The TL Plane Beam Element: Implementation
TABLE OF CONTENTS

§12.1 Introduction ........................................ 12–3
§12.2 Internal Force Vector ............................... 12–3
§12.3 Element Stiffness Matrix ............................ 12–3
  §12.3.1 Material Stiffness Matrix ....................... 12–4
  §12.3.2 Geometric Stiffness Matrix ...................... 12–6
  §12.3.3 Tangent Stiffness Matrix ......................... 12–7
  §12.3.4 Consistency Check By Finite Differences .......... 12–7
§12.4 Cantilever Beam Structure ......................... 12–9
  §12.4.1 Cantilever Beam as Benchmark ................... 12–9
  §12.4.2 Cantilever Beam Internal Force Assembler ........ 12–10
  §12.4.3 Cantilever Beam Tangent Stiffness Assembler ..... 12–12
  §12.4.4 Critical Load Finding Methods ................... 12–12
  §12.4.5 Buckling Analysis Via Zero Determinant .......... 12–13
  §12.4.6 Buckling Analysis Via Stability Eigenproblem ...... 12–14
  §12.4.7 Extrapolation of Buckling Load Results .......... 12–16
§12. Exercises ............................................. 12–17

12–2
§12.1. Introduction

This Chapter presents the implementation of the TL plane beam element formulated in Chapter 10. The set of modules described here are web posted as notebook TLPlaneBeam.nb linked to this Chapter Index. The important pieces of code included there are described below.

In Chapter 9 it was suggested that, when implementing geometrically nonlinear analysis by FEM, the equilibrium level should be done first. That would be followed by the incremental level, which includes the tangent stiffness matrix. In this Chapter both levels are implemented at the same time. The main reason is that only the stiffness matrix (separated into material and geometric) is needed for homework exercise work. The internal force module is only used as a consistency check for the stiffness matrix code, using finite differences, in §12.3.4. It is, however, used in a later Chapter.

§12.2. Internal Force Vector

The Mathematica module TLPlaneBeamInternalForce computes and returns the internal force vector of the TL plane beam finite element formulated in the previous Chapter. The module is listed in Figure 12.1 along with test statements. It is invoked by the calling sequence

\[
pe = \text{TLPlaneBeamIntForce}[\text{XYcoor}, S0, z0, uXY\theta, \text{numer}];
\]  

(12.1)

The arguments in the calling sequence are
- \text{XYcoor} List of \{X, Y\} coordinates of element end nodes in reference configuration, arranged as \{
{X_1,Y_1}, \{X_2,Y_2\}\}
- \text{S0} A list \{EA_0, GA_s, EI_0\} of the following cross sectional properties:
  - \(EA_0\): Axial rigidity in reference configuration
  - \(GA_s\): Shear rigidity in reference configuration (\(A_s\) is the shear area)
  - \(EI_0\): Bending (flexural) rigidity in reference configuration

To set up the RBF “unlocking” trick, replace argument \(GA_s\) by either \(12EI_0/L_0^2\), or the more refined value given in Remark 7 of Chapter 10.
- \text{z0} A list \{F_0, V_0, M_0\} that contains axial force \(F_0\), transverse shear force \(V_0\), and bending moment \(M_0\), in the reference configuration. These forces are assumed to be constant along the element.
- \text{uXY\theta} List of displacements and rotation of element end nodes from reference to current configuration, arranged as \{
{u_{X1}, u_{Y1}, \theta_1}, {u_{X2}, u_{Y2}, \theta_2}\}\}
- \text{numer} A logical flag: True to request numerical floating point work. Else set to False.

The output of the module is returned as function value:

\[
pe \text{ Element internal force returned as the } 6 \times 1 \text{ matrix}
\]

\[
\{\{p_{X1}\}, \{p_{Y1}\}, \{p_{\theta1}\}, \{p_{X2}\}, \{p_{Y2}\}, \{p_{\theta2}\}\}
\]

A test statement is shown in blue at the bottom of Figure 12.1. It uses a combination of symbolic and numeric data. The test output is shown in Figure 12.2. The returned value is printed as a one-row matrix to save space.
Chapter 12: THE TL PLANE BEAM ELEMENT: IMPLEMENTATION

\[
\text{TLPlaneBeamIntForce}[\text{XYcoor}_\text{,}S_0_\text{,}z_0_\text{,}u\theta_\text{,}\text{numer}_\text{,}]:=
\text{Module}[\{X1,Y1,X2,Y2,X21,Y21,L0,uX1,uY1,\theta\text{1},uX2,uY2,\theta\text{2},
\text{x21},\text{y21},\text{uX0},\text{uY0},\theta\text{m},c\theta,s\theta,Lc\psi,Ls\psi,N0,V0,M0,E0\text{0},G0\text{0},E0\text{0},
c\phi,s\phi,cm,sm,Nm,Vm,Mm,em,gm,\kappa,pe\},
\{\{X1,Y1\},\{X2,Y2\}\text{=}\text{XYcoor}; X21=X2-X1; Y21=Y2-Y1;\}
\{\{uX1,uY1,\theta\text{1}\},\{uX2,uY2,\theta\text{2}\}\text{=}u\theta\text{,}\}
x21=X21+uX2-uX1; y21=Y21+uY2-uY1;
L0=\text{PowerExpand}[\sqrt{X21^2+Y21^2}] ;
\theta\text{m}=\frac{\theta\text{1}+\theta\text{2}}{2}; c\theta=\cos[\theta\text{m}]; s\theta=\sin[\theta\text{m}];
Lc\psi=(X21*x21+Y21*y21)/L0; Ls\psi=(X21*y21-Y21*x21)/L0;
\text{em}=\frac{c\theta*Lc\psi+s\theta*Ls\psi}{L0-1}; \text{gm}=\frac{s\theta*Lc\psi-c\theta*Ls\psi}{L0}; \kappa=\frac{\theta\text{2}-\theta\text{1}}{L0};
\{N0,V0,M0\}=z0; \{E0\text{0},G0\text{0},E0\text{0}\}=S0; \text{Nm}=N0+E0\text{0}*em; \text{Vm}=V0+G0\text{0}*gm; \text{Mm}=M0+E0\text{0}*\kappa;
c\phi=X21/L0; s\phi=Y21/L0;
\text{cm}=c\theta*c\phi-s\theta*s\phi; \text{sm}=s\theta*c\phi+c\theta*s\phi;
\text{Bm}=\left\{\begin{array}{ll}
-cm,-sm, & L0*gm/2, \\
sm,-cm, & L0*gm/2, \\
0, & 0, -1, \\
0, & 0, 1
\end{array}\right\}/L0;\text{pe}=L0*\text{Transpose}[\text{Bm}].\{\{\text{Nm},\{\text{Vm},\{\text{Mm}}\};
\text{If} [\text{numer},\text{pe}=\text{N}[\text{pe}]]; \text{If} [\text{!numer},\text{pe}=\text{Simplify}[\text{pe}]];\text{Return}[\text{pe}];\text{\text{ClearAll}}[L,E0\text{a},G0\text{as},E0\text{i},F0,V0,M0];
\text{XYcoor}=\{(0,0),(L/\sqrt{2},L/\sqrt{2})\};\text{S0}=\{E0\text{a},G0\text{as},E0\text{i}\}; \text{z0}=\{F0,V0,M0\};
u\theta\text{,}=\{(0,0,\pi/2),(-2*L/\sqrt{2},0,\pi/2)\};
\text{pe}=\text{TLPlaneBeamIntForce}[\text{XYcoor},\text{S0},\text{z0},\text{u\theta\text{,}},\text{False}];
\text{Print}["\text{pe=},\text{N}[\text{pe}]//\text{InputForm}\];
\]

**Figure 12.1.** Module that computes and returns the internal force vector \( p \) for a TL plane beam element.

\[
\text{pe}=\left\{\begin{array}{c}
\frac{F0+V0}{\sqrt{2}}, \\
\frac{-F0+V0}{\sqrt{2}}, \\
\frac{1}{2}(-2M0-LV0), \\
\frac{-F0+V0}{\sqrt{2}}, \\
\frac{F0+V0}{\sqrt{2}}, \\
\frac{(M0-LV0)}{2}
\end{array}\right\} = \left\{\begin{array}{c}
0.7071067811865475*F0+(0.0), \\
0.7071067811865475*(-0.5*F0+V0), \\
(0.5*(-2*0.5+1.0*V0)), \\
0.7071067811865475*F0+V0), \\
0.7071067811865475*F0+(0.0)
\end{array}\right\}
\]

\section*{§12.3. Element Stiffness Matrix}

The tangent stiffness matrix of the TL plane bar element formulated in the previous Chapter is implemented in the three modules described next.

\subsection*{§12.3.1. Material Stiffness Matrix}

The material stiffness matrix, derived in ?\text{,} is implemented in \text{Mathematica} module called \text{TLPlaneBeamMatTanStiff}. The module is listed in Figure 12.3, along with test statements. It is invoked by the following calling sequence:

\[
\text{KMe = TLPlaneBeamTanStiff}[\text{XYcoor},S0,z0,u\theta\text{,},\text{numer}]; \tag{12.2}
\]

The arguments in the calling sequence are exactly the same as described for the internal force module \text{TLPlaneBeamIntForce} in (12.1) of §12.2.

To set up the RBF “unlocking” trick, replace argument \( GA_0 \) in the \( S0 \) list by either \( 12EI0/L0^2 \), or the more refined value given in Remark ? of Chapter 10.

The output of the module is returned as function value:

\[
\text{KMe} \quad \text{Element material stiffness returned as the} \ 6 \times 6 \ \text{matrix}.
\]
TLPlaneBeamMatStiff[XYcoor__, S0__, z0__, uXYθ__, numer__]:=
Module[{X1,Y1,X2,Y2,X21,Y21,L0,uX1,uY1,θ1,uX2,uY2,θ2,
x21,y21,uX0,uY0,θm,cθ,sm,em,κ,κN,N0,V0,M0,
EA0,GAs,EIs,EIsI,cφ,sφ,cm,sm,a1,KMe},
{X1,Y1}={X2,Y2}; X21=X2-X1; Y21=Y2-Y1;
{uX1,uY1,θ1},{uX2,uY2,θ2}={uX0,uY0,θm}+
uXYθ;
x21=x21+uX2-uX1; y21=y21+uY2-uY1;
L0=PowerExpand[Sqrt[X21^2+Y21^2]];
θm=(θ1+θ2)/2; cθ=Cos[θm]; sm=Sin[θm];
Lc=(X21*x21+Y21*y21)/L0; Ls=(X21*y21-Y21*x21)/L0;
em=(cθ*Lc+sm*Ls)/L0-1; gm=-(sm*Lc-cθ*Ls)/L0;
cφ=X21/L0; sm=X21/L0; cm=cφ+sm; sm=sφ+cφ+sm;
{N0,V0,M0}=z0; {EA0,GAs,EIs}=S0; a1=1+em;
KMe = (EA0/L0)*{
{cm^2,cm*sm,-cm*gm*L0/2,-cm^2,cm*gm*L0/2},
{cm*sm,sm^2,-sm*gm*L0/2,-cm*sm,sm^2},
{cm*gm*L0/2,-gm*L0*sm/2,sm*gm*L0^2/4,cm*gm*L0/2,-gm*L0*sm/2},
{-cm,sm*gm*L0/2,-sm*gm*L0^2/4,cm*gm*L0/2,sm*gm*L0^2/4}};
{GAs/L0}*{
{sm^2,sm*cm,-sm*a1*L0*sm/2,-sm^2,sm*cm*a1*L0*sm/2},
{-sm*cm,sm*cm,sm*cm,a1*L0*sm/2,sm*cm},
{-a1*L0*sm/2,cm*a1*L0/2,cm*a1*L0/2,a1^2*L0*sm/2/4},
{a1*L0*sm/2,-cm*a1*L0/2,-cm*a1*L0/2,a1^2*L0*sm/2/4},
{-sm*cm,cm*sm,sm*cm,cm*sm,a1*L0*sm/2},
{-cm*cm,cm*cm,cm*cm,a1*L0*sm/2,cm*cm}};
If[numer,KMe=N[KMe]]; If[!numer,KMe=Simplify[KMe]];
Return[KMe] ];
ClearAll[EA,GAs,EIs];
XYcoor={0,0},{10,0}; S0={EA,GAs,EIs}; z0={0,0,0};
uXYθ={0,0,0,0};
KM=TLPlaneBeamMatStiff[XYcoor,S0,z0,uXYθ,False];
Print[Quiet[KM//MatrixForm]];

Figure 12.3. Module that returns the material stiffness matrix $K_M$ for a TL plane beam element.

\[
\frac{EA}{10} \begin{array}{cccc}
0 & 0 & - \frac{EA}{10} & 0 \\
0 & \frac{GAs}{10} & \frac{GAs}{2} & 0 \\
0 & \frac{GAs}{2} & \frac{1}{10} & (EI + 25 GAs) \\
- \frac{EA}{10} & 0 & 0 & 0 \\
0 & \frac{GAs}{2} & 0 & 0 \\
0 & \frac{GAs}{2} & \frac{1}{10} & (-EI + 25 GAs) \\
\end{array}
\]

Figure 12.4. Result from test statements shown in Figure 12.3

A test statement is shown in blue text at the bottom of Figure 12.3. It uses a combination of symbolic
Chapter 12: THE TL PLANE BEAM ELEMENT: IMPLEMENTATION

TLPlaneBeamGeoStiff[XYcoor_, S0_, z0_, uXYθ_, numer_] :=

Module[{X1, Y1, X2, Y2, X21, Y21, L0, uX1, uY1, θ1, uX2, uY2, θ2,
x21, y21, u0, uθ, cθ, sθ, Lψ, Lsψ, em, gm, κ, N0, V0, M0,
EA0, GA0, EI0, L0h, cφ, sφ, cm, sm, Nm, Vm, KGe},

{{X1, Y1}, {X2, Y2}} = XYcoor; X21 = X2 - X1; Y21 = Y2 - Y1;

{{uX1, uY1, θ1}, {uX2, uY2, θ2}} = uXYθ;

x21 = X21 + uX2 - uX1; y21 = Y21 + uY2 - uY1;

L0 = PowerExpand[Sqrt[X21^2 + Y21^2]]; L0h = L0/2;

θm = (θ1 + θ2)/2; cθ = Cos[θm]; sθ = Sin[θm];

Lψm = (X21*x21 + Y21*y21)/L0; Lsψm = (X21*y21 - Y21*x21)/L0;

em = (cθ*Lψm + sθ*Lsψm)/L0 - 1; gm = -(sθ*Lψm - cθ*Lsψm)/L0;

κ = (θ2 - θ1)/L0; {N0, V0, M0} = z0; {EA0, GA0, EI0} = S0;

Nm = Simplify[N0 + EA0*em]; Vm = Simplify[V0 + GA0*gm];

cφ = X21/L0; sφ = Y21/L0; cm = cθ*cφ - sθ*sφ; sm = sθ*cφ + cθ*sφ;

KGe = Nm/2*{{0, 0, sm, 0, 0, sm}, {0, 0, -sm, 0, 0, -sm},

{sm, -sm, -L0h*(1 + em), -sm, cm, -L0h*(1 + em)} +

Vm/2*{{0, 0, cm, 0, 0, cm}, {0, 0, sm, 0, 0, sm},

{cm, sm, -L0h*gm, -sm, cm, -L0h*gm}} +

If [numer, KGe = N[KGe]]; If [!numer, KGe = Simplify[KGe]];

Return[KGe]];}

XYcoor = {{0, 0}, {4, 3}}; S0 = {1, 1, 1}; z0 = {10, 30, 20};
uXYθ = {{0, 0, 0}, {0, 0, 0}};

KGe = TLPlaneBeamGeoStiff[XYcoor, S0, z0, uXYθ, numer];

Print[N[KGe]]; Print[Chop[Evolution[N[KGe]]]];}

Figure 12.5. Module that computes and returns the geometric stiffness matrix \(K_G\) for a TL plane beam element.

Figure 12.6. Result from test statements shown in Figure 12.5

and numeric data. The test output is shown in Figure 12.4.

§12.3.2. Geometric Stiffness Matrix

The geometric stiffness matrix, derived in ?, is implemented in the Mathematica module TLPlaneBeamGeoTanStiff. The module is listed in Figure 12.5, along with test statements. It is invoked by the following calling sequence:

\[
K Ge = TLPlaneBeamGeoStiff[XYcoor, S0, z0, uXYθ, numer];
\]  

(12.3)

The arguments in the calling sequence are exactly the same as described for the internal force module TLPlaneBeamIntForce in (12.1) of §12.2.

The output of the module is returned as function value:

\(K Ge\) Element geometric stiffness returned as the 6 \(\times\) 6 matrix.

12–6
§12.3 ELEMENT STIFFNESS MATRIX

A test statement is shown in blue text at the bottom of Figure 12.5. The test uses numeric data. The output is shown in Figure 12.6. It includes getting the eigenvalues of $K_G$. For the supplied data, the matrix can be observed to have a rank of 2.

§12.3.3. Tangent Stiffness Matrix

The tangent stiffness matrix is returned by the Mathematica module TLPlaneBeamTanStiff. The module is listed in Figure 12.7. It is invoked by the following calling sequence:

$$K_e = TLPlaneBeamTanStiff[XYcoor, S0, z0, uXY\theta, numer];$$ (12.4)

The arguments in the calling sequence are exactly the same as described for both the material stiffness module TLPlaneBeamMatStiff in §12.3.1, and the geometric stiffness module TLPlaneBeamGeoStiff in §12.3.2. Module TLPlaneBeamTanStiff simply calls those modules and adds $K_M$ and $K_G$ to get $K$, which is returned as function value. No test statements are included since the logic is straightforward.

§12.3.4. Consistency Check By Finite Differences

Recall from Chapters 3–4 that, if the mathematical model discretized by FEM is separable, the tangent stiffness matrix is the gradient of the internal force vector with respect to the degrees of freedom (DOF) collected in the state vector $u$:

$$K = \frac{\partial p}{\partial u}. \quad (12.5)$$

If the programmed modules for the internal force and tangent stiffness of an element comply with (12.5) for all legal inputs, the implementations are called consistent. In practice, the program modules that return $p$ and $K$ are usually coded separately. How can consistency be checked? The most common technique relies on finite differences. Basically this works as follows:

1. Select a set of arbitrary numerical inputs that defines the reference configuration, material and fabrication properties, initial stresses, and node displacements. Let $C$ be the current configuration produced by those test inputs.

2. Perturb $C$ by moving each DOF in turn by a tiny amount, say $\epsilon$, twice: forward and backward. Denote by $C_i^+$ and $C_i^-$ the perturbed configurations that result by displacing the $i^{th}$ DOF forward and backward, respectively. Denote the internal force vectors evaluated at $C_i^+$ and $C_i^-$, by $p_i^+$ and $p_i^-$, respectively. Then the $i^{th}$ row of $K$, denoted by $K_i$, can be approximated by the central difference formula as

$$K_{FD}^i = \frac{p_i^+ - p_i^-}{2\epsilon}. \quad (12.6)$$

12–7
Chapter 12: THE TL PLANE BEAM ELEMENT: IMPLEMENTATION

ClearAll[Ea, Ga, Ei, N0, V0, M0, eps, tol, numer];
tol = 0.001; eps = 0.001;
numer = True;
{X1, Y1} = {-3, 7}; {X2, Y2} = {-8, 11}; u1 = {-0.6, -0.9, 0}; u2 = {1.5, 2.1, 0};
XYcoor = {{X1, Y1}, {X2, Y2}}; X21 = X2 - X1; Y21 = Y2 - Y1;
L0 = Sqrt[X21^2 + Y21^2]; u1 = {0.1, 0.25, -0.3}; u2 = {0.45, 0.58, -0.6};
EA = 100; GA = 240; Ei = 300; (* GA = 12*Ei/L0^2; RBF *)
S = {EA, GA, Ei}; z0 = {10, -20, 30};
u0 = Flatten[{u1, u2}]; KFD = Table[0, {6}, {6}];
(* p0 = TLPlaneBeamIntForce[XYcoor, u1, u2, S, z0]; *)
For [i = 1, i <= 6, i++,
  u = u0; u[[i]] = u[[i]] + eps; uXYθ = {Take[u, 3], Take[u, -3]};
pplus = TLPlaneBeamIntForce[XYcoor, S, z0, uXYθ, numer];
u = u0; u[[i]] = u[[i]] - eps; uXYθ = {Take[u, 3], Take[u, -3]};
pminus = TLPlaneBeamIntForce[XYcoor, S, z0, uXYθ, numer];
KFD[[i]] = Flatten[Simplify[(pplus - pminus)/(2*eps)]];]
KFD = N[KFD];
Print["K by finite differences: "]; Print[KFD // MatrixForm];
u = u0; uXYθ = {Take[u, 3], Take[u, -3]};
K = TLPlaneBeamTanStiff[XYcoor, S, z0, uXYθ, numer];
Print["K by stiffness modules: "]; Print[K // MatrixForm];
Print["This diff should be zero or small", 
  " (Chop used with ", tol, ") tolerance")];
Print[Chop[K - KFD, tol] // MatrixForm];
Print[Chop[Eigenvalues[K]]];
Print[Chop[Eigenvalues[KFD]]];

Figure 12.8. Consistency check of the tangent stiffness matrix versus internal force vector of the TL plane beam element. The script uses central finite differences to numerically verify that K = ∂p/∂u.

Figure 12.9. Results from running the consistency check script listed in Figure 12.8.

12–8
Perform this operation for each DOF, and store the resulting rows to form matrix $K^{FD}$.

3. Compare the tangent stiffness matrix $K$ evaluated at $C$ with $K^{FD}$. It is a good idea to vary $\epsilon$ down until the entries of $K^{FD}$ stabilize.\(^1\)

4. If $K^{FD}$ and $K$ agree within numerical tolerance, repeat the test for other randomly selected input sets until confident that the implementations are consistent.

A consistency test for the TL plane beam element is illustrated with the script listed in Figure 12.8. The DOF perturbations are specified by variable $\text{eps}$, which is defined at the script start. The test output in Figure 12.9 gives some assurance that the implementations of the internal force vector and tangent stiffness matrix modules are consistent.\(^2\)

Remark 12.1. This test is invaluable for programming complex elements, for example spatial beams, plates and shells. In fact, there are some rough nonlinear problems, such as those that involve path-dependent nonlinear material behavior, in which the only way to practically implement $K$ is by finite differences.

Remark 12.2. Sometimes a priori inconsistencies are inevitable. This can happen if approximations are made in the formulation of $K$ (for example, linearizations) but not on that of $p$. Even in that case, comparing $K$ and $K^{FD}$ may shed light on whether the approximations are justified.

Remark 12.3. If consistency is expected but the test fails, at least one of the modules is wrong. One telling case is when the problem is conservative but $K^{FD}$ comes out unsymmetric. That points to errors in the internal force implementation, regardless of whether $K$ is correctly programmed.

§12.4. Cantilever Beam Structure

The remaining modules are specialized to the simple structure shown in Figure 12.10. This is a cantilever plane beam-column of length $L$, modulus $E$, area $A$ and inertia $I$, loaded by an axial force $P$ as shown. It moves in the plane of the figure. The structure is discretized into $N_e$ TL plane beam elements of equal length. It has $N_e + 1$ nodes and $3N_e + 3$ degrees of freedom (DOF). Upon suppression of the three DOF at the fixed end node the number of DOF is reduced to $3N_e$.

§12.4.1. Cantilever Beam as Benchmark

The objective is to compute the classical buckling load $P_{cr}$ depicted in Figure 12.10(b) using FEM discretizations of varying $N_e$, and compare those results with the known analytical value. In classical buckling analysis, all deformations prior to buckling are neglected. Consequently $C^0$ coalesce, as indicated in Figure 12.10(a). The structure stiffness $K = K_M + K_G$ is evaluated on the coalescing $C^0 \equiv C$, configurations, with $K_G$ evaluated from the internal force state at $C^0$. This state is easily evaluated from statics to be $F_0 = P$, others zero.

This stability model just described is called linearized prebuckling or LPB, and is studied in detail in the part of the course dealing with stability. Under those assumptions, $K_M$ is constant while $K_G$ is linear in the applied forces. Critical point analysis then leads to a linear algebraic eigensystem called the buckling eigenproblem. The smallest eigenvalue characterizes the critical load, which for the LPB model can be shown always to be a bifurcation point.

\(^1\) But $\epsilon$ should not be set too small, as cancelling errors eventually may kill accuracy in (12.6).

\(^2\) Note the use of the Chop function of Mathematica to replace small numbers by exact zeros to clean up the output.
The classical buckling load for the configuration of Figure 12.10 is \( P_{cr} = \lambda_{cr} EI / L^2 \) with \( \lambda_{cr} = -\frac{1}{4} \pi^2 = -2.467401100272339 \) (negative because \( P \) has to compress the beam to achieve buckling). Scripts used to carry out the FEM buckling analysis with two different methods are described in §12.4.5 and §12.4.6.

§12.4.2. Cantilever Beam Internal Force Assembler

The module that assembles and returns the master internal force vector for the cantilever beam problem is listed in Figure 12.11. It is invoked by the following calling sequence:

\[
p = \text{CantBeamMasterIntForce}([L,P,Ne],[Em,A0,I0],u,\text{numer}];
\] (12.7)

The arguments in the calling sequence are

- **XYcoor** List of \( \{X,Y\} \) coordinates of element end nodes in reference configuration, arranged as \( \{\{X_1,Y_1\},\{X_2,Y_2\}\} \)
- **L** Beam length \((L = L_0)\)
- **P** Axial force \(P\)
- **Ne** Number of elements along span
- **Em** Elastic modulus
- **A0** Cross section area
- **I0** Second moment of inertia that governs plane bending in \(\{X,Y\}\) plane
- **u** A vector of \(3N_e + 3\) initial displacements. Normally set to all zeros.
- **numer** A logical flag: True to request numerical floating point work. Else set to False.

The output of the module is returned as function value:

\[
p \quad \text{Master internal force vector arranged a } 3N_e \times 1 \text{ two-dimensional list. Forces corresponding to the DOF of fixed-end node 1 are not returned as part of this vector.}
\]

Note that MacNeal’s Residual Bending Flexibility (RBF) “shear unlocking” trick is done by feeding \(12EI_0/L^2\) as second entry of argument \(S0\) of module TLPlaneBeamIntForce. As a result, neither the shear modulus \(G\) nor the shear area \(A_s\) need to be specified.
§12.4 CANTILEVER BEAM STRUCTURE

CantBeamMasterIntForce[{L_, P_, Ne_}, {Em_, A0_, I0_}, u_, numer_]:=
Module[{e, numnod, numdof, u1, u2, u3, Le, X1, X2, S0, z0,
   XYcoor, eftab, pe, p},
  numdof=3*numnod-3; Le=L/Ne;
  numnod=Ne+1;
  p=Table[0, {numdof}, {1}];
  z0={P, 0, 0};
  S0={Em*A0, 12*Em*I0/Le^2, Em*I0};
  (* RBF *)
  For [e=1, e<=Ne, e++,
    X1=(e-1)*Le; X2=e*Le;
    XYcoor={{X1, 0}, {X2, 0}};
    If [e==1, u1={0, 0, 0}; u2=Take[u, {3*e-2, 3*e}];
      eftab={0, 0, 3*e-2, 3*e-1, 3*e}];
    If [e>1, u1=Take[u, {3*e-5, 3*e-3}]; u2=Take[u, {3*e-2, 3*e}];
      eftab={3*e-5, 3*e-4, 3*e-3, 3*e-2, 3*e-1, 3*e}];
    pe=TLPlaneBeamIntForce[XYcoor, S0, z0, {u1, u2}, numer];
    (*Print["pe=",pe];*)
    p=MergeElemIntoMasterIntForce[pe, eftab, p];
  ];
  Return[Simplify[p]]
]

MergeElemIntoMasterIntForce[pe_, eftab_, pm_]:=
Module[{i, ii, nf=Length[eftab], p},
  p=pm;
  For [i=1, i<=nf, i++,
    ii=eftab[[i]];
    If [ii<=0, Continue[]];
    p[[i,1]]+=pe[[ii,1]]
  ];
  Return[p]
]

Figure 12.11. Module that assembles the master internal force vector for the cantilever beam problem of Figure 12.10.

CantBeamMasterTanStiff[{L_, P_, Ne_}, {Em_, A0_, I0_}, u_, numer_]:=
Module[{e, numnod, numdof, u1, u2, u3, Le, X1, X2, S0, z0,
   XYcoor, uXYθ, eftab, pe, p},
  numdof=3*numnod-3; Le=L/Ne;
  numnod=Ne+1;
  K=Table[0, {numdof}, {numdof}];
  z0={P, 0, 0};
  S0={Em*A0, 12*Em*I0/Le^2, Em*I0};
  (* RBF *)
  For [e=1, e<=Ne, e++,
    X1=(e-1)*Le; X2=e*Le;
    XYcoor={{X1, 0}, {X2, 0}};
    If [e==1, u1={0, 0, 0}; u2=Take[u, {3*e-2, 3*e}];
      eftab={0, 0, 3*e-2, 3*e-1, 3*e}];
    If [e>1, u1=Take[u, {3*e-5, 3*e-3}]; u2=Take[u, {3*e-2, 3*e}];
      eftab={3*e-5, 3*e-4, 3*e-3, 3*e-2, 3*e-1, 3*e}];
    Ke=TLPlaneBeamTanStiff[XYcoor, S0, z0, {u1, u2}, numer];
    (* Print["Ke=", Ke//MatrixForm]; *)
    K=MergeElemIntoMasterStiff[Ke, eftab, K]
  ];
  Return[Simplify[K]]
]

MergeElemIntoMasterStiff[Ke_, eftab_, Km_]:=
Module[{i, j, ii, jj, nf=Length[eftab], K},
  K=Km;
  For [i=1, i<=nf, i++, ii=eftab[[i]];
    If [ii<=0, Continue[]];
    For [j=i, j<=nf, j++, jj=eftab[[j]];
      If [jj>0,
        K[[jj, ii]]+=Ke[[i, j]]
      ]
  ];
  Return[K]
]

Figure 12.12. Module that assembles the master tangent stiffness matrix for the cantilever beam problem of Figure 12.10.
CantBeamMasterIntForce uses an auxiliary module MergeElemIntoMasterIntForce, also listed in Figure 12.11, to merge the element internal force vector into the master internal force of the full structure. This assembler module is actually not used in this Chapter. It is provided for use in later Chapters.

§12.4.3. Cantilever Beam Tangent Stiffness Assembler

The module that assembles and returns the master tangent stiffness for the cantilever beam problem is listed in Figure 12.12. It is invoked by the following calling sequence:

$$K = \text{CantBeamMasterTanStiff}[[L,P,Ne],[Em,A0,I0],u,\text{numer}]; \quad (12.8)$$

The arguments in the calling sequence are exactly the same as those described for the master internal force module CantBeamMasterIntForce in the previous subsection.

The output of the module is returned as function value:

$$K$$

Master tangent stiffness matrix as a $3N_e \times 3N_e$ two-dimensional list. Rows and columns that pertain to the three DOF of fixed-end node 1 are not returned as part of this matrix.

The returning $K$ is the sum of the material and geometric stiffnesses. They can be separated \textit{a posteriori} using the \texttt{Coefficient} function of \textit{Mathematica} if the only internal force is $P$, which is supplied symbolically. This operation is described in detail in §12.4.6 below.

Note that MacNeal’s Residual Bending Flexibility (RBF) “shear unlocking” trick is done by feeding $12EI_0/L^2$ as second entry of argument $S0$ of module TLPlaneBeamIntForce. As a result, neither the shear modulus $G$ nor the shear area $A_s$ need to be specified.

CantBeamMasterTanStiff uses an auxiliary module MergeElemIntoMasterTanStiff, also listed in Figure 12.12, to merge the element tangent stiffness matrix into the master tangent stiffness of the full structure. This assembler module is used in the scripts presented in §12.4.5 and §12.4.6.

§12.4.4. Critical Load Finding Methods

For a buckling stability analysis based on the LPB assumptions (outlined in the previous subsection) the critical load $P_{cr}$ may be obtained using two numerical procedures:

1. Find the roots of the determinant expressed as a polynomial in the axial load $P$ or, equivalently, the load multiplier $\lambda$. This is easy to code if the programming language provides a determinant evaluator and a polynomial root solver. The root closest to zero (if real) is of primary interest. The method has a serious drawback: it “blows up” as the structure gets more complicated that just a few elements. (More details are provided in §12.4.5. A minor defect is that the computation of the associated buckling mode shapes, if wanted, is not straightforward and requires separate coding.

2. Separate the material and geometric stiffness evaluated at $C \equiv C^0$, and solve the stability eigenproblem

$$K_M z_i = \lambda_i K_G z_i. \quad (12.9)$$

This is a symmetric, generalized, algebraic eigenproblem. The eigenvalues $\lambda_i$ are critical values of the load parameter $\lambda$, while the associated eigenvectors $z_i$ provide buckling mode shapes. If either $K_M$ or $K_G$ is nonnegative, the eigenvalues $\lambda_i$ are guaranteed to be real. The
§12.4 CANTILEVER BEAM STRUCTURE

The script listed in Figure 12.13 obtains critical loads for the cantilever beam structure of Figure 12.10, using the zero-stiffness-determinant condition. This is also called the singular stiffness criterion, and it was introduced in Chapter 5. We set $E = I_0 = A_0 = L_0 = 1$ from the start to simplify computations, while leaving $P = \lambda$ as symbolic variable. Everything else is numeric. The only nonzero internal force is $F_0 = P$, which is the same for all elements. The tangent stiffness $K$ is assembled using the module CantBeamMasterTanStiff described in §12.4.3.

The determinant $\det(K)$ is formed explicitly using the Det function of Mathematica. This function returns $\det(K) = \det[K]$ as a polynomial in $P$. This is called the characteristic polynomial. All of its roots are numerically computed via function NSolve. The root closest to zero is taken to define the critical load of the FEM discretization. Note that for this problem that value is negative.

As is, the script is set to run the C-style For loop twice. The number of elements $N_e$ is preset to 1, and is doubled after every pass. Thus the posted script will do 1 and 2 element discretizations. The output results are shown in Figure 12.14. To continue the computations for more elements, say 4, 8, ...
ClearAll[L,P,Em,A0,I0]; Em=A0=I0=L=1; Ne=1; numer=False;
For [k=1, k<=2, k++,
Print["Number of elements=",Ne];
LPNe={L,P,Ne}; EAI={Em,A0,I0}; u=Table[0,{3*Ne+3}];
K=CantBeamMasterTanStiff[{L,P,Ne},EAI,u,numer];
KM=Coefficient[K,P,0]; KG=Coefficient[K,P,1];
A=LinearSolve[N[KM],N[KG]];(*Print["KM=",KM//MatrixForm];Print["KG=",KG//MatrixForm];*)
(*Print["A=",A//MatrixForm];*)
emax=-Max[Eigenvalues[A]]; Print["FEM lambda cr=(1/emax)"; Ne=2*Ne;];
Print["exact buckling lambda coeff is ",-N[Pi^2/4]];

Figure 12.15. Script for buckling analysis of cantilever beam of Figure 12.10, carried out by solving
the stability eigenproblem (12.9). The For loop on k runs twice, and so the script goes over 1 and 2 element
discretizations.

Number of elements=1
FEM lambda cr=-3.16515
Number of elements=2
FEM lambda cr=-2.60391
exact buckling lambda coeff is -2.4674

Figure 12.16. Results from running the script shown in Figure 12.15 for discretizations of 1 and 2 elements.

eetc., modify the loop limit on k appropriately. This is done in a homework exercise.

For computations with a larger number of elements, say 8 or higher, it is important to set the logical
flag numer to False, so that the characteristic polynomial is exact (meaning that its coefficients are
exact integers). If the flag is set to True, soon one is faced with spurious complex roots because of
insufficient numerical precision, and all hell breaks loose.

§12.4.6. Buckling Analysis Via Stability Eigenproblem

The script listed in Figure 12.15 obtains critical loads for the cantilever beam structure of Fig-
ure 12.10, using the stability eigenproblem defined in (12.9). As in the script of Figure 12.13,
we set $E = I_0 = A_0 = L_0 = 1$ from the start to simplify computations, while leaving $P = \lambda$ as
symbolic variable. Everything else is numeric. The only nonzero internal force is $F_0 = P$, same for
all elements. The tangent stiffness $K$ is assembled using the module CantBeamMasterTanStiff
described in §12.4.3.

As this point the script takes a different path. To set up the eigenproblem (12.9) it is necessary to
separate the material and geometric stiffness components. For this one takes advantage that $K$ is
known to have the form

$$K = K_M + P \hat{K}_G. \quad (12.10)$$

in which $K_M$ and $\hat{K}_G$ are numeric matrices and $P$ is the symbolic variable that stands for axial
force. Matrix $\hat{K}_G$ is called the normalized geometric stiffness, which is simply $K_G$ for $P = 1$. The
extraction is easily done with the Coefficient function of Mathematica. The pertinent statements
in the script are

$$KM=\text{Coefficient}[K,P,0]; \quad KG=\text{Coefficient}[K,P,1]; \quad (12.11)$$
§12.4 CANTILEVER BEAM STRUCTURE

Shanks[Z_] := Module[{m, n, shankedZ},
If [Length[Dimensions[Z]] <= 1, n = Length[Z];
   If [n <= 2, Return[{}]]; shankedZ = Table[0, {n - 2}];
   Do [
     shankedZ[[j]] = (Z[[j + 2]]*Z[[j]] - Z[[j + 1]]^2)/
       (Z[[j + 2]] - 2*Z[[j + 1]] + Z[[j]]),
     {j, 1, n - 2}]; Return[shankedZ];
   ];
{m, n} = Dimensions[Z];
If [n <= 2, Return[{{}}]]; shankedZ = Table[0, {m}, {n - 2}];
Do [
  Do [
    shankedZ[[i, j]] = (Z[[i, j + 2]]*Z[[i, j]] - Z[[i, j + 1]]^2)/
      (Z[[i, j + 2]] - 2*Z[[i, j + 1]] + Z[[i, j]]),
  {j, 1, n - 2}], {i, 1, m}];
Return[shankedZ];
]

Z = {−3.165151389911680, −2.603908889389283, −2.499688345543354,
   −2.475364052205859, −2.469385123708907, −2.467896687873336,
   −2.467524972077594, −2.467432067643745};
Print["Shanked sequence=", Shanks[Z] // InputForm,
   " vs. exact: -2.467401100272339"];
Print["Re-Shanked sequence=", Shanks[Shanks[Z]] // InputForm,
   " vs. exact: -2.467401100272339"];
Print["Re-Re-Shanked sequence=", Shanks[Shanks[Shanks[Z]]] // InputForm,
   " vs. exact: -2.467401100272339"];

Figure 12.17. Extrapolation of cantilever beam buckling analysis results produced by the script of Figure 12.15, with 1, 2, 4, ..., 128 elements along the structure. Extrapolation done by Shanks’ method implemented as Wynn’s epsilon algorithm. Script performs three recursions.

Shanked_sequence = {−2.475921606110241, −2.467958557672605, −2.467436534172408,
   −2.4674033252507503, −2.467401241305521, −2.4674011103913855} vs. exact: -2.467401100272339
Re-Shanked_sequence = {−2.4673999117262855, −2.4674010691141155,
   −2.467401101778348, −2.46740110150284} vs. exact: -2.467401100272339
Re-Re-Shanked_sequence = {−2.467401102526432, −2.467401102690187} vs. exact: -2.467401100272339

Figure 12.18. Results from running the Shanks-extrapolation script of Figure 12.17.

in which $K_G$ is actually $\hat{K}_G$. Since $P \equiv \lambda$, the stability problem can be set up as a slight variation of (12.9):

$$K_M z_i = \lambda_i \hat{K}_G z_i.$$  \hfill (12.12)

Here we run into a procedural difficulty. *Mathematica* (unlike *Matlab*) has no built-in function to directly process the generalized symmetric eigenproblem (12.12). It is necessary to transform it to the standard form. The transformation can be done easily if either $K_M$ or $\hat{K}_G$ is nonsingular. Assume the former. Premultiply both sides of (12.12) by $K_M^{-1}$ to get

$$z_i = \lambda_i K_M^{-1} \hat{K}_G z_i.$$  \hfill (12.13)

Calling $A = K_M^{-1} \hat{K}_G$ and $\mu_i = 1/\lambda_i$, this can be recast as the standard algebraic eigenproblem

$$A z_i = \mu_i z_i.$$  \hfill (12.14)

The most efficient way to form $A$ is to solve the multiple-right-hand-side linear system:

$$K_M A = K_G.$$  \hfill (12.15)
In the script of Figure 12.15, this is done by the statement

$$A = \text{LinearSolve}[N[KM], N[KG]];$$

(12.16) in which $N[.]$ converts its matrix argument to numeric floating point to speed up LinearSolve. The Mathematica function Eigenvalues returns all eigenvalues $\mu_i$ of $A$ as a list. Since we are interested in the smallest $\lambda_i$ as critical load factor, we pick the largest $\mu_i$ (with appropriate sign) and reciprocate it. That completes the computations for a specific FEM discretization.

The output results for one and two elements is shown in Figure 12.16. This computation is continued for more elements in a homework exercise.

§12.4.7. Extrapolation of Buckling Load Results

Running the stability-eigenproblem script of Figure 12.15 for 128 elements one obtains a buckling load multiplier of $\lambda_{cr} = -2.467432067643745$. This agrees with the analytical value $\lambda_{cr} = -\frac{1}{4}\pi^2 = -2.467401100272339$ to 5 figures.

Additional accuracy can be gained from the extrapolation of the sequence of critical loads for 1, 2, 4, ... 128 elements using the so-called Shanks’ transformation. This is most efficiently implemented as Wynn’s epsilon algorithm. See, for example, the monograph [885], or the Wolfram MathWorld web site at [http://mathworld.wolfram.com/WynnsEpsilonMethod.html](http://mathworld.wolfram.com/WynnsEpsilonMethod.html), which has additional references.

The sequence extrapolation is done by the script listed in Figure 12.17. Running it we obtain the results shown in Figure 12.18. The extrapolated value of $\lambda_{cr}$ agrees with $-\frac{1}{4}\pi^2$ to 9 places. For an engineer this would be overkill. But there are some problems in Astronomy and Atomic Physics that require this high precision, so the extrapolator is handy.
Homework Exercises for Chapter 12
The TL Plane Beam Element: Implementation

EXERCISE 12.1 [A+N:20]
Consider a plane prismatic beam of length $L_0$, cross section area $A_0$ and second moment of inertia $I_0$. In the reference configuration the beam extends from node 1 at $(X = Y = 0)$ to node 2 at $(X = L_0, Y = 0)$. The beam is under axial prestress force $N^0 \neq 0$ in the reference configuration, while both $V^0$ and $M_0$ vanish.

(a) Obtain the internal force $p$, the material stiffness matrix $K_M$ and the geometric stiffness matrix $K_G$ in the reference configuration, for which $u_{X1} = u_{Y1} = u_{X2} = u_{Y2} = \theta_1 = \theta_2 = 0$.

(b) The beam rotates 90° rigidly to a current configuration for which $u_{X1} = u_{Y1} = 0, u_{X2} = -L_0, u_{Y2} = L_0, \theta_1 = \theta_2 = 90^\circ$. Check that the stress resultants do not change (that is, $N = N_0$ and $V = M = 0$ because $e = \gamma = \kappa = 0$), and obtain $K_M$ and $K_G$ in that configuration.

EXERCISE 12.2

[N:20] Analyze the pure bending of a cantilever discretized into an arbitrary number of elements. (Incomplete, to be expanded some day)

EXERCISE 12.3

[A/C: 20] Consider the linear Timoshenko plane beam element shown in Figure E12.1. This Figure is taken from [?, Chapter 13], with several notational changes to fit this course. The element has length $L$, and is referred to Cartesian axes $\{X, Y\}$ as shown. The elastic modulus $E$, shear modulus $G$, second cross-section moment of inertia $I$ and shear cross-section area $A_s$ are constant over the element span. The element has four degrees of freedom (DOF), configured as

$$u_b = [u_{Y1}, \theta_1, u_{Y2}, \theta_2]^T.$$

(E12.1)

The optimal linear stiffness matrix for this element, derived in [?] using cubic Legendre polynomials, is

$$K_{opt} = \frac{EI}{L^3 (1 + \Phi)} \begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & L^2 (4 + \Phi) & -6L & L^2 (2 - \Phi) \\ -12 & -6L & 12 & -6L \\ 6L & L^2 (2 - \Phi) & -6L & L^2 (4 + \Phi) \end{bmatrix},$$

(E12.2)

3 Optimal in the sense of nodal exactness, as proven in that reference. It is also free of shear locking for any $\Phi$. 

12–17
in which
\[ \Phi = \frac{12EI}{GA_s L^2}, \]  
(E12.3)

is a nonnegative dimensionless ratio that characterizes the shear slenderness of the beam element. (It is not an intrinsic beam property because it involves the element length.) If \( \Phi = 0 \), (E12.2) reduces to the well known linear stiffness matrix of the Bernoulli-Euler (BE) beam element; see for example [?, Chapter 12]. This is also known to be optimal.

The objective of this exercise is to derive MacNeal’s Residual Bending Flexibility (RBF) shear unlocking correction for the TL plane beam element formulated in the previous Chapter. Steps:

(a) Get the material stiffness matrix \( K_M \) of the TL plane beam element in the reference configuration \( C^0 \) either using the module TLPlaneBeamMatTanStiff documented in §12.3.1, or by hand from the results of the previous Chapter. As input data take: node coordinates \( \{ X_1 = Y_1 = 0 \} \) and \( \{ X_2 = L, Y_2 = 0 \} \), all node displacements and rotations zero, all initial (prestress) forces zero, elastic modulus \( E \), shear modulus \( G \), second moment of inertia \( I \), cross section area \( A \), and shear area \( A_s \). Important: \( L, E, G, A \) and \( A_s \) should be kept symbolic. Instead of specifying \( GA \), shear rigidity, however, input an unknown symbolic variable \( R \), which is to be determined in (c) below. (If you use TLPlaneBeamMatTanStiff, plug \( R \) as the second entry of argument \( \Phi \).

(b) The resulting \( 6 \times 6 \) matrix \( K_M \) produced by item (a) should be reduced to a \( 4 \times 4 \) by removing rows and columns 1 and 4 that correspond to axial (bar) displacements, so that

\[ K_M = \begin{bmatrix}
K_{11} & K_{12} & K_{13} & K_{14} & K_{15} & K_{16} \\
K_{21} & K_{22} & K_{23} & K_{24} & K_{25} & K_{26} \\
K_{31} & K_{32} & K_{33} & K_{34} & K_{35} & K_{36} \\
K_{41} & K_{42} & K_{43} & K_{44} & K_{45} & K_{46} \\
K_{51} & K_{52} & K_{53} & K_{54} & K_{55} & K_{56} \\
K_{61} & K_{62} & K_{63} & K_{64} & K_{65} & K_{66}
\end{bmatrix} \quad \Rightarrow \quad K_{M_b} = \begin{bmatrix}
K_{22} & K_{23} & K_{25} & K_{26} \\
K_{32} & K_{33} & K_{35} & K_{36} \\
K_{52} & K_{53} & K_{55} & K_{56} \\
K_{62} & K_{63} & K_{65} & K_{66}
\end{bmatrix}. \]  
(E12.4)

Set up also the optimal matrix (E12.2) keeping \( \Phi \) as a symbol; that is, don’t replace it yet by (E12.3).

(c) The entries of the \( 4 \times 4 \) matrix in (E12.4) will be functions of \( R \). Now find the \( R \) that makes this matrix coalesce, like magic, with the optimal stiffness (E12.2). The magic expression will be a function of \( E, I, L \) and \( \Phi \) (Note that \( G \) and \( A_s \) will indirectly come in through \( \Phi \).) By now using (E12.3), verify that it agrees with that quoted in Remark ? of the previous Chapter.

If you use Mathemtica for this Exercise, you may want to use Solve to do item (c). Assuming that the \( 4 \times 4 \) matrices of (E12.4) and (E12.2) are in arrays \( \text{KMb} \) and \( \text{Kopt} \), respectively, this could be done as

\[
\text{sol=}\text{Simplify}[\text{Solve}[\text{KMb}[[1,1]]==\text{Kopt}[[1,1]],R]]; \quad \text{Print[sol]}; \quad \text{Rbest=R/.sol[[1]]}; \quad \text{Print["verification: ",-\text{Simplify}[(\text{KMb}/.R->\text{Rbest})-\text{Kopt}]]/\text{MatrixForm}];
\]

**EXERCISE 12.4**

[N:25] This exercise pertains to the cantilever beam-column shown in Figure 12.10, which is described at the start of §12.4. The objective is to compute critical loads as roots of the characteristic polynomial \( \det(K) \), and pick up that closest to zero. To do that, run the script of Figure 12.13 until the largest number of elements \( N_e \) that can be reasonably handled by your Mathemtica version (this will depend on memory and CPU speed). Termination can be controlled by adjusting the C-style loop on \( k \), which doubles \( N_e \) on each pass.

Report on the following items:

(a) Your computed \( \lambda_{cr} \)'s for the \( N_e \) you were able to run. Comment on whether they seem to converge toward \(-\pi^2/4\). If convergence is observed, is it monotonic? Is it linear or quadratic? (FEM theory says that eigenvalues converge quadratically.)

12–18
(b) Why does \texttt{NSolve} give \(2N_e\) roots?\(^4\) Recall that an \(N_e\)-element model has \(3N_e\) freedoms; where have the missing \(N_e\) roots gone? Hint: Print matrix \(K_G\) for a small \(N_e\); what do the zero rows and columns tell you?

(c) Why are the roots real?\(^5\)

(d) Why do the computations get noticeably slower as \(N_e\) increases?\(^6\)

**EXERCISE 12.5**

[N:20] This also pertains to the cantilever beam-column shown in Figure 12.10. The brute-force technique used to find \(\lambda_{cr}\) in the previous exercise is easy to implement if you have a polynomial root solver and a determinant evaluator at your disposal. But it becomes inefficient and unreliable as the number of elements increases.\(^7\)

A more effective technique that uses the stability eigenproblem is implemented in the script of Figure 12.15. Some of the algebraic manipulations therein are explained in §12.4.6. This script works fine on my Mac PowerBook up to 128 elements, beyond which memory and CPU time requirements grow too large (it actually runs out of RAM at 512 elements after about 30’ of grinding).

Run the script of Figure 12.13 until the largest number of elements \(N_e\) that can be reasonably handled by your \texttt{Mathematica} version (this will depend on available memory and CPU speed). Termination can be controlled by adjusting the C-style loop on \(k\), which doubles \(N_e\) on each pass. As answer to this exercise, report on the following:

(a) How far were you able to go?

(b) Are the critical loads the same as those obtained in the previous exercise? (Just in case, they should agree with those stored in list \(Z\) in the “extrapolation script” of Figure 12.17.)

\(^4\) Only \(N_e\) of which are of practical interest; this can be verified by computing and plotting eigenvectors, but that is not part of this Exercise.

\(^5\) Assuming that exact integer arithmetic is used to form the determinant. If you evaluate the determinant in floating point by setting \texttt{numer=True} at script start, complex roots will soon emerge because of limited precision and the fact that some polynomial roots are extremely sensitive to tiny changes in coefficients.

\(^6\) If you can’t guess, try printing the characteristic polynomials for moderate \(N_e\). The CPU time may go up exponentially in \(N_e\), depending on how clever the polynomial root solver is. The one supporting \texttt{NSolve} is OK.

\(^7\) Forming the characteristic polynomial by expanding \(\det(K)\) is frowned upon by numerical analysts for several reasons, one of which is that the coefficients tend to get enormously large. This requires either extended precision if done in exact integer arithmetic (as in the script of Figure 12.13), or rapidly leads to exponent overflow if done in floating point.