1,j<=numdof,j++,
           If [tags[[j]]==0,fmod[[j]]-=K[[i,j]]*d]];
      ]; ClearAll[tags,vals,fixdof];
  Return[fmod]];

ReducedStiffMatrix[nodtag_,K_]:=Module[{numdof,tags,i,ii,j,jj,
  nred,retdof,Kred}, tags=Flatten[nodtag]; numdof=Length[tags];
  retdof=Flatten[Position[tags,_?(#==0 &)]]; nred=Length[retdof];
  If [nred<=0, Return[Null]]; Kred=Table[0,{nred},{nred}];
  For [i=1,i<=nred,i++,  ii=retdof[[i]];
       For [j=1,j<=nred,j++, jj=retdof[[j]];
            Kred[[i,j]]=K[[ii,jj]] ]];
  ClearAll[tags,retdof]; Return[Kred]];

CompleteDispVector[nodtag_,ured_]:=Module[{numdof,tags,i,ii,
  nred,retdof,u}, tags=Flatten[nodtag]; numdof=Length[tags];
  retdof=Flatten[Position[tags,_?(#==0 &)]]; nred=Length[retdof];
  u=Table[0,{numdof}]; If [nred<=0, Return[u]];
  For [i=1,i<=nred,i++,  ii=retdof[[i]]; u[[ii]]=ured[[i]] ];
  ClearAll[tags,retdof]; Return[u]];
```

FIGURE 30.4. LPB frame analysis program: BC application modules.

`KMorKG` A two-letter character string: `"KM"` or `"KG"` to request the master material or geometric stiffness, respectively. If neither of these, the module prints an error message and returns `Null`.

The module output is

`K` Master material stiffness matrix if `KMorKG` is `"KM"`
Master geometric stiffness matrix if `KMorKG` is `"KG"`.

§30.2.3. PFabule Boundary Condition Applicators

The four modules listed in Figure 30.4 handle application of boundary conditions at the assembly level, and thus are independent of element information. The calling sequences are

```
Kmod=PFabuleModifiedStiffMatrix[nodtag,K];                    (30.5)
fmod=PFabuleModifiedForceVector[nodtag,nodval,K,f];           (30.6)
Kred=PFabuleReducedStiffMatrix[nodtag,nodval,K,f];            (30.7)
   u=PFabuleCompleteDispVector[nodtag,nodval,ured];           (30.8)
```

The arguments in the calling sequences are

`nodtag` A list of BC tags for each DOF grouped node-by-node, arranged as $\{\{\texttt{tag}_{X1},\texttt{tag}_{Y1},\texttt{tag}_{\theta 1}\},\ldots,\{\texttt{tag}_{XN},\texttt{tag}_{YN},\texttt{tag}_{\theta N}\}\}$. The tag is 0 if the force on the DOF is prescribed, and 1 if the displacement (or rotation) is prescribed.

`nodval` Prescribed values grouped node by node, configured exactly as `nodtag`: $\{\{\texttt{val}_{X1},\texttt{val}_{Y1},\texttt{val}_{\theta 1}\},\ldots,\{\texttt{val}_{XN},\texttt{val}_{YN},\texttt{val}_{\theta N}\}\}$. A value corresponsing to a 0-tag (1-tag) in `nodtag` is the prescribed force (displacement or rotation) value.

`K` The master stiffness matrix containing *all* DOF to be modified (if argument to `ModifiedStiffMatrix`) or reduced (if argument to `ReducedStiffMatrix`).

`f` The master force vector to be modified if argument to `ModifiedStiffMatrix`). Configured as a *flat* list that includes *all* DOF.

`ured` A reduced displacement vectoir, in which all rows and columns pertaining to prescribed displacements have been removed. See Remark 30.1 for use.

The function returns are as follows:

`fmod` Modified (a.k.a. effective) force vector as flat list containing all DOF. The prescribed-displacement entries are set to the specified values in `nodval`.

`Kmod` Modified stiffness matrix, in which all rows and columns pertaining to prescribed displacements are cleared except for the diagonal entry, which is set to one.

`Kred` Reduced stiffness matrix, in which all rows and columns pertaining to prescribed displacements are removed.

`u` The complete displacement vector, in which all entries corresponding to prescribed displacements are set to those values.

Remark 30.1. The need for having both modification and reduction modules is dictated by the two phases of the LPB analysis. In Phase 1, the master stiffness equations are modified for BC before the linear solver is called; this procedure is standard in static analysis. For the eigensolution, however, it is convenient to explicitly reduce the material and geometric stiffness matrices to simplify the eigensolver work. This analysis produced reduced eigenvectors lacking prescribed DOF. To simplify printing and plotting those DOF are restored in the eigenvectors via `CompleteDispVector`.

Remark 30.2. The BC tags specified via `nodtag` may vary during the LPB analysis.

§30.2.4. Initial Stress Computation

Module `PFabuleAnalysisInitialStress`, listed in Figure 30.5, computes the initial axial stresses acting in the beam-column and bar members in the reference configuration. This is Phase 1 of the LPB analysis; see §29.4.2. As previously noted, this phase may be skipped if the initial stresses can be directly computed from statics, as is the case in isolated columns.

The module is invoked as

```
elestr=PFabuleInitialStress[nodXYZ,eletyp,elenod,elemat,elefab,
                            nodtag0,nodval,options];
```
(30.9)

where the arguments have been specified previously, except for a comment about `nodval0`. This contains the BC tags to be used in the linear static analysis that yields the initial stresses. As noted in the previous Chapter, these tags may be changed for the eigensolution phase.

```
LPBPlaneFrameInitialStress[nodXYZ_,eletyp_,elenod_,elemat_,elefab_,
  nodtag0_,nodfor_,options_]:=Module[{numele=Length[eletyp],u,e,ni,nj,
  type,Em,X1,Y1,X2,Y2,X21,Y21,uX1,uY1,uX2,uY2,ebar,elestr,m=Length[options],
  numer=False,srule={},KM,Kmod,modname="TLPlaneFrameInitialStress:"},
  elestr=Table[0,{numele}];
  If [m>=1,numer=options[[1]]]; If [m>=2,srule=options[[2]]];
  KM=LPBPlaneFrameMasterStiff[nodXYZ,eletyp,elenod,elemat,
        elefab,elestr,options,"M"]; If [KM==Null, Return[Null]];
  Kmod=ModifiedStiffMatrix[nodtag0,KM];
  (*Print["KM=",KM//MatrixForm," Kmod=",Kmod//MatrixForm,
   " f=",Flatten[nodfor]//MatrixForm," u=",u//MatrixForm];*)
  u=LinearSolve[Kmod,Flatten[nodfor]]; elestr=Table[0,{numele}];
  For [e=1, e<=numele, e++,  type=eletyp[[e]];
      If [type!="Spring"&&type!="Bar"&&type!="Beam", Print [modname,
          " Illegal element type: ",type]; Return[Null]];
      If [type=="Spring", Continue[]];
      If [type=="Bar"||type=="Beam",
          {ni,nj}=elenod[[e]]; Em= elemat[[e]];
          {X1,Y1}=nodXYZ[[ni]]; {X2,Y2}=nodXYZ[[nj]];
          X21=X2-X1; Y21=Y2-Y1; LL=Simplify[X21^2+Y21^2];
          {uX1,uY1,uX2,uY2}={u[[3*ni-2]],u[[3*ni-1]],
                             u[[3*nj-2]],u[[3*nj-1]]};
          ebar=X21*(uX2-uX1)+Y21*(uY2-uY1)/LL;
          elestr[[e]]=Em*ebar];
      ]; If [!numer,elestr=Simplify[elestr,srule]];
  Return[elestr]];
```

FIGURE 30.5. PFabule program: initial stress computation module.

The output is

`elestr` Initial stresses: configured as $\{s_0^{(1)}, s_0^{(2)}, \ldots s_0^{N_e}\}$, in which s_0^e is the initial axial stress of the e^{th} bar element in the reference configuration if element is a beam-column or a bar. For a node spring element the value is zero.

§30.2.5. Stability Eigenmatrices

Module `PFabuleAnalysisEigenmatrices`, listed in Figure **?**, sets up the satbility matrices to be submitted to the eigensolver module. These are the material and geometric master stiffness matrices upon application of boundary conditions and (optionally) a master-slave transformation that reflects multifreedom kinematic constraints. The module is invoked as

```
{KMred,KGred}=PFabuleEigenMatrices[ nodXYZ,eletyp,elenod,
                elemat,elefab,elestr,nodtag,Tdof,options];
```
(30.10)

All arguments have been previously described with two exception:

`elestr` This initial stress list has the configuration specified in §30.2.4. It is either returned by module `LPBFrameAnalysisInitialStress` as described there, or can be directly inserted if known from statics.

Tdof A master slave transformation matrix that is used to apply kinemtic constraints on the reduced degrees of freedom. Its use is illustrated in several examples presented in the next chapter. If no transformation is to be applied, enter an empty list: { }.

```
PFabuleEigenMatrices[nodXYZ_,eletyp_,elenod_,elemat_,elefab_,
  elestr_,nodtag_,Tdof_,options_]:=Module[{KM,KG,numer=False,
  m=Length[options],srule={},TdofT,KMred,KGred},
  If [m>=1,numer=options[[1]]]; If [m>=2,srule=options[[2]]];
  KM=PFabuleMasterStiff[nodXYZ,eletyp,elenod,elemat,
        elefab,elestr,options,"M"];
  KG=PFabuleMasterStiff[nodXYZ,eletyp,elenod,elemat,
        elefab,elestr,options,"G"];
  If [KM==Null||KG==Null, Return[{Null,Null}]];
  KMred=ReducedStiffMatrix[nodtag,KM];
  KGred=ReducedStiffMatrix[nodtag,KG];
  If [Length[Tdof]>0, TdofT=Transpose[Tdof];
      KMred=Simplify[TdofT.KMred.Tdof];
      KGred=Simplify[TdofT.KGred.Tdof]];
  If [!numer, {KMred,KGred}=Simplify[{KMred,KGred},srule]];
  ClearAll[KM,KG]; Return[{KMred,KGred}]];

PFabuleEigenSolution[KMred_,KGred_,nodtag_,nodval_,Tdof_,invKM_,evonly_,
  options_]:=Module[{m=Length[options],numer=False,srule={},invKG=!invKM,
  nv=Length[KMred],Q,ev,evec={},λ,λv={},vred,V={}},
  If [m>=1,numer=options[[1]]]; If [m>=2,srule= options[[2]]];
  If [invKM, Q=-LinearSolve[KMred,KGred]];
  If [invKG, Q=-LinearSolve[KGred,KMred]];
  If [numer, Q=N[Q]]; If [!numer, Q=Simplify[Q,srule]];
  If [evonly, ev=Eigenvalues[Q],{ev,evec}=Eigensystem[Q]];
  If [!numer,{ev,evec}=Simplify[{ev,evec},srule]];
  If [numer, {ev,evec}=Chop[{ev,evec}]];
  For [i=1,i<=nv,i++, λ=ev[[i]];
       If [(invKM&&λ==0)||(invKG&&λ==Infinity), Continue[]];
       If [invKM, AppendTo[λv,1/λ], AppendTo[λv,λ]]];
  If [evonly, ClearAll[Q,ev,evec]; Return[{λv,{}}]];
  For [i=1,i<=nv,i++, λ=ev[[i]];
       If [(invKM&&λ==0)||(invKG&&λ==Infinity), Continue[]];
       vred=V[[i]]; If [Length[Tdof]>0, vred=Tdof.vred];
       AppendTo[V,CompleteDispVector[nodtag,nodval,vred]]];
  ClearAll[Q,ev,evec]; Return[{λv,V}]];
```

FIGURE 30.6. LPB frame analysis program: stability eigensolution modules.

The function returns

KMred,KGred — Reduced material and geometric stiffness matrices, respectively, to be submitted to the eigensolution module. These incorporate all boundary and kinematic constraint conditions.

§30.2.6. Stability Eigensystem Solution

Module `PFabuleAnalysisEigensolution`, listed in Figure 30.6, receives the two matrices set up by module `PFabuleStabilityMatrices`, and solves the stability eigenproblem. It returns the eigenvalues (buckling loads or load factors) and optionally the associated eigenvectors (buckling shapes). The module is invoked as

```
{λv,V}=PFabuleEigensolution[KMred,KGred,nodtag,nodval,Tdof,
                            invKM,evonly,options];
```
(30.11)

The arguments `KMred`, `KGred`, `nodtag`, `nodval`, `Tdof` and `options` have been previously described. New ones are

`invKM` A logical flag that specified how to reduce the generalized eigenproblem $\mathbf{K}_M\,\mathbf{v}_i = \lambda_i \mathbf{K}_G\,\mathbf{v}_i$ to the standard eigenproblem. $\mathbf{A}\,\mathbf{v}_i = \mu_i\,\mathbf{v}_i$.
If `invKM` is `True`, premultiply by $\mathbf{K}_M$ to get $\mathbf{K}_M^{-1}\,\mathbf{K}_G\,\mathbf{v}_i = (1/\lambda_i)\,\mathbf{v}_i$ so $\mathbf{A} = \mathbf{K}_M^{-1}\,\mathbf{K}_G$ and $\mu_i = 1/\lambda_i$. This requires $\mathbf{K}_M$ to be nonsingular.
If `invKM` is `False`, premultiply by $\mathbf{K}_G$ to get $\mathbf{K}_G^{-1}\,\mathbf{K}_M\,\mathbf{v}_i = \lambda_i\,\mathbf{v}_i$ so $\mathbf{A} = \mathbf{K}_G^{-1}\,\mathbf{K}_M$ and $\mu_i = \lambda_i$; this requires $\mathbf{K}_G$ to be nonsingular.
If both matrices are singular, none of these reductions is feasible and more advanced methods must be used, or (more practical) additional BC applied to "cure" the double singularity. For example: if one forgets to delete a rotational DOF that has no beam-columns attached to it.

`evonly` A logical flag. If `True`, only the eigenvalues are return in `λv`, whereas `V` is an empty list. If `False`, both eigenvalues and eigenvectors are returned.

The output is

`λv` The computed eigenvalues of the LPB eigenproblem (29.11)

`V` If `evonly` is `false`, `V` is the matrix of computed eigenvectors, stored as rows. These are associated in one-to-one correspondence to the eigenvalues returned in `λv`.

How many eigenvalues are returned? This is a bit tricky.

Each eigenvector is dimensioned to the total number of "raw" degrees of freedom, that is, three times the number of nodes. To achieve this, the computed eigenvectors are completed with the single-DOF boundary condions as well as the slaves DOF eliminated by the master-slave transformation, if any.

All eigenvalues and eigenvectors are returned. Reason: no particular ordering is implied in symbolic computation.[1]

[1] In numerical computation, eigenvalues are returned in ascending order, but there are scenarios in which the user may want to look at more than one.

```
LPBPlaneFrameAnalysis[nodXYZ_,eletyp_,elenod_,elemat_,elefab_,
  nodtag0_,nodtag_,nodfor_,options_]:=Module[{elestr,λv,V},
  elestr=LPBPlaneFrameInitialStress[nodXYZ,eletyp,elenod,elemat,elefab,
      nodtag0,nodfor,options]; Print["elestr=",elestr];
  If [elestr==Null,Return[{Null,Null}]];
  {λv,V}=LPBPlaneFrameEigenSolution[nodXYZ,eletyp,elenod,elemat,
        elefab,elestr,nodtag,options];
  Return[{λv,V}]];
```

FIGURE 30.7. LPB frame analysis program: stability analysis driver.

§30.2.7. Analysis Driver

Module `LPBFrameAnalysisDriver`, listed in Figure 30.7, performs all LPB steps by calling the previous two modules in turn. The module is invoked as

```
{λv,V}=PFabuleAnalysisDriver[nodXYZ,eletyp,elenod,elemat,elefab,
                             nodtag0,nodtag,options];
```
(30.12)

The arguments are described in previous subsections. It should be noted that *two* DOF tag arrays are supplied: `nodtag0` is used in Phase 1 (initial stress computation) whereas `nodtag` is for Phases 2-3 (set up and solve eigenproblem).

The funtion output is

`λv,V`	Eigenvalues and eigenvectors computed by the stability eigensolver; see **?**.

§30.3. Print Utilities

Print utilities are supplied for convenient tabular display of input data as wll as output results. Ten modules are described in this subsection. With the exception of `PFabulePrintEigenVector`, which calls `PFabulePrintDispVector`, they are self-contained. The calling sequences are grouped below many of their arguments repeat:

```
PFabulePrintNodeCoord[nodXYZ,title,digits,form];                          (30.13)
PFabulePrintElemTypeAndNodes[eletyp,elenod,title,digits,form];            (30.14)
PFabulePrintElemMatAndFab[elemat,elefab,title,digits,form];               (30.15)
PFabulePrintElemGeoPlot[eletyp,elegeo,title,digits,form];                 (30.16)
PFabulePrintInitialStress[elesig,title,digits,form];                      (30.17)
PFabulePrintDOFActivity[nodtag,nodval,title,digits,form];                 (30.18)
PFabulePrintDispVector[noddis,title,digits,form];                         (30.19)
PFabulePrintForceVector[nodfor,title,digits,form];                        (30.20)
PFabulePrintEigenValues[λv,{imin,imax},title,digits,form];                (30.21)
PFabulePrintEigenVector[V,ivec,title,digits,form];                        (30.22)
```

None of these modules returns a function value since all output goes to the print channel. The code of the print modules is listed in Figures 30.8 through Figures 30.10.

```
PFabulePrintNodeCoord[nodXYZ_,title_,digits_,form_]:= Module[
 {RV,ReV,numnod=Length[nodXYZ],n,Xn,Yn,d=6,f=6,label,
  pf=InputForm,tab},  tab=Table[" ",{numnod}];
  If [Length[digits]==2,{d,f}=digits];
  RV[expr_]:=VectorQ[expr,NumericQ[#]&&(Head[#]===Real)&];
  ReV=RV[Flatten[nodXYZ]];
  If [ReV,
     For [n=1,n<=numnod,n++, {Xn,Yn}=nodXYZ[[n]];
          tab[[n]]={ToString[n],PaddedForm[Xn,{d,f}],
                                PaddedForm[Yn,{d,f}]}]];
  If [!ReV, If [MemberQ[{InputForm,StandardForm,TextForm,
                TraditionalForm},form], pf=form];
     For [n=1,n<=numnod,n++, {Xn,Yn}=nodXYZ[[n]];
          tab[[n]]={ToString[n],pf[Xn],pf[Yn]}]];
  If [StringLength[title]>0, Print[title]];
  label={"node","X-coor","Y-coor"};
  Print[TableForm[tab, TableAlignments->{Right},
        TableDirections->{Column,Row},TableSpacing->{0,2},
        TableHeadings ->{None,label}]];
  ClearAll[tab]];

PFabulePrintElemTypeAndNodes[eletyp_,elenod_,title_,digits_,form_]:=
  Module[{ReV,RV,e,numele=Length[eletyp],type,enl,label,d=4,f=4,
          pf=InputForm,tab}, tab=Table[" ",{numele}];
  If [Length[digits]==2,{d,f}=digits];
  For [e=1,e<=numele,e++, enl=elenod[[e]]; type=eletyp[[e]];
       tab[[e]]={ToString[e],type,ToString[enl]}];
  If [StringLength[title]>0, Print[title]];
  label={"elem","type","nodelist"};
  Print[TableForm[tab, TableAlignments->{Right},
        TableDirections->{Column,Row},TableSpacing->{0,2},
        TableHeadings ->{None,label}]];
  ClearAll[tab]];

PFabulePrintElemMatAndFab[elemat_,elefab_,title_,digits_,form_]:=
  Module[{ReV,RV,e,numele=Length[elemat],mat,fab,
         label,d=4,f=2,pf=InputForm,tab},
   tab=Table[" ",{numele}]; If [Length[digits]==2,{d,f}=digits];
   RV[expr_]:=VectorQ[expr,NumericQ[#]&&(Head[#]===Real)&];
   ReV=RV[Flatten[{elemat,elefab}]];
   If [ReV,
       For [e=1,e<=numele,e++,  mat=elemat[[e]]; fab=elefab[[e]];
            tab[[e]]={ToString[e],ToString[PaddedForm[mat]],
                                  ToString[PaddedForm[fab]]}]];
   If [!ReV, If [MemberQ[{InputForm,StandardForm,TextForm,
                 TraditionalForm},form], pf=form];
       For  [e=1,e<=numele,e++,  mat=elemat[[e]]; fab=elefab[[e]];
             tab[[e]]={ToString[e],pf[mat],pf[fab]}]];
   If [StringLength[title]>0, Print[title]];
   label={"elem","material","fabrication"};
   Print[TableForm[tab, TableAlignments->{Right},
       TableDirections->{Column,Row},TableSpacing->{0,2},
       TableHeadings ->{None,label}]];
   ClearAll[tab]];
```

FIGURE 30.8. Print utilities for node coordinates, element nodes, material and fabrication.

```
PFabulePrintElemIniStress[eletyp_,elestr_,title_,digits_,form_]:=
  Module[{ReV,RV,e,numele=Length[eletyp],type,sig,label,
  d=4,f=4,pf=InputForm,tab}, tab=Table[" ",{numele}];
  If [Length[digits]==2,{d,f}=digits];
  RV[expr_]:=VectorQ[expr,NumericQ[#]&&(Head[#]===Real)&];
  ReV=RV[elestr];
  If [ReV,
     For [e=1,e<=numele,e++, type=eletyp[[e]]; sig=elestr[[e]];
          tab[[e]]={ToString[e],type,
                    ToString[PaddedForm[sig,{d,f}]]}]];
  If [!ReV, If [MemberQ[{InputForm,StandardForm,TextForm,
                TraditionalForm},form], pf=form];
     For [e=1,e<=numele,e++, type=eletyp[[e]]; sig=elestr[[e]];
          tab[[e]]={ToString[e],type,pf[sig]}]];
  If [StringLength[title]>0, Print[title]];
  label={"elem","type","axial stress"};
  Print[TableForm[tab, TableAlignments->{Right},
        TableDirections->{Column,Row},TableSpacing->{0,2},
        TableHeadings ->{None,label}]];
  ClearAll[tab]];

PFabulePrintDOFActivity[nodtag_,nodval_,title_,digits_,form_]:= Module[
  {numnod=Length[nodtag],n,t1,t2,t3,v1,v2,v3,d=6,f=4,
  pf=InputForm,tab},
  tab=Table[" ",{numnod}]; If [Length[digits]==2,{d,f}=digits];
  RV[expr_]:=VectorQ[expr,NumericQ[#]&&(Head[#]===Real)&];
  ReV=RV[Flatten[{nodtag,nodval}]];
  If [ReV,
      For [n=1,n<=numnod,n++,
           {t1,t2,t3}=nodtag[[n]]; {v1,v2,v3}=nodval[[n]];
           tab[[n]]={ToString[n],
           PaddedForm[t1,d],PaddedForm[t2,d],
           PaddedForm[t3,d],PaddedForm[v1,{d,f}],
           PaddedForm[v2,{d,f}],PaddedForm[v3,{d,f}] }]];
  If [!ReV, If [MemberQ[{InputForm,StandardForm,TextForm,
                TraditionalForm},form], pf=form]; Print["pf=",pf];
      For [n=1,n<=numnod,n++,
           {t1,t2,t3}=nodtag[[n]]; {v1,v2,v3}=nodval[[n]];
           tab[[n]]={ToString[n],pf[t1],pf[t2],pf[t3],
                                 pf[v1],pf[v2],pf[v3] }]];
   If [StringLength[title]>0, Print[title]];
   Print[TableForm[tab,TableAlignments->{Right},
         TableDirections->{Column,Row},TableSpacing->{0,1},
         TableHeadings->{None,{"node", "X-tag", "Y-tag","θ-tag",
                       "X-value", "Y-value", "θ-value"}} ]];
   ClearAll[tab]];

PFabulePrintDispVector[noddis_,title_,digits_,form_]:= Module[
 {ReV,RV,numnod,uv,n,uXn,uYn,θn,d=6,f=6,label,
  pf=InputForm,tab}, uv=Partition[noddis,3]; numnod=Length[uv];
  tab=Table[" ",{numnod}]; If [Length[digits]==2,{d,f}=digits];
  RV[expr_]:=VectorQ[expr,NumericQ[#]&&(Head[#]===Real)&];
  ReV=RV[noddis];
  If [ReV,
     For [n=1,n<=numnod,n++, {uXn,uYn,θn}=uv[[n]];
          tab[[n]]={ToString[n],PaddedForm[uXn,{d,f}],
          PaddedForm[uYn,{d,f}],PaddedForm[θn,{d,f}]}]];
  If [!ReV, If [MemberQ[{InputForm,StandardForm,TextForm,
                TraditionalForm},form], pf=form];
     For [n=1,n<=numnod,n++, {uXn,uYn,θn}=uv[[n]];
          tab[[n]]={ToString[n],pf[uXn],pf[uYn],pf[θn]}]];
  If [StringLength[title]>0, Print[title]];
  label={"node","X-dis","Y-dis","θ-rot"};
  Print[TableForm[tab, TableAlignments->{Right},
        TableDirections->{Column,Row},TableSpacing->{0,2},
        TableHeadings ->{None,label}]];
  ClearAll[u,tab]];
```

FIGURE 30.9. Print utilities for initial stresses, DOF activity, and displacement vector.

```
PFabulePrintForceVector[nodfor_,title_,digits_,form_]:= Module[
 {ReV,RV,numnod,fv,n,fXn,fYn,mθn,d=6,f=6,label,
  pf=InputForm,tab}, fv=Partition[nodfor,3]; numnod=Length[fv];
  tab=Table[" ",{numnod}]; If [Length[digits]==2,{d,f}=digits];
  RV[expr_]:=VectorQ[expr,NumericQ[#]&&(Head[#]===Real)&];
  ReV=RV[nodfor];
  If [ReV,
     For [n=1,n<=numnod,n++, {fXn,fYn,mθn}=u[[n]];
          tab[[n]]={ToString[n],PaddedForm[fXn,{d,f}],
          PaddedForm[fYn,{d,f}],PaddedForm[mθn,{d,f}]}]];
  If [!ReV, If [MemberQ[{InputForm,StandardForm,TextForm,
               TraditionalForm},form], pf=form];
     For [n=1,n<=numnod,n++, {fXn,fYn,mθn}=fv[[n]];
          tab[[n]]={ToString[n],pf[fXn],pf[fYn],pf[mθn]}]];
  If [StringLength[title]>0, Print[title]];
  label={"node","X-for","Y-for","θ-mom"};
  Print[TableForm[tab, TableAlignments->{Right},
        TableDirections->{Column,Row},TableSpacing->{0,2},
        TableHeadings ->{None,label}]];
  ClearAll[fv,tab]];

PFabulePrintEigenVector[V_,ivec_,digits_,form_]:= Module[{i,evec},
  If [ivec<=0||ivec>Length[V], Print["Illegal ivec"]; Return[]];
  i=ToString[ivec]; evec=V[[ivec]];
  PFabulePrintDispVector[evec,"Eigenvector #"<>i,digits,form]];

PFabulePrintEigenValues[λv_,{imin_,imax_},title_,digits_,form_]:=
  Module[{ReV,RV,ibeg,iend,nev,i,j=1,λi,d=6,f=6,label,pf=InputForm,
  tab}, ibeg=Max[imin,1]; iend=Min[imax,Length[λv]]; nev=imax-imin+1;
  If [nev<=0, Print[" Empty prt eigval range"]; Return[]];
  tab=Table[" ",{nev}]; If [Length[digits]==2,{d,f}=digits];
  RV[expr_]:=VectorQ[expr,NumericQ[#]&&(Head[#]===Real)&]; ReV=RV[λv];
  If [ReV,
     For [i=ibeg,i<=iend,i++, λi=λv[[i]];
          tab[[j++]]={ToString[i],PaddedForm[λi,{d,f}]}]];
  If [!ReV,  If [MemberQ[{InputForm,StandardForm,TextForm,
                TraditionalForm},form], pf=form];
     For [i=ibeg,i<=iend,i++, λi=λv[[i]];
          tab[[j++]]={ToString[i],pf[λi]}]];
  If [StringLength[title]>0, Print[title]];
  label={"node","eigenvalue"};
  Print[TableForm[tab, TableAlignments->{Right},
        TableDirections->{Column,Row},TableSpacing->{0,2},
        TableHeadings ->{None,label}]];
  ClearAll[tab]];
```

FIGURE 30.10. Print utilities for force vector, eigenvalues and eigenvectors.

The arguments that have not been described in the previous sections are:

`title` A textstring to be printed as title. If supplied as either `""` or { }, no title is printed.

`digits` A list of two positive integers: {`d,f`} that specifies print width and number of digits after the decimal point when floating-point values are printed. For details see function `PaddedForm` in *Mathematica*. If given as a blank list: { }, internal defaults are used, which work fine for most cases. For printing integer values or symbolic expressions this specification is ignored.

```
DisplayChannel[]:=Module[{},
     If [$VersionNumber>=6.0,Return[Print]];
     Return[$DisplayFunction]];

ColorMap[color_]:=Module[{colspec=color},
  If [color=="Black",     colspec=RGBColor[0,0,0]];
  If [color=="Blue",      colspec=RGBColor[0,0,1]];
  If [color=="Green",     colspec=RGBColor[0,1,0]];
  If [color=="Red",       colspec=RGBColor[1,0,0]];
  If [color=="White",     colspec=RGBColor[1,1,1]];
  If [color=="GrayBG",    colspec=GrayLevel[.8]];
  If [color=="BlueBG",    colspec=Hue[.50,.6,1]];
  If [color=="GreenBG",   colspec=Hue[.25,.6,1]];
  If [color=="YellowBG",  colspec=Hue[.18,.6,1]];
  Return[colspec]];
```

FIGURE 30.11. Very low level plot utilities: `DisplayChannel` and `ColorMap`

`form`	Controls print style when symbolic expressions (including exact rational numbers) are to be printed. Options are `InputForm`, `StandardForm`, `TextForm`, and `TraditionalForm`. If given as blank list: { }, `InputForm` is assumed. Ignored if output consists only of numerical values.
`imin,imax`	In `PrintEigenValues`, range of eigenvalues to be printed.
`ivec`	In `PrintEigenVector`, index of eigenvector (buckling mode) to be printed.

§30.4. Plot Utilities

`PFabule` includes modules for plotting the original FEM model, as well as deflected shapes upon buckling. These two high-level modules in turn call low level ones, as sketched in Figure 30.1. All plot modules are described in this section, starting with the low-level ones. The reason for going "bottom up" is that it is important to understand how low level modules work in order to properly call high-level ones. In addition, they can occasionally be called directly for other tasks. Examples of use are provided where appropriate.

§30.4.1. Get Display Channel

This utility, listed in Figure 30.11, is implemented as a function (a module format is not needed). It is invoked as

```
dfun=DisplayChannel[]
```
(30.23)

This function takes no arguments. It returns the display function channel as per *Mathematica* version. If that version is less than 6.0, `dfun` is set to `$DisplayFunction`; else to `Print`.

§30.4.2. Color Name Mapping

This module, listed in Figure 30.11, receives a textstring color name as argument, for example `"Red"` or `"Black"`. As function value it returns `RGBColor`, `Hue` or `GrayLevel` specification for the built-in graphics. It is invoked as

```
colorspec=ColorMap[color]
```
(30.24)

Implemented color names may be seen in the code listing. Here "Black" through "White" are standard color names that are mapped to RGBColor specifications. Names "GrayBG", "BlueBG", "GreenBG", and "YellowBG" are used for background colors, which are lighter (lower intensity) than the standard ones. These are mapped to Hue or GrayLevel specifications as appropriate.

If color does not match one of the textstrings, the argument is returned unchanged.

Example 30.1. The calls

```
cs=ColorMap["Blue"];   cs=ColorMap["GrayBG"];   cs=ColorMap["YellowBG"];
```

return RGBColor[0,0,1], GrayLevel[.8], and Hue[.18,.6,1], respectively. On the other hand the call cs=ColorMap[RGBColor[1,1,0]]; returns RGBColor[1,1,0] unchanged in cs. This feature may be used to specify colors absent from the ColorMap internal list.

§30.4.3. Plot Cubic Shape

This module, listed in Figure 30.12, returns a graphic object that draws the deflected midline shape of a Bernoulli-Euler beam element. This shape is a combination of cubic (Hermitian) shape functions, whence the name. The calling sequence is

```
p=PlotCubicShape[xye,ue,amp,th,color,dash,subs]
```  (30.25)

The arguments are:

xye	$\{x, y\}$ coordinates of element end nodes (denoted here by 1,2) in reference configuration, configured as $\{\{x_1^e, y_1^e\},\{x_2^e, y_2^e\}\}$.
ue	Array of element node displacements configured as $\{u_{x1}, u_{y1}, \theta_1, u_{x2}, u_{y2}, \theta_2\}$.
amp	Displacement amplification factor. If zero, the midline is plotted as a straight line.
th	Thickness of deflected shape in points.
color	Line color specification. Either a textstring, such as "Black" or "Red", which is mapped by ColorMap, or a raw specification via RGBColor, GrayLevel, or Hue.
dash	A logical flag. If False, the cubic shape is drawn with a solid line. If True, it is drawn with a dashed line with dashing lengths controlled by th (see code).
subs	Number of subdivisions of the cubic shape into line segments. Normally 12 of greater to evoke a smooth curve. If subs=1, only one segment is drawn, a case that is used by the higher level module to plot a bar deflected shape.

The function return is

p	A graphic object that may be displayed with Show or combined with other objects.

§30.4.4. Cubic Shape Minimum Bounding Frame

This module, listed in Figure 30.12, go through similar motions as PlotCubicShape. But instead of producing a graphic object, it returns the $\{x, y\}$ coordinates of the smallest, axes-aligned, rectangle that encloses the plot. The is called a *minimum bounding frame* or MBF. This information is used by the higher elevel module to produce a MBF for the whole plot. The calling sequence is

```
{xmin,xmax,ynin,ymax}=CubicShapePlotMBF[xye,ue,amp,subs]
```  (30.26)

```
PlotCubicShape[xye_,ue_,amp_,th_,color_,dash_,subs_]:=Module[
  {x1,y1,x2,y2,x21,y21,L,c,s,x0,y0,ux1,uy1,θ1,ux2,uy2,θ2,k,ξ,
   uxb1,uyb1,uxb2,uyb2,uxb,uyb,darg={},xyP,p},
  {{x1,y1},{x2,y2}}=xye; x21=x2-x1; y21=y2-y1;
  {ux1,uy1,θ1,ux2,uy2,θ2}=amp*ue; L=N[Sqrt[x21^2+y21^2]];
   If [L<=0||th<=0, Return[{}]]; If [dash, darg={4,3}*th];
   p={Graphics[AbsoluteThickness[th]],Graphics[color],
      Graphics[AbsoluteDashing[darg]]}; c=x21/L; s=y21/L;
   uxb1= c*ux1+s*uy1; uxb2= c*ux2+s*uy2;
   uyb1=-s*ux1+c*uy1; uyb2=-s*ux2+c*uy2;
   xyP=Table[{0,0},{subs+1}]; xyP[[1]]={x1+ux1,y1+uy1};
   For [k=1, k<=subs, k++, ξ=N[(2*k-subs)/subs];
       x0= 0.5*(x1+x2+x21*ξ); y0=0.5*(y1+y2+y21*ξ);
       uxb=0.5*(uxb1+uxb2+(uxb2-uxb1)*ξ);
       uyb=0.125*(4*(uyb1+uyb2)+2*(uyb1-uyb2)*(ξ^2-3)*ξ+
           L*(ξ^2-1)*(θ2-θ1+(θ1+θ2)*ξ));
       xyP[[k+1]]={x0+uxb*c-uyb*s,y0+uxb*s+uyb*c}];
   AppendTo[p,Graphics[Line[xyP]]]; ClearAll[xyP];
   Return[p]];

CubicShapeMBF[xye_,ue_,amp_,subs_]:=Module[
  {x1,y1,x2,y2,x21,y21,L,c,s,x0,y0,ux1,uy1,θ1,ux2,uy2,θ2,k,ξ,
   uxb1,uyb1,uxb2,uyb2,uxb,uyb,xnew,ynew,xmin,xmax,ymin,ymax},
  {{x1,y1},{x2,y2}}=xye; x21=x2-x1; y21=y2-y1;
  {ux1,uy1,θ1,ux2,uy2,θ2}=amp*ue; L=N[Sqrt[x21^2+y21^2]];
   xmin=xmax=x1+ux1; ymin=ymax=y1+uy1;
   If [L<=0, Return[{{xmin,xmax,ymin,ymax}}]]; c=x21/L; s=y21/L;
   uxb1= c*ux1+s*uy1; uxb2= c*ux2+s*uy2;
   uyb1=-s*ux1+c*uy1; uyb2=-s*ux2+c*uy2;
   For [k=1, k<=subs, k++, ξ=N[(2*k-subs)/subs];
       x0= 0.5*(x1+x2+x21*ξ); y0=0.5*(y1+y2+y21*ξ);
       uxb=0.5*(uxb1+uxb2+(uxb2-uxb1)*ξ);
       uyb=0.125*(4*(uyb1+uyb2)+2*(uyb1-uyb2)*(ξ^2-3)*ξ+
           L*(ξ^2-1)*(θ2-θ1+(θ1+θ2)*ξ));
       xnew=x0+uxb*c-uyb*s; ynew=y0+uxb*s+uyb*c;
       xmin=Min[xmin,xnew]; ymin=Min[ymin,ynew];
       xmax=Max[xmax,xnew]; ymax=Max[ymax,ynew];
       ]; Return[{xmin,xmax,ymin,ymax}]];
```

FIGURE 30.12. Low level plot utilities dealing with cubic shapes: `PlotCubicShape` and `CubicShapeMBF`

The arguments `xye`, `ue` ,`amp` and `subs`, are exactly the same as for `PlotCubicShape`; see §30.4.3.

The function returns the 4-item list

{ xmin,. . . ymax } $\{x, y\}$ coordinates of MBF.

§30.4.5. Plot Rectilinear Spring

This module, listed in Figure 30.13, returns a graphic object that depicts a rectilinear spring joining two points. A sketch of the spring is shown in Figure 30.14(a) to help understand the module arguments listed below.

The calling sequence is

```
p=PlotRectSpring[xye,rise,m,th,color,hats]
```
(30.27)

```
PlotRectSpring[xye_,rise_,m_,th_,color_,hats_]:=Module[
 {x1,y1,x2,y2,x0,y0,k,n,r,q,subs,lines,L,H,ξη,xL,yL,xyG,p},
   {{x1,y1},{x2,y2}}=xye; x21=x2-x1; y21=y2-y1; t=Min[th,1];
   L=Sqrt[x21^2+y21^2]; {x0,y0}={x1+x2,y1+y2}/2;
   If [L==0||th<=0||hats<=0||m<=3||m>12, Return[{}]];
   r={4,0,3,0,8/3,0,0,0,12/5}[[m-3]]; If [r==0, Return[{}]];
   subs=r*hats; n=subs/m; If [!IntegerQ[n], Return[{}]];
   lines=hats+3; H=rise*L/subs; q=Mod[hats,2];
   ξη=Table[{(2*k-2-subs)/subs,0},{k,subs+1}];
   For [k=n+2,k<=subs-n-2+2*q,k=k+4, ξη[[k,2]]= 1];
   For [k=n+4,k<=subs-n-2*q  ,k=k+4, ξη[[k,2]]=-1];
   For [k=2,k<=n,k++, ξη[[k,1]]= ξη[[k,2]]=Null];
   For [k=subs-n+2,k<=subs,k++, ξη[[k,1]]=ξη[[k,2]]=Null];
   For [k=n+3,k<=subs-n-1,k++, If [ξη[[k,2]]==0,
        ξη[[k,1]]=ξη[[k,2]]=Null]];
   ξη=Transpose[DeleteCases[Transpose[ξη],Null,2]];
   p={Graphics[AbsoluteThickness[th]],Graphics[ColorMap[color]]};
   xyG=Table[{0,0},{lines+1}]; c=x21/L; s=y21/L;
   For [k=1,k<=lines+1,k++, xL=L*ξη[[k,1]]/2; yL=H*ξη[[k,2]];
        xyG[[k]]={x0+c*xL-s*yL,y0+s*xL+c*yL}];
   AppendTo[p,Graphics[Line[xyG]]]; ClearAll[ξη,xyG];
   Return[p]];

PlotCircSpring[xye_,R_,rise_,m_,th_,color_,hats_]:=Module[{x1,y1,
  x2,y2,x0,y0,arg,α,θ,k,n,r,q,subs,lines,L,H,ξη,xL,yL,xyG,p},
   If [R==×,Return[PlotRectSpring[xye,rise,m,th,color,hats]]];
   {{x1,y1},{x2,y2}}=xye; x21=x2-x1; y21=y2-y1;
   L=Sqrt[x21^2+y21^2]; {x0,y0}={x1+x2,y1+y2}/2; arg=L/(2*R);
   If [L==0||th<=0||hats<=0||m<=3||m>12||Abs[arg]>1, Return[{}]];
   α=ArcSin[N[L/(2*R)]]; c=x21/L; s=y21/L;
   r={4,0,3,0,8/3,0,0,0,12/5}[[m-3]]; If [r==0, Return[{}]];
   subs=r*hats; n=subs/m; If [!IntegerQ[n], Return[{}]];
   lines=hats+2*n+1; H=rise*L/subs; q=Mod[hats,2];
   ξη=Table[{(2*k-2-subs)/subs,0},{k,subs+1}];
   For [k=n+2,k<=subs-n-2+2*q,k=k+4, ξη[[k,2]]= 1];
   For [k=n+4,k<=subs-n-2*q  ,k=k+4, ξη[[k,2]]=-1];
   For [k=n+3,k<=subs-n-1,k++, If [ξη[[k,2]]==0,
        ξη[[k,1]]=ξη[[k,2]]=Null]];
   ξη=Transpose[DeleteCases[Transpose[ξη],Null,2]];
   p={Graphics[AbsoluteThickness[th]],Graphics[ColorMap[color]]};
   xyG=Table[{0,0},{lines+1}];
   For [k=1,k<=lines+1,k++, θ=α*ξη[[k,1]]; h=H*ξη[[k,2]];
        xL=(h-R)*Sin[θ]; yL=R*Cos[α]+(h-R)*Cos[θ];
        xyG[[k]]={x0+c*xL-s*yL,y0+s*xL+c*yL}];
   AppendTo[p,Graphics[Line[xyG]]]; ClearAll[ξη,xyG];
   Return[p]];
```

FIGURE 30.13. Plot utilities to draw rectilinear and torsional springs.

The arguments are:

xye	$\{x, y\}$ coordinates of attachment points A and B stored as $\{\{x_A, y_A\},\{x_B, y_B\}\}$. Point A is a structural node, which can generally move. Point B, which is fixed, is collocated when the fabrication data of the spring element is stored in `elefab`.
rise	Tangent of the "hat rise angle" α shown in Figure 30.14(a). Normally 1 to 2.
m	An integer that defines the ratio between the total spring length L and the "hatless" end arm length L_a, as $L_a = L/m$. Legal values of `m` are 4, 6, 8 or 12. Usually `m=6` or `m=8` gives the best picture.

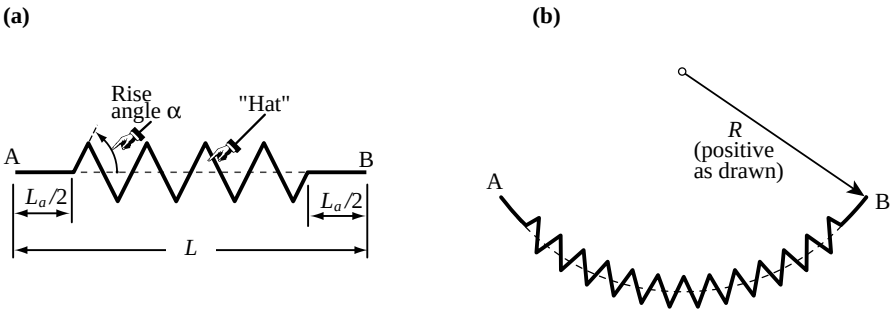

FIGURE 30.14. Rectilinear and circular spring sketches.

th	Thickness of spring lines in points. Usually 2 to 4.
color	Line color specification. See §30.4.3
hats	Number of hats to be drawn. In Figure 30.14(a), the number of hats is 8. This number is constrained by the value of `m`: If `m=4`, `hats` can be any positive integer: 1,2,3 . . . If `m=6`, `hats` must be positive even: 2,4,6 . . . If `m=8`, `hats` must be a multiple of 3: 3,6,9 . . . If `m=12`, `hats` must be a multiple of 5: 5,10,15 . . .

The function return is

p	A graphic object that may be displayed with `Show` or combined with other objects.

The plot of Figure 30.14(a) is produced by
```
p=PlotRectSpring[{{0,0},{1,0}},1,6,2,
"Black",8]; Show[p,AspectRatio->1/6,DisplayFunction->DisplayChannel[]];
```

If an argument is invalid, for example `m=10`, no plot is generated and the empty list is returned.

§30.4.6. Plot Circular Spring

This module, listed in Figure 30.13, returns a graphic object that depicts a torsional spring that joints two points. The spring is drawn over an arc of circle, whence the name. A circular spring sketch is shown in Figure 30.14(b) to help understand the module arguments listed below.

The calling sequence is

```
p=PlotCircSpring[xye,R,rise,m,th,color,hats]
```  (30.28)

The arguments are:

xye	$\{x, y\}$ coordinates of attachment points A and B stored as $\{\{x_A, y_A\},\{x_B, y_B\}\}$. When this module is called from a higher level one, the convention is that point A is attached to a beam or bar structural member, which will generally move, whereas point B is fixed. Note that A is not drawn attached to a node, but at a certain distance from it to clearly exhibit the torque lever arm. These points are defined via the fabrication data stored in `elefab`.
R	Arc radius. Must exceed the semidistance betrween A and B. Its sign is important as it defines the curvature sense; for example in Figure 30.14(b) `R>0` because the arc

curvature is positive (concave with respect to y). If `R` is specified as infinite (that is, `R=Infinity`), the `PlotRectSpring` module is called to return `p`.

rise	Same as in `PlotRectSpring`.
m	Same as in `PlotRectSpring`. Normally `m=12` is used.
th	Same as in `PlotRectSpring`
color	Same as in `PlotRectSpring`
hats	Same as in `PlotRectSpring`. In Figure 30.14(b), `hats=20`. Legal hats number are restricted by `m` according to the same rules as for the rectilinear spring.

The function return is

p	A graphic object that may be displayed with `Show` or combined with other objects.

The plot of Figure 30.14(b) is produced by `p=PlotCircSpring[{{0,0},{Sqrt[2],0}},1,2,12, 2,"Black",20]; Show[p,AspectRatio->1/4,DisplayFunction->DisplayChannel[]];`. This particular spring spans an angle of 90°.

If an argument is invalid, for example the radius `R` is too small, no plot is generated and the empty list is returned.

§30.4.7. Plot FEM Model

Module `PFabulePlotFEMModel`, listed in Figure 30.15, produces as plot of the finite element model. This highre level module has a many options, through which one may include element numbers, node numbers and boundary conditions in the plot. The calling sequence is

```
p=PFabulePlotFEMModel[nodXYZ,eletyp,elenod,elegeo,tcinfo,ainfo,
                      einfo,ninfo,frame,backgr,imgsiz,title]
```
(30.29)

The arguments are:

nodXYZ	List of node coordinates in the reference configuration.
eletyp	List of element types.
elenod	List of element nodes
elegeo	Presently a dummy argument
tcinfo	List of thickness and color information for different element types. Configured as `{{tbeam,cbeam},{tbar,cbar},{tspr,cspr}}`. Here `tbeam`, `tbar` and `tspr` are the thicknesses used to draw beam, bar and spring elements, respectively, specified in points; whereas `cbeam`, `cbar` and `cspr` are the color specifications for beam, bar and spring elements, respectively.
backgr	Background color. See §30.4.2
ainfo	A list directly or indirectly contrlling the plot aspect ratio.
einfo	Element labeling instructions. Either { } to skip element labels, or a list `lb items1,item2,item3`.
ninfo	Node labeling instructions. Either { } to skip node labels, or a list `lb items1,item2,item3`.

```
PFabulePlotFEMModel[nodXYZ_,eletyp_,elenod_,elegeo_,tcinfo_,ainfo_,
   einfo_,ninfo_,frame_,backgr_,imgsiz_,title_]:= Module[{
   numele=Length[elenod],numnod=Length[nodXYZ],ltc=Length[tcinfo],
   lai=Length[ainfo],lei=Length[einfo],lni=Length[ninfo],i,k,e,enl,
   n,ni,nj,x,y,asp=Automatic,xmin,xmax,ymin,ymax,dx,dy,xdim=0,ydim=0,
   dmin,tbeam=4,cbeam="Red",tbar=2,cbar="Blue",tspr=1,cspr="Black",
   ex,ey,cnx=2.,cny=1.5,xy0,xye,xyn,elabs=nlabs=False,frade,fradn,xylab,
   sink,rade=0,radn=0,elab,nlab,backg,dfun,arat,cmargin=0.08,mx,my,
   plow,phigh,pbeam=pbar=pspr=pnod=pnlab=pelab=pecir=pedisk=pbound={},
   Glin,Gtbeam,Gcbeam,Gtbar,Gcbar,Gtspr,Gcspr,Gtcir,Gwhite,Gblack},
   x=nodXYZ[[All,1]]; y=nodXYZ[[All,2]]; cnx=cny=0; cey=0.002;
  {xmin,xmax,ymin,ymax}=N[{Min[x],Max[x],Min[y],Max[y]}];
   If [ltc>=1,{tbeam,cbeam}=tcinfo[[1]]];
   If [ltc>=2,{tbar,cbar}=  tcinfo[[2]]];
   If [ltc>=3,{tspr,cspr}=  tcinfo[[3]]];
   If [lei==1, elabs=True; {frade}=einfo];
   If [lni==1, nlabs=True; {fradn}=ninfo];
   If [lni==3, nlabs=True; {fradn,cnx,cny}=ninfo];
   If [lai>=1, asp= ainfo[[1]]]; If [lai>=2, xdim=ainfo[[2]]];
   If [lai>=3, ydim=ainfo[[3]]]; If [lai>=4, cmargin=ainfo[[4]]];
   {dx,dy}={xmax-xmin,ymax-ymin};
   If [dx<xdim, xmin-=(xdim-dx)/2; xmax+=(xdim-dx)/2];
   If [dy<ydim, ymin-=(ydim-dy)/2; ymax+=(ydim-dy)/2];
   {dx,dy}={xmax-xmin,ymax-ymin}; dmin=Min[dx,dy];
   rade=frade*dmin; radn=fradn*dmin; sink={0,0.1*rade};
   For [e=1,e<=numele,e++, etype=eletyp[[e]]; enl=elenod[[e]];
        If [StringTake[type,3]=="Spr", Continue[]]; {ni,nj}=enl;
        xye={nodXYZ[[ni]],nodXYZ[[nj]]}; Glin=Graphics[Line[xye]];
        If [etype=="Beam", AppendTo[pbeam, Glin]];
        If [etype=="Bar",  AppendTo[pbar,  Glin]];
        If [elabs, xy0=(xye[[1]]+xye[[2]])/2; xylab=xy0-sink;
            AppendTo[pedisk,Graphics[Disk[xy0,rade]]];
            AppendTo[pecir,Graphics[Circle[xy0,rade]]];
            elab=StyleForm[e,FontFamily->"Times",FontSize->11];
            AppendTo[pelab,Graphics[Text[elab,xylab]]]]
       ]; {ex,ey}={cnx*radn,cny*radn}; {mx,my}=cmargin*{dx,dy};
   For [n=1,n<=numnod,n++, xyn=nodXYZ[[n]]; xylab=xyn+{ex,ey};
        If [nlabs,nlab=StyleForm[n,FontFamily->"Times",
                  FontSize->11, FontWeight->Bold];
            AppendTo[pnlab,Graphics[Text[nlab,xylab]]]];
        AppendTo[pnod, Graphics[Disk[xyn,radn]]]
       ];  plow={xmin-mx,ymin-my}; phigh={xmax+mx,ymax+my};
   pbound={Graphics[Point[plow]],Graphics[Point[phigh]]};
   If [asp>0, arat=asp, arat=dy/dx]; If [asp<0, arat=Automatic];
   dfun=DisplayChannel[]; backg=ColorMap[backgr];
   Gtbeam=Graphics[AbsoluteThickness[tbeam]];
   Gtbar =Graphics[AbsoluteThickness[tbar]];
   Gtspr =Graphics[AbsoluteThickness[tspr]];
   Gcbeam=Graphics[ColorMap[cbeam]]; Gcbar=Graphics[ColorMap[cbar]];
   Gcspr= Graphics[ColorMap[cspr]];  Gtcir=Graphics[AbsoluteThickness[1.25]];
   Gwhite=Graphics[RGBColor[1,1,1]]; Gblack=Graphics[RGBColor[0,0,0]];
   Show[Gtbeam,Gcbeam,pbeam,Gtbar,Gcbar,pbar,Gcspr,Gtspr,pspr,
        Gblack,pnod,Gtcir,Gwhite,pedisk,Gblack,pecir,
        pelab,pnlab,Gwhite,pbound, Background->backg,
        Frame->frame,PlotLabel->title,ImageSize->imgsiz,
        PlotRange->{{xmin-mx,xmax+mx},{ymin-my,ymax+my}},
        Axes->False,AspectRatio->arat,DisplayFunction->dfun];
    ClearAll[x,y,pnod,pnlab,pesid,pelab,pecir,pedisk,pbound];
];
```

FIGURE 30.15. Module to plot finite element model.

```
PFabulePlotDeformedShape[nodXYZ_,eletyp_,elenod_,elegeo_,u_,
 amp_,subs_,tcinfo_,ainfo_,frame_,backgr_,imgsiz_,title_]:=Module[{
 numele=Length[eletyp],numnod=Length[nodXYZ],na=Length[ainfo],
 e,etype,enl,i,ni,nj,ii,jj,ue,uez,xye,x,y,asp=Automatic,xdim=0,ydim=0,
 xmin,xmax,ymin,ymax,sframe,tbeam,cbeam,tbar,cbar,tspr,cspr,tref,cref,
 x1,x2,y1,y2,n,t,col,dash,arat=Automatic,dfun,pbound,pdef,pref},
 uez=Table[0,{6}]; x=nodXYZ[[All,1]]; y=nodXYZ[[All,2]];
 {xmin,xmax,ymin,ymax}=N[{Min[x],Max[x],Min[y],Max[y]}]; pdef=pref={};
{{tbeam,cbeam},{tbar,cbar},{tspr,cspr},{tref,cref}}=tcinfo;
 For [e=1,e<=numele,e++, etype=eletyp[[e]]; enl=elenod[[e]];
     If [etype=="Spring", Continue[]]; {ni,nj}=enl;
     xye={nodXYZ[[ni]],nodXYZ[[nj]]}; ii=3*ni; jj=3*nj;
     ue={u[[ii-2]],u[[ii-1]],u[[ii]],u[[jj-2]],u[[jj-1]],u[[jj]]};
     If [etype=="Beam", n=subs; t=tbeam; col=ColorMap[cbeam]];
     If [etype=="Bar",  n=1;    t=tbar;  col=ColorMap[cbar]];
     AppendTo[pdef,PlotCubicShape[xye,ue,amp,t,col,False,n,{}]];
     If [tref>0, col=ColorMap[cref];
         AppendTo[pref,PlotCubicShape[xye,ue,0,tref,col,True,1,{}]]];
     {x1,x2,y1,y2}=CubicShapeMBF[xye,ue,amp,subs];
     xmin=Min[xmin,x1]; ymin=Min[ymin,y1];
     xmax=Max[xmax,x2]; ymax=Max[ymax,y2];
     ];
 If [na>=1, asp= ainfo[[1]]]; If [na>=2, xdim=ainfo[[2]]];
 If [na>=3, ydim=ainfo[[3]]]; {dx,dy}={xmax-xmin,ymax-ymin};
 If [dx<xdim, xmin=xmin-(xdim-dx)/2; xmax=xmax+(xdim-dx)/2];
 If [dy<ydim, ymin=ymin-(ydim-dy)/2; ymax=ymax+(ydim-dy)/2];
 {dx,dy}={xmax-xmin,ymax-ymin};  If [asp==0, arat=dy/dx];
 If [asp>0, arat=asp]; dfun=DisplayChannel[]; backg=ColorMap[backgr];
 pbound={Graphics[RGBColor[1,1,1]],Graphics[AbsolutePointSize[1]],
         Graphics[Point[{xmin,ymin}]],Graphics[Point[{xmax,ymax}]]};
 Show[pref,pdef,pbound, Background->backg, AspectRatio->arat,
      ImageSize->imgsiz,Frame->frame,
      PlotLabel->title,DisplayFunction->dfun];
 ClearAll[x,y,pdef,pref,pbound];
 ];
```

FIGURE 30.16. Module that plots deflected structure shape.

frame	A logicak flag to specifies a frame be drawn around the plot: `True`: yes, `False`: no
imgsiz	Plot image horizontal size in points
title	Optional plot heading. Omitted if `""`

The function does not return a value.

§30.4.8. Plot Deflected Shape

Module `PFabulkePlotFEMModel`, listed in Figure 30.15, produces as plot of the deflected finite element model in one of the buckling models. The calling sequence is

```
p=PFabulePlotDeflectedShape[nodXYZ,eletyp,elenod,elegeo,u,amp,subs,tcinfo,ainfo,
                            frame,backgr,imgsiz,title]
```
(30.30)

The arguments are the same as the prvious module, except for

u	Nodal displacements
amp	Amplification factor for the displacements

subs — Number of subdiviions of beam elemenet when plotting the cubic deflection as piecewise linear segments. See **?** for details.

tcinfo — List of thickness and color information for different element types. Same as for the `PFabulaPlotFEModel` but the list is configured with one extra item: `{{tbeam,cbeam},{tbar,cbar},{tspr,cspr}, {tref,cref}}`. Here `tref` and `cref` denote "reference thickness" and "reference color," respectively. If `tref>0`, the reference shape of the FEM model is drawn in dashed lines with thickness `tref` in points, else it is not.

The function does not return a value.