PDHonline Course S281 (5 PDH)

Tapered Frames and Monopoles

Instructor: Marvin Liebler, P.E.

2014

PDH Online | PDH Center

5272 Meadow Estates Drive
Fairfax, VA 22030-6658
Phone & Fax: 703-988-0088
www.PDHonline.org
www.PDHcenter.com

An Approved Continuing Education Provider

1. INTRODUCTION

Most structures are assemblies of prismatic beams, that is, the member cross-sections are constant with length. There are, however, some classes of structures which have members whose cross-sections vary with member length. Among these structures are metal building frames and monopoles. The next two pages illustrate these classes.

In both of the cases, the objective of the taper is to decrease material cost, putting the highest section modulus in the vicinity of the highest moment.

Modern analysis of structures composed of beams recognizes both first and second order analysis. First order analysis does not include the effects of moment magnification by node displacement (P-Δ effect) and by eccentricity of the beam center line from the chord joining the nodes (P-δ effect). Second order analysis, includes these effects.

Analysis of the assembly is done by the finite element method. Key to this method is the generation of the stiffness matrices for the various beam elements.

In first order analysis, derivation of the stiffness matrix is done explicitly, although significantly more complex for a tapered member as opposed to the prismatic case. Section 2. Described this process, section 3. deals with assembling the beams, and section 4. Gives examples.

The second order analysis stiffness matrix cannot, however, be solved explicitly (to the author's knowledge). Recourse is made to power series representations for displacement (unknown) and varying moment of inertia (known). Generation of the power series is done though use of Chebyshev polynomials, as shown in section 5. Section 6 shows the revised assembly method, and section 7. Concludes with examples, including a numerical check with an AISC standard.

© Marvin Liebler

This illustration depicts the basic framing of a metal building. The primary load-bearing frames are composed of columns and rafters of varying cross-section. The roof supporting members (purlins) and and wall members (girts) are of constant cross-section. All the members are prefabricated to assemble as a unit, and available as a package from a manufacturer.

Monopoles are characterized by support of a flag, highway sign, advertisement, or communication antennas and/or transmitters or receivers. This is done by a single member, circular or near-circular in cross-section, with diameter decreasing with elevation. Foundations may be drilled piers or spread footings. The primary load is not weight, but lateral force at the top. An external ladder, which may be open (as above) or caged is used to access the top.

2. FIRST ORDER STIFFNESS MATRICES

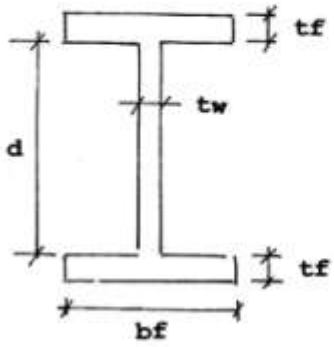
Two types of tapered beams cross-sections are considered, doubly symmetric I-beams and circular conduits with linear tapers.

A linear web taper is one in which the cross-section depth, d, is given by d(x) = d0 + (d1-d0)*x/L. Here d0 is the diameter at the near end of the beam and d1 is the diameter at the far end of the beam.

.

The following chart illustrates these two cross-sections, with defining equations for area and moments of inertia.

I-BEAM

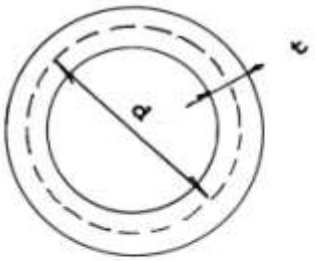


A = tw*d+2*bf*tf

I = tw*d^3/12 +
(d+tf)^2*bf*tf/2
+bf*tf^3/6

d = d0+(d1-d0)*x/L

CONDUIT



d = (doutside-dinside)/2

A = Π*d*t

I = Π*d*t*(d^2+t^2)/8

d = d0+(d1-d0)*x/L

d0 represents 'd' at first node (initial node) and d1 represents 'd' at second node.

Dimensions typically in inches.

The basic building blocks of any finite element program are the stiffness matrices of the elements comprising the structure to be examined.
Stiffness matrices relate the displacements (axial, transverse, and rotation) at each end to the forces (axial, shear, and moment) at each end of the beam. In shorthand matrix notation,

[f] = [K][d] where

[f] = 6x1 matrix of forces
[K] = 6X6 stiffness matrix
[d] = 6x1 matrix of displacements

[K] is expanded as :

$$\begin{bmatrix} k11 & k12 & k13 & k14 & k15 & k16 \\ k21 & k22 & k23 & k24 & k25 & k26 \\ k31 & k32 & k33 & k34 & k35 & k36 \\ k41 & k42 & k43 & k44 & k45 & k46 \\ k51 & k52 & k53 & k54 & k55 & k56 \\ k61 & k62 & k63 & k64 & k65 & k66 \end{bmatrix}$$

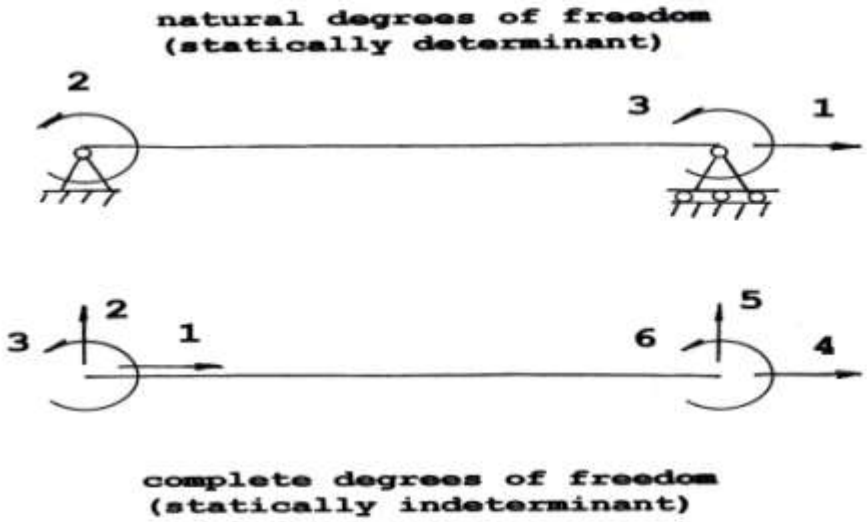
Typical units of forces are kips and kip-inches, with typical displacements of inches and radians. The units of the entries in the stiffness matrix are u(i/j) where i is the unit of the ith forces and j the unit of the jth displacement, with i and j each varying from 1 to 6. For example, the entry k23 relates force 2 (first end shear) to displacement 3 (first end rotation).
For prismatic beams the kij terms may be set down explicitly. However, with tapered beams, since the area and moment of inertia vary along the length, it is not as straightforward.

The method used here to find the first order stiffness matrix of a tapered beam is based on the work of Dr. L. Yaw, shown in Reference 1. It is based on two models of a beam, an element with six (6) degrees of freedom called the complete element, and one with three (3) degrees of freedom called the natural element. Three degrees of freedom must be fixed to obtain static equilibrium, i.e., the natural element. In the following drawing, note that the fixed degrees of freedom necessary to obtain the natural beam

from the complete beam are vertical support at each end and horizontal support at the first node.

The method is very general, as it may be used for any cross-section and taper functions which can be integrated.

2D BEAM ELEMENT



The method may be broken down into :
(1) The method of virtual forces
(2) Natural beam flexibility matrix (3x3)
(3) Natural beam stiffness matrix (3x3)
(4) Complete beam stiffness matrix (6x6)

2.1 The Virtual Force Method

In this method, the external work must be equal to the internal work.
External work = Σ QD and internal work = Σ qd where:
Q,q = virtual forces in equilibrium, i.e., forces

caused by a virtual load, external and internal, respectively
D,d = real compatible displacements, external and internal, respectively
Then QD = qd, and Q taken = 1

For axial loads,
n = result of virtual load (force) on internal structure
N = applied virtual external load
D = N*dx/E*A = differential real internal displacement (axial displacement at internal point)

$$\text{Thus } 1*D = \int_0^L \frac{n*N*dx}{EA}$$

For flexural loads :
m = result of virtual load (moment) on internal structure
M = applied virtual external load (moment)
dΘ = Mdx/EI = differential real internal displacement (angle, in radians, of internal point)

$$\text{Thus } 1*D = \int_0^L \frac{m*M*dx}{EI}$$

2.2 Natural Beam Flexibilty Matrix

For these calculations, let the flexibility matrix coefficient fij be the displacement in direction i caused by the force in direction j. It is true that the matrix is symmetric, that is, fij = fji.
The fij correspond to the 'D' values in section 2.1 above.

Consider first a virtual unit load in the natural d.o.f. direction 1 (the free end).
Here n= N =1 and f21 = f31 = f12 = f13 = 0, as there is no transverse or angular displacement due to an axial load.

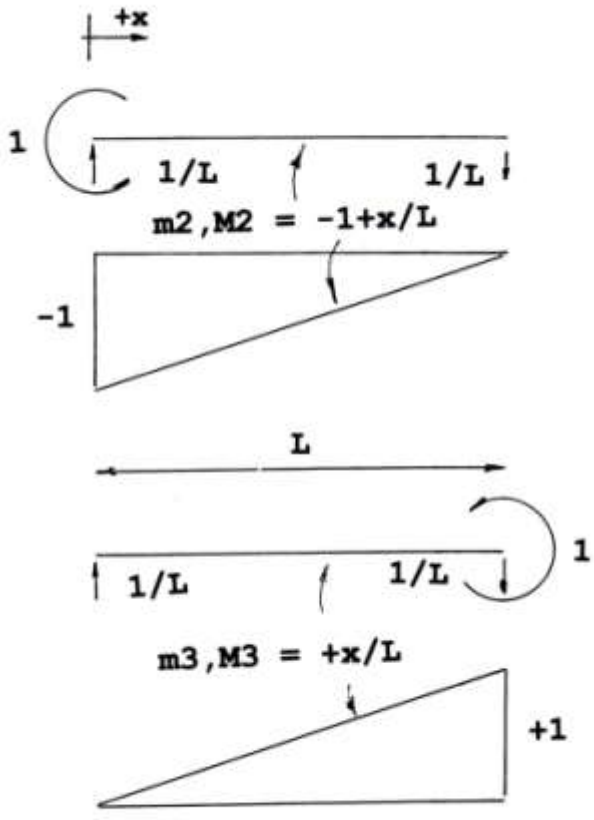
$$f11 = \int_0^L \frac{1*1\ *dx}{E*A}$$

Flexural terms and moment diagrams are shown next.

$$f22 = \int_0^L \frac{m2*M2*dx}{E*I} = \int_0^L \frac{(x/L-1)^2 *dx}{E*I}$$

$$F33 = \int_0^L \frac{m3*M3*dx}{E*I} = \int_0^L \frac{(x/L)^2 *dx}{E*I}$$

$$f23 = \int_0^L \frac{m2*M3*dx}{E*I} = \int_0^L \frac{(x/L-1)^2 *(x/L)*dx}{E*I}$$



For a prismatic (constant cross-section) beam, these coefficients may be obtained directly as:

f11 = +L/E*A
f22 = +L/3*E*I
f33 = +L/3.E*I
f23 = -L/6*E*I

Unfortunately these integrations are not as straightforward for a tapered beam, as the area and moment of inertia vary with distance along the beam.
The two cross-sections studied here may be explicitly integrated. Details are given in Appendix 1, for the interested reader.

2.3 Inversion of the 3x3 Flexibilty Matrix

The inverse of the 3x3 flexibility matrix is the 3x3 stiffness matrix, where the independent variables are now displacements, and the dependent variables forces, as:

[f] = [K][d]

The inverse of a non-singular (determinant not equal to zero) square matrix is the transpose of the signed minor matrix.
For the prismatic matrix described above, the flexibility matrix [fij] and stiffness matrix [kij] are shown as

$$[fij] = \begin{bmatrix} \frac{L}{A*E} & 0 & 0 \\ 0 & \frac{L}{3*E*I} & -\frac{L}{6*E*I} \\ 0 & -\frac{L}{6*E*I} & \frac{L}{3*E*I} \end{bmatrix}$$

Inverting [fij] we have :

$$[kij] = \begin{bmatrix} \frac{E*A}{L} & 0 & 0 \\ 0 & \frac{4*E*I}{L} & -\frac{2*E*I}{L} \\ 0 & -\frac{2*E*I}{L} & \frac{4*E*I}{L} \end{bmatrix}$$

Calculating [fij]*[kij] obtains the 3x3 identity matrix.

2.4 Transformation of [K]3x3 to [K]6x6

As shown in Reference 1, this transformation is done by [K 6x6] = Γ′*[K 3x3]*Γ where Γ is the 3x6 matrix:

$$\Gamma = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & +1/L & 1 & 0 & -1/L & 0 \\ 0 & +1/L & 0 & 0 & -1/L & 1 \end{bmatrix}$$

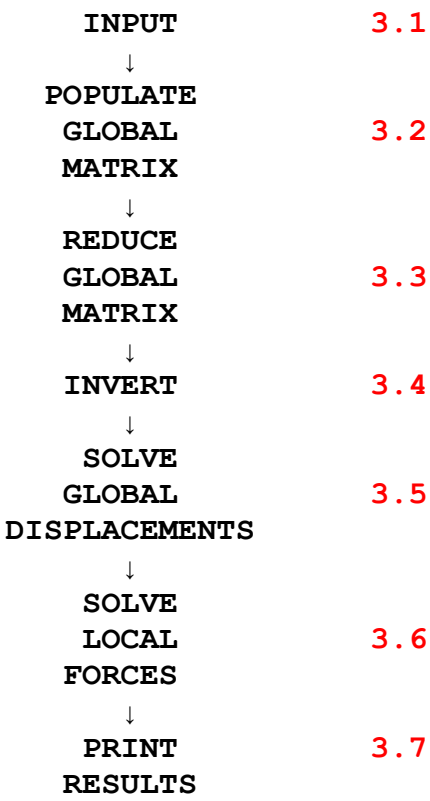
For a prismatic beam, the 6x6 stiffness matrix is:

$$\begin{bmatrix} +AE/L & 0 & 0 & -AE/L & 0 & 0 \\ 0 & +12EI/L^3 & +6EI/L^2 & 0 & -12EI/L^3 & +6EI/L^2 \\ 0 & +6EI/L^2 & +4EI/L & 0 & -6EI/L^2 & +2EI/L \\ -AE/L & 0 & 0 & +AE/L & 0 & 0 \\ 0 & -12EI/L^3 & -6EI/L^2 & 0 & +12EI/L^2 & -6EI/L^2 \\ 0 & +6EI/L^2 & +2EI/L & 0 & -6EI/L^2 & +4EI/L \end{bmatrix}$$

The programs developed are tpri.c and tprc.c for the tapered I-beam and conduit, respectively.

3. FIRST ORDER TAPERED BEAM ASSEMBLIES

Using the stiffness matrices derived in section 2., we can analyze complete structures, including metal building frames and momopoles. An outline of the programs is:



3.1 Input

The input quantities are:

- **coordinate matrix (node locations)**
- **beam connection matrix (end nodes)**
- **property vector for each beam type**
- **fixed degrees of freedom – unyielding support at node (x-displacement, y-displacement, rotation)**
- **load vector – applied loads at node points**

3.2 Populate Global Matrix

For each of the beams, the following four steps are required :

- **Find the local stiffness matrix for this particular beam.**

- **Find the transform matrix relating local beam coordinates to the global beam coordinates.**
- **Calculate the beam stiffness matrix in global coordinates, using the transform matrix times the local matrix.**
- **Add the thirty-six (36) elements of this matrix to the assembly global matrix.**

3.3 Reduce Global Matrix

Here we delete the row and column of the global stiffness matrix for each fixed degree of freedom. For example, consider a structure with four (4) nodes. Each node has three (3) degrees of freedom (x-displacement, y-displacement, and rotation) due to applied loads.
The number of rows and columns are 4x3, obtaining a 12x12 square matrix. Assume two nodes are pinned, fixing four (4) degrees of freedom. The resulting reduced matrix is then 8x8, i.e., only eight degrees of freedom may vary from zero upon application of loads.

3.4 Invert

Inverting the global stiffness matrix yields the global flexibility matrix, the displacements as a function of applied forces for the particular degrees of freedom chosen. This matrix has the same dimensions as the global stiffness matrix.

3.5 Solve Global Displacements

[global displacements] = [flex matrix]*[node loads]

3.6 Solve Local Forces

For each of the beams, three steps are required:

- **Again find the stiffness matrix for this beam in local coordinates.**
- **Transform the global displacements for this beam to local coordinates.**
- **Solve for beam forces in local coordinates: [forces] = [stiffness matrix]*[displacements]**

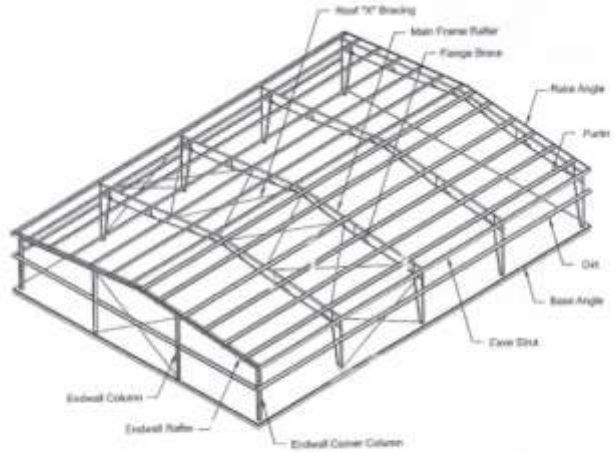
3.7 Print Results

Print global displacements (three for each node) and local beam forces (axial, shear, and moment) at each end of each beam. It should be noted that this is a single pass program.

4. FIRST ORDER EXAMPLES

4.1 Metal Buildings

Metal building structures are typically comprised of interdependent roof systems, wall systems, secondary framing members, primary framing members, and foundation-earth systems. The basic stick diagram of a metal building from Reference 2 and construction of a school building in Buffalo, New York (8/2011) are shown.



The exterior roof and wall systems are typically metal panels, cold-worked to provide flexural strength in one direction. They are attached to purlins (roof) or girts (walls) by field welds and/or screws. Detailing of corners and ends is particularly important as they protect the

building interior from the elements. Their in-plane shear resistance depends both on material thickness and number and arrangement of connections to supporting secondary members. A roof diaphragm is formed by some combination of the connected deck, tension rods, and compression struts (pipes or reinforced purlins).

The secondary members, purlins and girts, are typically cold-formed, as opposed to hot-rolled, steel members. Shapes include C, double C, Z, nested Z, and hats. Their analysis and design is to AISI (American Iron and Steel Institute) specifications, rather than those of the AISC (American Institute of Steel Construction, Reference). A major advantage of cold-formed members is the significant savings in member weight.

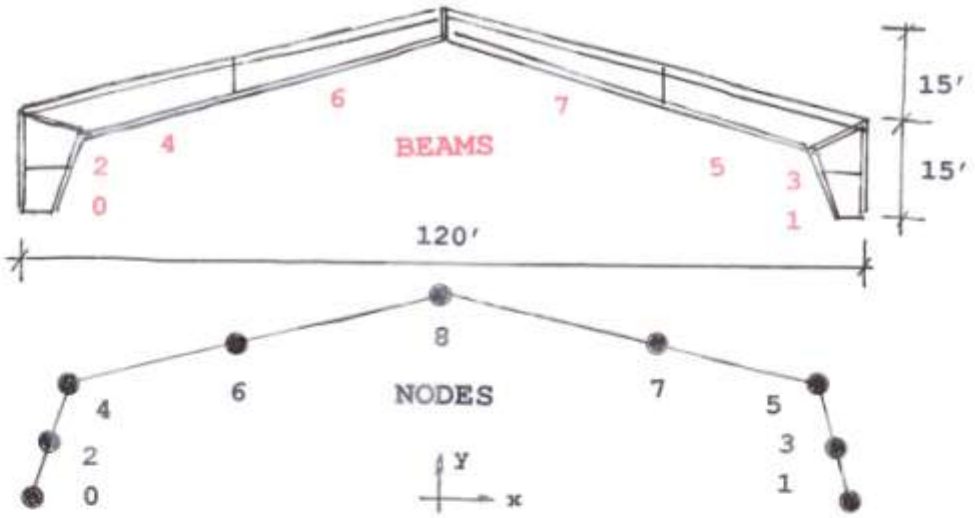
The purlins and girts frame into the primary frame members, internally pinned portal or gable frames, and external (end walls) either non-expandable post-and-beam, or expandable pinned portal or gable frames. The moment-resisting frames are composed of tapered columns and beams, with cross-sections chosen to provide a much better match to the moment and shear diagrams over their length, than can prismatic members.

Example 4.1

Consider a frame with the following characteristics :

Span = 120′
Eave height = 15′
Ridge height = 30′
Vertical loads as shown
Bases modeled as pins.
Outline of configuration and F.E. model shown on next page.

Find moment and axial load on members 0,2,4, and 6 which are the same as those of members 1,3,5,and 7 respectively, because of symmetry.
Also compare deflections, axial loads, and moments for pinned condition and release of one horizontal restraint.



Node	x(in.)	y(in.)
0	-711.000	0.000
1	+711.000	0.000
2	-704.000	90.000
3	+704.000	90.000
4	-698.344	162.715
5	+698.344	162.715
6	-360.000	252.000
7	+360.000	252.000
8	0.000	347.000

beam	d1(in.)	d2(in.)
0,1	14.000	28.000
2,3	28.000	39.312
4,5	41.398	32.000
6,7	32.000	22.000

All bf = 12.00 in.
All tf = 2.00 in.
All tw = 0.75 in.

d0,d1 = web depth exclusive of flanges

Nodes 4 and 5 , and web depths at nodes 4 and 5, calculated at Intersections of column and rafter center lines.

Vertical loads of 25k, 50k, 50k 50k, and 25 k at nodes 4, 6, 8, 7, and 5 respectively.

Results with pinned supports:

Node	Moment(kip-inches)
0	0.00
1	0.00
2	-7850.44
3	-7850.44
4	-14192.97
5	-14192.97
6	+2700.32
7	+2700.32
8	+2674.87

Here the minus sign denotes tension on the outer fiber while the plus sign denotes tension on the inner fiber.

Beam	Axial (kip)
0	107.07
2	107.07
4	107.07
6	107.07
8	111.00
10	111.00
12	98.39
14	98.39

All axial loads are compressive.

Component	Pinned	One Horiz. Restraint
Ridge Vert.Δ	1.428″	22.975″
Ridge Horiz. Δ	0.000″	8.362″
Max. rafter P	110.00 kip	19.13 kip
Eave moment	-14192.97 k-″	+1265.59 k-″
Ridge moment	+2674.87 k-″	+35641.37 k-″

To maintain pinned operation:

- **rafter axial load must be accommodated**
- **anchor bolts resist significant shear**
- **foundation has minimal lateral and torsional displacements**

4.2 TAPERED CONDUITS (MONOPOLES)

Tapered conduits (hollow circular or near-circular cross-section) are used in flagpoles, antenna towers, and off-highway signs. As an illustration consider flagpoles designed to "Guide Specifications for Design of Metal Flagpoles", FP1001-07, shown as Reference 3.

The principal load is wind , which varies with geographical location and height of flag and pole above ground. The wind loads on flags varying in dimensions from 5′ x 8′ to 20′ x 30′ were tested by the NAAMM in 1984, at air speeds from 60 mph to 110 mph. These tests provide for the empirical formulas below for flag loads. This range roughly correspond to the military storm, post, and garrison flags with dimensions 5′ x 9.5′, 8′-11 3/8″ x 17′ and 20′ x 38′, respectively.

Basic wind speed is determined from the wind speed maps in Reference 3. The terms used in the basic design formulas are:

A	=	pole segment projected area, ft^2
Af	=	flag area, ft^2
Cd	=	drag coefficient, dimensionless
Ch	=	height coefficient, dimensionless
d	=	pole diameter, ft
G	=	gust factor, dimensionless, 1.14 minimum
P	=	wind pressure on pole, lbf/ft^2
V	=	wind speed, 3 second gust, miles/hour
Wf	=	flag load, at top of attachment, lbf
Wp	=	flag load on pole segment, lbf
z	=	height above grade, feet

The drag and height coefficients depend upon Wp*d and height:

Wp times d	Cd
<= 39	1.10
39<=V*d< 78	129/(V*d)^(1/3)
< 78	0.45

height above grade	Ch
<= 16.4	0.86
16.4 < z <= 900	2.01*(z/900)^(2/9.5)

Basic wind pressure on a pole segment is:

P = 0.00256*V^2*Ch*G

The load on a pole segment is:

Wp = P*A*d

And the load due to the flag is:

Wf = 0.0010*V^2*Ch*G*Af^(1/2) for nylon and cotton flags
Wf = 0.0014*V^2*Ch*G*Af^(1/2) for polyester flags

Example 4.2

As an example consider a flag and monopole with the following properties:

Height above ground	=	80ft
Flag	=	post (8′-11 3/8″ x 17′), cotton
Wind (3 second)	=	110 mph
Base diameter	=	20″
Tip diameter	=	8″
Taper	=	linear
Thickness	=	0.5″
Material	=	steel

Solution

Divide the pole into four segments, each 20′ long, with diagrams shown on the next page.

First find weights (vertical load) at each node.
Weight = density*Π*t*(d0*L+α*L^2/2) = density*volume
α = d1-d0/L, the constant taper
density = .283 lbf/in.^3
In general, d0 = d(outside) -t

Node	d0(in.)	L(in.)	weight(lbf)
0	19.5	120	1014
1	16.5	240	1627
2	13.5	240	1307
3	10.5	240	987
4	7.5	120	373

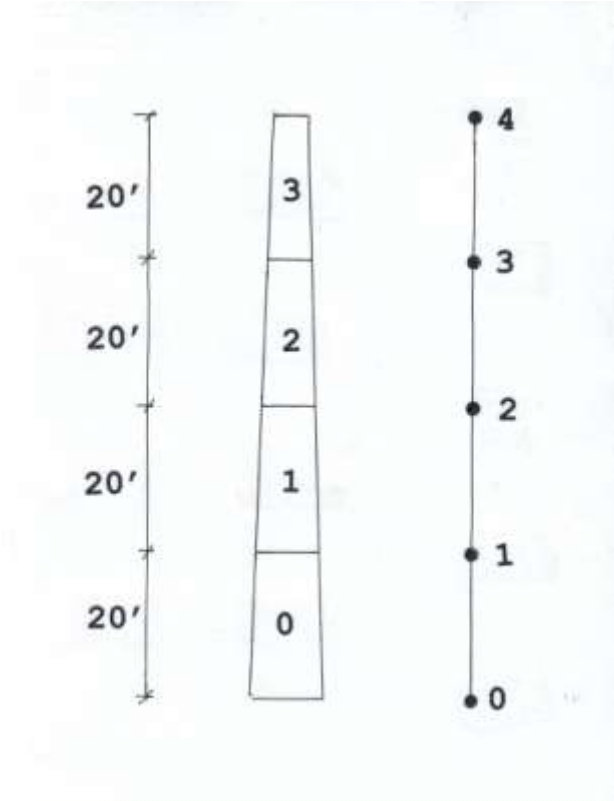
Let G = 1.14, and conservatively let Cd = 1.00

Next find Ch, P, and Wp, noting A = d(outside)*L

Node	Ch	P(psf)	A(ft^2)	Wp(lbf)
1	0.902	31.85	28.333	902
2	1.044	36.87	23.333	860
3	1.137	41.32	18.333	736
4	1.208	42.66	6.667	284

Wf = 0.0019*100^2*1.208*1.14*sqrt(152.115)
Wf = 206 lbf

Total horizontal load on node 4 = 284+206 = 490 lbf



Program results for the flagpole give:
Base moment = 1630 kip-inches
Tip deflection = 15.26 inches

5. 2nd ORDER TAPERED BEAM STIFFNESS MATRICES

The second order stiffness matrix provides the inclusion of P-δ effects by modification of the stiffness matrix and the P-Δ effects are gotten by revising the coordinates used for calculations of the beam length and node coordinates. The method used here follows References 4. And 5.

The taper of the beam has a complicating effect on the stiffness matrix derivation. This effect allows no closed form representation of the displacement function, in contrast to that of a prismatic beam, as shown in Reference 6. Solution of the governing equation of equilibrium is done by substituting a power series for the unknown transverse displacements which are then differentiated twice to obtain the moments along the beam.

Section 5.1 below discusses the generation of the required power series, followed by Section 5.2 which generates the stiffness matrices for tapered I-beams and conduits, and Section 6 describing the assembly program.

5.1 Power Series Representation of a Function

A power series is the infinite sum of ascending powers of the independent variable as

$$y = a0 + a1*x^1 + a2*x^2 + .. = \sum_{i=0}^{\infty} ai*x^i$$

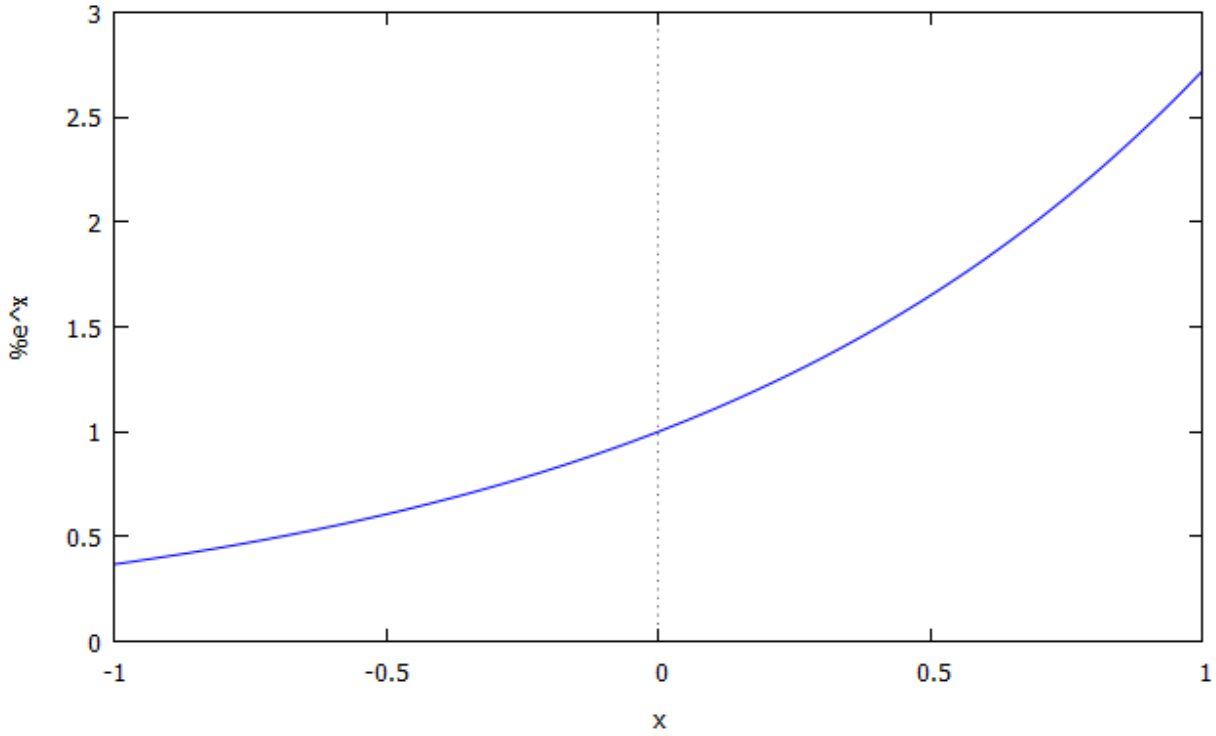
In practice, only the first N+1 terms are used, where N is large enough to obtain sufficient accuracy.

$$y = \sum_{i=0}^{N} ai*x^i$$

In the calculations which follow, power series are used both to represent unknown functions (displacements) and known functions (moments of inertia). The choice of the form of the series is key to maintaining good accuracy.

A common series used in calculations is the Maclaurin Series, shown here for the function f = e^x, where e is the base of natural logarithms.

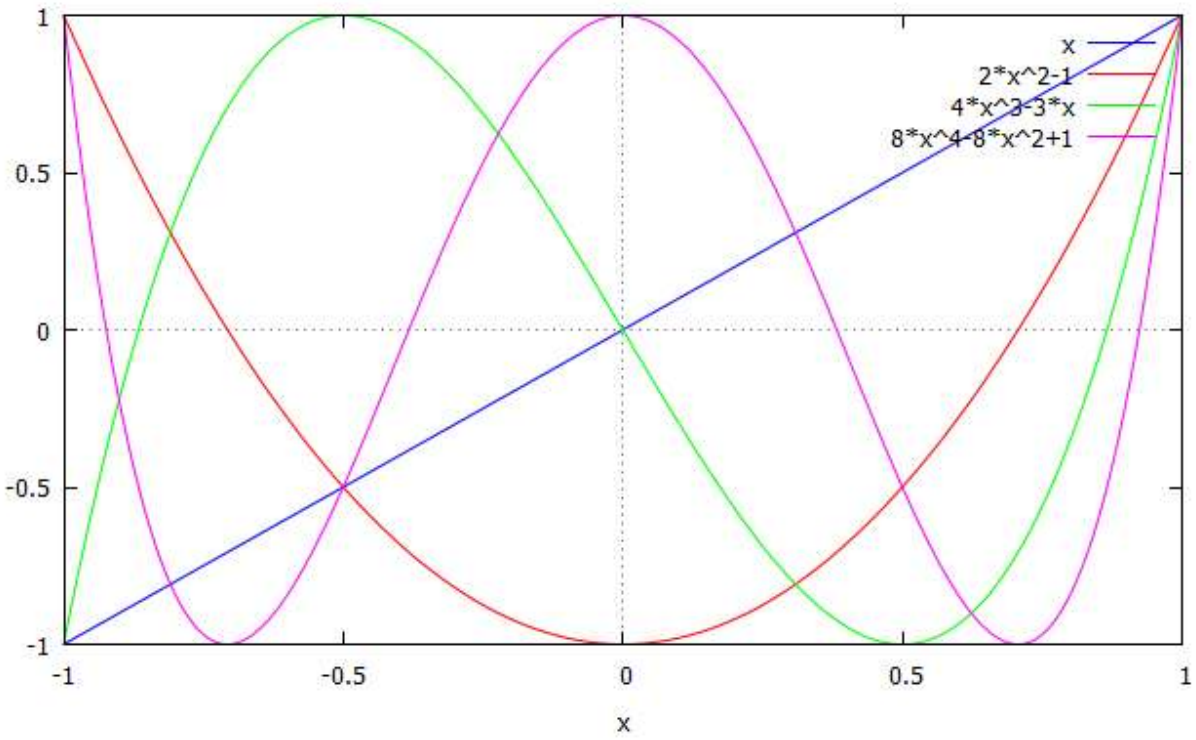
e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! +..
as graphed.



This series gives good results in the immediate vicinity of x = 0, but diverges away from the correct answer away from this vicinity. Each problem has an "immediate vicinity" which must be found to give credence to the answer.
A superior method of series representation is the "Chebyshev Series", named after Pafnuty Chebyshev, a famous Russian mathematician. The terms of this series are represented by the following expression:

Tn(x) = cos(n*arcos(x)) where
Tn(x) = nth term of series and -1<= x <= +1

This interval is very convenient as any range of independent variable may be examined by using the normalizing transformation ζ = new variable = x/xmax, xmax being the maximum absolute value of x.
The first five terms of the series are:
T0(x) = 1
T1(x) = x
T2(x) = 2*x^2-1
T3(x) = 4*x^3-3*x
T4(X) = 8*x^4-8^x^2+1



The chart above shows T1(x) through T4(x). The roots of these functions, namely the values of x where Tn(x) crosses through zero, are given by xk:

$$xk = \cos(\Pi(2*k-1)/2*n),\ k = 1,2,3,..,n$$

The functions for Tn(x) and xk are used to finds the Chebyshev series of a function by the relationship shown below, where the function f(x) is known at the eros of Tn(x).

$$f(x) = \sum_{i=0}^{N} ci*Ti(x)$$

$$c0 = (1/N)* \sum_{k=1}^{N} f(xrk) \quad \text{and} \quad ci = \sum_{k=1}^{N} f(xrk)*Ti(xrk)$$

where xrk = zeros of Tn(x)

The strategy to find the power series of a known function :

(1) Find roots of Tn(x) by the equation for xk.

(2) Find the N values of the given function at the roots of Tn(x), i.e., the f(xrk).

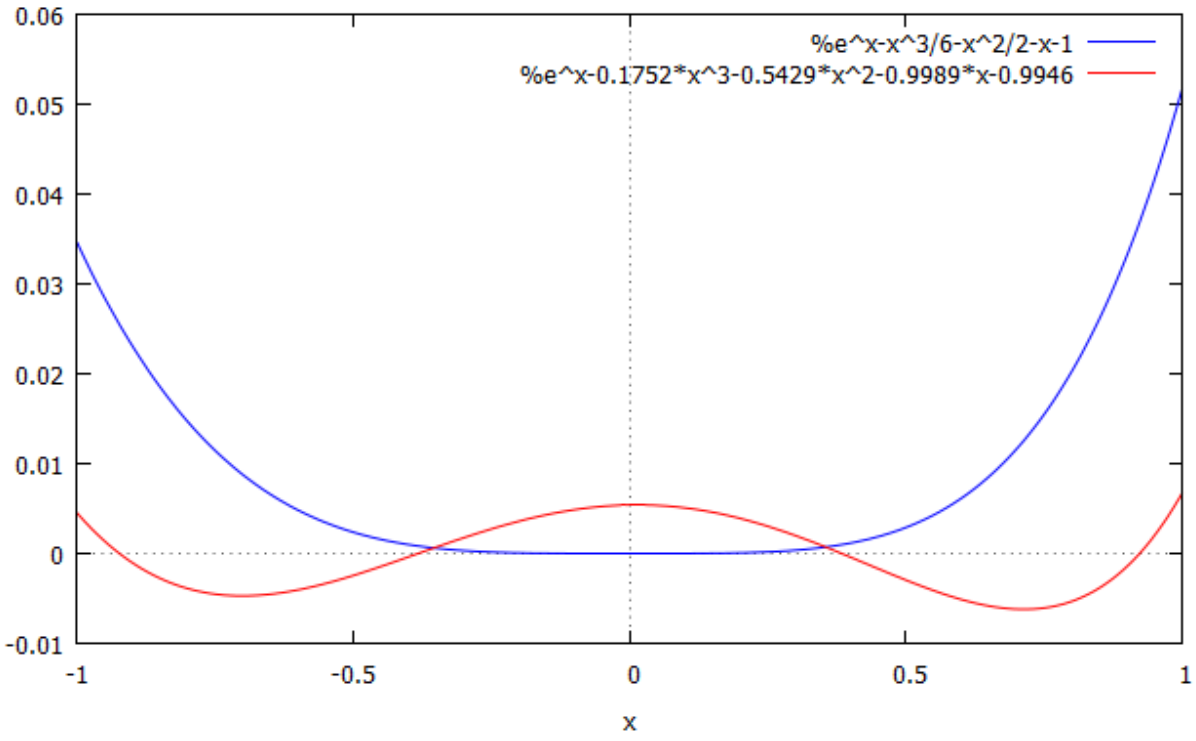
(3) Calculate c0, which involves only the given function

(4) Calculate the Ti(xrk), i.e., the values of Ti at the roots of Tn(x).
(5) Find ci as above.
(6) f(x) may now be represented as a sum of Chebyshev functions.
(7) Finally convert the series in (6) to a power series, using the polynomial coefficients of the Chebyshev terms T0, T1, T2, ..Tn.

A comparison of the errors for four terms of the Macluarin power series for e^x with four terms of the Chebyshev power series, the shown as:

Maclaurin = 1.0000 + 1.0000x + 0.5000x^2 + .1667x^3
Chebyshev = 0.9946 + 0.9989x + 0.5429x^2 + .1752x^3

The series' coefficients are quite close, but not the errors shown plotted below. The Maclaurin series (blue) is quite accurate in the immediate neighborhood of the origin while the error in the Chebyshev series (red)is more or less evenly spread out over the entire normalized variable range, and is zero at the xk values.



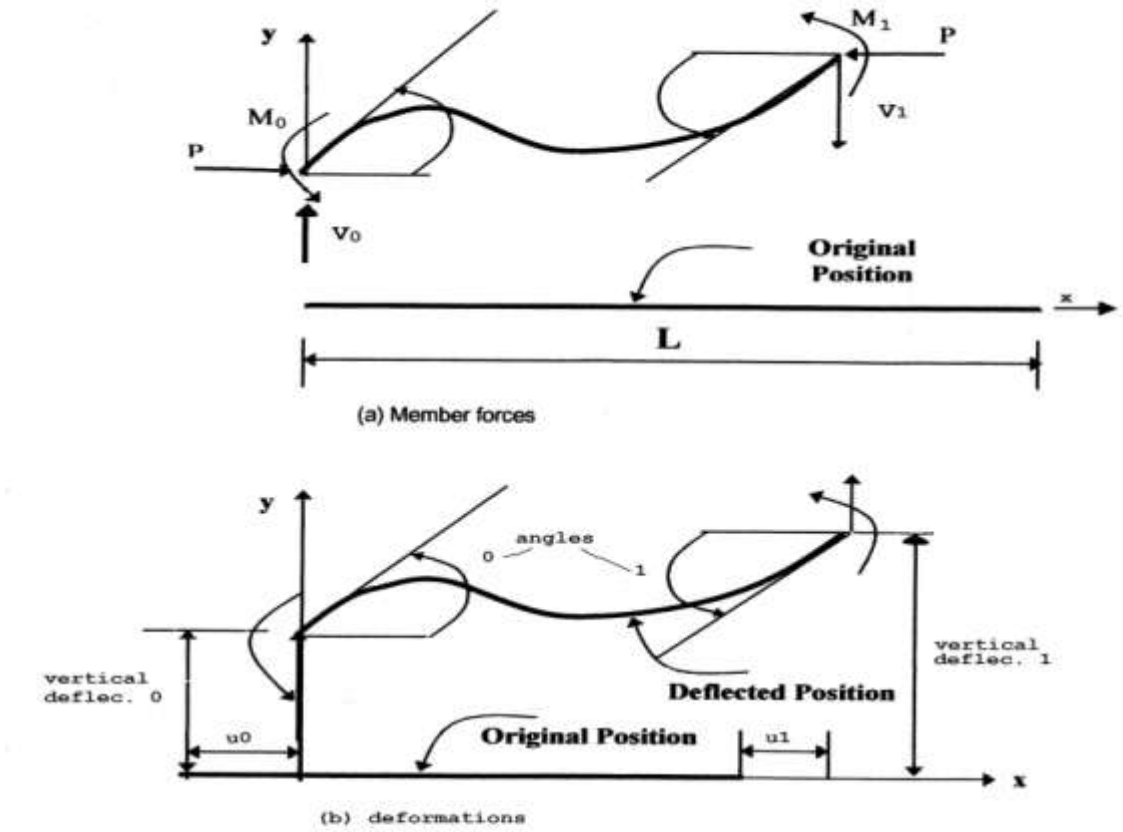
5.2 Series Solution for Stiffness Matrix

Reference 4 and 5 solve the problem of the second order Linear elastic beam-column as a series expansion of transverse displacement as :

$$y(\zeta) = \sum_{i=0}^{\infty} a_i * \zeta^i \quad \text{where } \zeta = x/L, \text{ the normalized distance along the beam}$$

This is called a power series, as each term is multiplied by a power of the independent variable

A diagram of the beam forces and displacements is:



Nonprismatic beam-column member neglecting the presence of semirigid joint Connections.

The approach used here is outlined as:

(1) Find expression for displacement in terms of moment of inertia (expressed as a power series) and end conditions.

(2) Solve (1) for displacement coefficients.

(3) Find M0 and M1 from m = -E*I*y''(ζ)/L^2 where y'' = second derivative of displacement with respect to ζ.

(4) Find V0 and V1 from v = -dm/dζ

(5) Using (3) and (4), which give the forces in terms of the end displacements, the stiffness matrix is formed.

This analysis includes both P-Δ and P-δ effects, when used with a modified assembly matrix, as shown in Section 6 below.

There is no restriction on the form of taper with this method, the only requirement being that the power series for I be defined at the roots of the Chebyshev equations, either by equation or numerical values. Thus this is a very general approach.

(1) FIND EXPRESSION FOR DISPLACEMENT

$$y(\zeta) = \sum_{i=0}^{\infty} a_i \zeta^i$$

This is a power series in displacement as a function of the normalized variable, ζ = x/L, which varies from -1 to +1. In practice, the series is terminated at a finite value, large enough to provide satisfactory results. Much of the effort in finding the stiffness matrix is done to find the ai here.

$$I(\zeta) = \sum_{j=0}^{\infty} I_j \zeta^j$$

This is a second power series, which is known as Ij versus ζ is known. It is first calculated in terms of Chebyshev polynomials and then transformed to a power series, as described in Section 5.1.

It has been found that the number of power terms required for I(ζ) is relatively small, say 5 or 6, but the number of terms for 0.1% accuracy of y

is quite large, say 60 or 70. In the programs developed, 11 terms where used for I and 101 ai terms for y.

(2) FIND THE DISPLACEMENT COEFFICIENTS

Four conditions relate the ai coefficients to the end conditions of the beam (called 0 and 1):

δ0 = vertical deflection at end 0 = a0

dy/dζ at ζ=0 = θ0 = gives a1 = L*θ0

$$\delta 1 = \text{vertical deflection at end 1} = \sum_{i=0}^{\infty} ai$$

$$\frac{dy}{dx} \text{ at } \zeta=1 = \theta 1 = \text{gives } L*\theta 1 = \sum_{i=0}^{\infty} (i+1)*(ai+1)$$

These equations cannot be solved explicitly as the ai coefficients are functions of the actual end conditions.

A fifth condition relating the coefficients is:

$$a(i+4) = -\frac{1}{I0*(i+4)*(i+3)} * F$$

$$F = \left[\sum_{j=1}^{i+2} Ij*(i+4-j)*(i+3-j*a(i+4-j)) \right] - P*L^2*(ai+2)/E$$

$$\text{where } I(\zeta) = \sum_{i=0}^{\infty} Ii*\zeta^i$$

The summations in practice are only carried out To the Nth term.

This last condition shows that we may solve for ai coefficients if we can represent the Ii as a power series, which we can do by Section 5.1 above.
Note that this condition is a recursive relation, that is, we solve for each term in terms of the preceding (lower number) terms.

Expanding the first series,

$$a0 + a1 + a2 + a3 + \sum_{n=4}^{\infty} an = \delta 1$$

$$a2 + a3 + \sum_{n=4}^{\infty} an = \delta 1 - \delta 0 - L*\theta 0 = Q1$$

Expanding the second series,

$$1*a1 + 2*a2 + 3*a3 + \sum_{n=4}^{\infty} n*an = L*\theta 1$$

$$2*a2 + 3*a3 + \sum_{N=4}^{\infty} n*an = L*(\theta 1 - \theta 0) = Q2$$

Because of the recursive nature of the coefficients an, the series are functions of a2 and a3 alone.

Now breakdown the coefficients as a2(i)*a2 and a3(i)*a3, where a2(i) and a3(i) are functions of geometry only and a2 and a3 are functions of the end conditions only.

Noting that a2 and a3 may be factored out of the series, the equations become :

$$a2 + a3 + a2* \sum_{n=4}^{\infty} a2(n) + a3* \sum_{n=4}^{\infty} a3(n) = Q1$$

$$2*a2 + 3*a3 + a2* \sum_{n=4}^{\infty} n*a2(n) + a3* \sum_{n=4}^{\infty} n*a3(n) = Q2$$

The series immediately above are independent of end conditions, but are functions of moment of inertia versus ζ. Call these series sum1, sum2, sum3, and sum4, respectively.
The two equations are now:

$$(1 + sum1)*a2 + (1 + sum2)*a3 = Q1$$
$$(1 + sum3)*a2 + (1 + sum4)*a3 = Q2$$

Solve these equations for a2 and a3

a2 = [+(3+sum4)*Q1-(1+sum2)*Q2]/den
a3 = [-(2+sum3)*Q1+(1+sum1)*Q2]/den
and den = (1+sum1)*(3+sum4)-(1+sum2)*(2+sum3)

Now we have a2 and a3 as functions of both geometry (the four sums) and end conditions (Q1 and Q2). It should be noted that the sums may now be calculated using the recursive property as above.

(3) SOLVE FOR M0 AND M1, THE END MOMENTS

$$y(\zeta) = a0 + a1*\zeta^1 + a2*\zeta^2 + a3*\zeta^3 + \sum_{n=4}^{\infty} an*\zeta^n$$

Now taking the second derivative we have

$$y'' = 2*a2 + 6*a3*\zeta^1 + a2*\sum_{n=4}^{\infty} n*(n-1)*a2(n)\zeta^{(n-2)} + a3*\sum_{N=4}^{\infty} n*(n-1)*a3(n)\zeta^{n-2}$$

y'' = 2*a2 + 6*a3*ζ^1 + a2*sum5 + a3*sum6

When ζ = 0, y'' = 2*a2
When ζ = 1, y'' = a2*(2 + sum5) + a3*(6 + sum6)

$$sum5 = \sum_{n=4}^{\infty} n*(n-1)*a2(n) \qquad sum6 = \sum_{n=6}^{\infty} n*(n-1)*a3(n)$$

Again noting sum5 and sum6 are finctions of geometry only.

In general, m = -E*I(ζ)*y''(ζ)/L^2

M0 = -E*I00*2*a2/L^2 = -2*E*I00(f1*Q1+f2*Q2/L^2
M0 = g1*Q1 + g2*Q2
Here I00 is the moment of inertia,in terms of ζ at the near node, not the first moment of inertia in the power series for I(ζ).

M1 = -E*I11*(2*a2+6*a3+a2*sum5+a3*sum6)/L^2

This is expanded to :
M1 = (-E*I11/L^2)*(f1*(2+sum5)+f3*(6+sum6))+
(-E*I11/L^2)*(f2*(2+sum5)+f4*(6+sum6))
M1 = g3*Q1 + g4*Q2

© Marvin Liebler

(4) SOLVE FOR V0 and V1, THE END SHEARS

v = shear = -dm/dζ
m = -E*I(ζ)*y''(ζ)/L^2

Using the product rule for derivatives,

v = [E*I(ζ)*y'''(ζ) + E*y''(ζ)*dI(ζ)/dζ]/L^2

$$y'''(\zeta) = 6*a3 + a2*\sum_{n=4}^{\infty} n(n-1*(n-2)*a2(n)*\zeta^{n-3}$$

$$+ a3*\sum_{n=4}^{\infty} n(n-1)*(n-2)*a3(n)*\zeta^{n-3}$$

y'''(ζ) = 6*a3 + a2*sum5 + a3*sum6

Now the shear will be constant at the value at the first node, because there are no loads within the element, thus V1 = -V0

V0 = E*[I00*6*a + 2*a2*dI(ζ)/dζ]/L^3
where dI/dζ is evaluated at ζ = 0.

The terms multiplying a3 and a2 are dependent upon the particular beam element.

The terms can be combined to give :

V0 = h1*Q1 + h2*Q2
V1 = - V0

(5) POPULATE THE STIFFNESS MATRIX

Collecting results,

V0 = +h1*Q! + h2*Q2
M0 = +g1*Q1 + g2*Q2
V1 = -h1*Q1 – h2*Q2
M1 = +g3*Q1 + g4*Q2
Q1 = δ0–δ1–L*θ0
Q2 = L*(θ1-θ0)

Consider the composition of V0,

V0 = h1*(δ1-δ0-L*θ0) +h2*L*(θ1-θ0)
V0 = -h1*δ0 –L*(h1+h2)*θ0 +h1*δ2 +h2*L*θ1

Therefore K22 = -h1
K23 = -L*(h1+h2)
K25 = +h1
K26 = +h2*L

V1 = -V0 → K52 = +h1
K53 = +L*(h1+h2)
K55 = -h1
K56 = -h2*L

By similar reasoning,

K32 = -g1
K33 = -L*(g1+g2)
K35 = +g1
K36 = +g2*L

K62 = -g3
K63 = -L*(g3+g4)
K65 = +g3
K66 = +g4*L

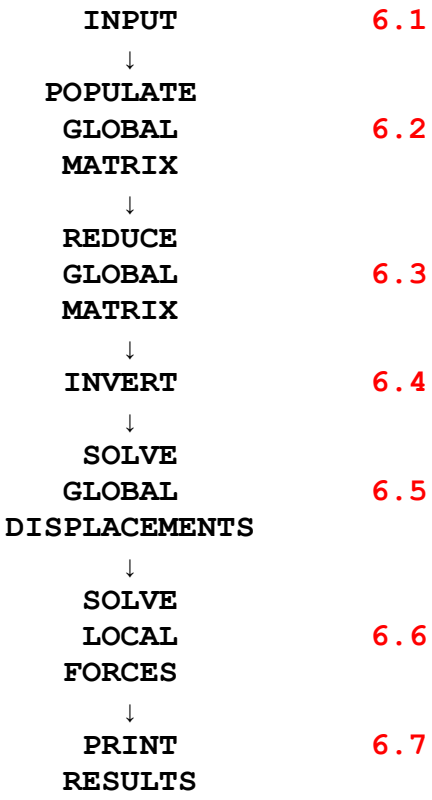
From the previous 1st order analysis, the axial Terms are given by K11 = K44 = -K14 = -K41 =

$$\int_0^L \frac{dx}{E*A(x)}$$

The program for the I beam is ktprir1.h, shown as Appendix 2, for the interested reader.

6. 2nd ORDER TAPERED BEAM ASSEMBLIES

The second order assembly program is run for a prescribed number of sets, required to approach the final result. The block diagram from section 3 is repeated, together with changes from the 1st order method. This new program is non1r1.c for the I-beam.



1. **Input – the number of sets is added.**

2. **Populate global matrix using terms derived in section 5.3 above.**

3, 4, 5 – **Same as first order.**

6. **After solving for local forces, form axial load vector and return to step 2. with displacements (used for calculating updated lengths) and load vector until sets complete.**

7. **Print results, including change in displacements for each calculation set to ensure minimum change in last two displacements .**

© Marvin Liebler

7. EXAMPLES

7.1 CHECK WITH INDEPENDENT SOURCE

Reference 7, AISC Design Guide 25, "Frame Design Using Web-tapered Members", has an example of a cantilever beam with the following properties:

bf = 6 in.
d0 = 24.5 in.
d1 = 9.5 in.
E = 29000 ksi
lateral load at free end = 10 kip
Pcr = elastic flexural buckling load in actual configuration kip
αPcr = axial load, varies, kip
Pr = required load, kip
α = 1.0 for LRFD, 1.6 for ASD

αPr/Pcr	AISC		non1r1.c	
	αPr	Δ(in.) M(k")	Δ(in.)	M(k")
0	0	2.23 1963.2	2.23	1965.1
.1	64.9	2.46 2124.0	2.49	2122.8
.2	129.8	2.77 2320.8	2.77	2321.2
.3	194.7	3.16 2575.2	3.16	2575.0
.4	259.6	3.67 2912.4	3.67	2911.4

$$\text{Let difference} = \frac{\text{non1r1.c value} - \text{AISC value}}{\text{AISC value}}$$

αPr/Pcr	Δ diff.	M diff.
0	0	-0.005%
.1	+0.40%	-0.056%
.2	0	+0.017%
.3	0	-0.008%
.4	0	-0.034%

7.2 FRAME EXAMPLE

Consider first the structure analyzed in section 4.1 with both first and second order analysis. The results are :

Quantity	1st order	2nd order	difference
Ridge deflection	1.428″	1.429″	+0.07%
Eave deflection	0.287″	0.288″	+0.35%
Ridge moment	2675 k″	2638 k″	-1.30%
Eave moment	14173 k″	14219 k″	+).18%

The flexural stresses in these members is quite low. Try reducing the member sizes based on the moments above :

Node	x(in.)	y(in.)	Beams	d1(in.)	d2(in.)
0	-714.500	0.000	0,1	9.000	21.000
1	+714.500	0.000	2,3	21.000	31.248
2	-708.500	90.000	4,5	32.584	24.000
3	+708.500	90.000	6,7	24.000	15.000
4	-703.376	166.864			
5	+703.376	166.864	all bf = 6 in.		
6	-360.000	257.000	all tf = 1 in.		
7	+360.000	257.000	att tw = ½ in.		
8	0.000	351.500			

The results now are :

Quantity	1st order	2nd order	difference
Ridge deflection	9.829″	9.855″	+0.267%
Eave deflection	2.496″	2.502″	+0.24%
Ridge moment	2422 k″	2661 k″	+9.87%
Eave moment	14759 k″	14921 k″	+1.10%

It is seen that the ridge moment shown by second order analysis increases significantly.

© Marvin Liebler

7.3 MONOPOLE EXAMPLE

Use the same example as in section 4.2. The results are :

Quantity 0	1st Order	2nd Order	Difference
Base moment	1630 k″	1639 k″	+0.55%
Tip deflection	15.26 ″	15.34 ″	+0.52%

8. CONCLUDING REMARKS

(1) Although not required by Reference 3, second order analysis is recommended for flagpoles, especially those with increased vertical weight.
(2) Semi-rigid connections may be added similar to those methods in Reference 6.
(3) Second order analysis is the core of the direct method.
(4) If, in lieu of the second order analysis developed here, recourse is made to dividing the tapered member into a series of prismatic members, using Reference 6 techniques, it is a good idea to check this against the ASIC reference cited above.

REFERENCES

1. "Stiffness Matrix for 2D tapered beams", L.L. Yaw, 2009, www.people.wallawalla.edu/~louie.yaw

2. "Rigid-frame-erection-manual", Worldwide Steel Buildings, www.worldwidesteelbuildings.com

3. "Specifications for Design of Metal Flagpoles", FP1001-07, National Association of Architectural Metal Manufacturers, www.acmelingo.com/flagpoles/FP1001-07.pdf

4. "Exact Secant Stiffness Matrix for Nonprismatic Beam-Columns with Elastic Semirigid Joint Connections", S.Z. Al-Sadder and H.Y. Qasrawi, Emirates Journal for Engineering Research, 9 (2),127-135, (2004), www.google.com

5. "A tapered Timoshenko-Euler beam element for analysis of steel portal frames", G.-Q. Li and J.J. Li, Journal of Constructional Steel Research 58 (2002), 1531-1544, www.elsevier.com/locate/jcsr.

6. "The Direct Method in Steel", Course 274, www.pdhonline.org

7. "Frame Design Using Web-tapered Members", AISC Design Guide 25, www.aisc.org

8. "Mathematical Handbook", M.R. Spiegel, Schaum's Outline, McGraw-Hill, 1968 www.barnesandnoble.com

APPENDIX 1

Consider first the doubly-symmetric I-beam with a linear web taper.

$d = \text{height of web} = d1 + \alpha * x$ where

$\alpha = \frac{d2-d1}{L} = \text{taper and } \tan^{-1}(\alpha) = \text{degree of taper}$

$A(x) = 2*bf*tf + d*tw = 2*bf*tf + tw*(d + \alpha*x)$

$I(x) = tw*\frac{d^3}{12} + 2*bf*tf*(\frac{d+tf}{2})^2 + \frac{2*bf*tf^3}{12}$

Evaluating f11 for the tapered beam,

$$f11 = \frac{1}{E}\int_0^L \frac{1}{2*bf*tf+tw*d1+\alpha*x}\,dx = \frac{1}{E}\int_0^L \frac{1}{\alpha*x+b}\,dx$$

where $b = 2*bf*tf+tw*d1$

Note that α can be + for d2>d1 and - for d2<d1.

Evaluating the integral from reference 8,

$$f11 = \frac{1}{\alpha*E}\ln\left(\frac{\alpha*L+b}{\alpha}\right) \text{ and } = \frac{L}{E*A} \text{ for prismatic beam}$$

From the discussion in Section 2. above, it is seen there are three types of integrals to be evaluated for the remaining coefficients, as:

$$\frac{1}{E}\int_0^L \frac{x^2}{I(x)}\,dx, \quad \text{const.}\int_0^L \frac{x}{I(x)}\,dx, \text{ and } \text{const.}\int_0^L \frac{1}{I(x)}\,dx$$

The equation above for I(x) may be expressed as :

$I(x) = ga*x^3 + gb*x^2 + gc*x + gb$, where :

$ga = tw*c1^3$

$gb = (d1*tw/4 + bf*tf/2)*c1^2$

$gc = (d1^2*tw/4 + bf*(d1+tf)*tf)*c1$

$gd = bf*tf*(d1+tf)^2/2 + bf*tf^3/6 + tw*d1^3/12$

$c1 = \alpha = \frac{d2-d1}{L}$

This can be changed to give a unit multiplier of the cubic term, I(x) = ga*(x^3 + b*x^2 + c*x + d), where:
b = gb/ga
c = gc/ga
d = gd/ga

Integrals of the type of cubic above in the denominator cannot be found by the author, but denominators of the type ga*x*(a1*x^2+b1*x+c)and numerators of the type x^2, x, and 1 can yield closed-form integrals, as shown in reference 8.

We require a new variable, x1, such that
x^3 + b*x^2 + c*x + d = x1*(a1*x1^2 + b1*x1 +c1)

Each cubic equation must have either three real roots or one real and two complex roots. All I(x) examined had one real and two complex roots.

Solving the above equation for a1, b1, and c1 :
a1 = 1
b1 = 2*z1 – 2*z2real
c1 = (z1-z2real)^2 + z2imag
x1 = x – z1, x = x1 + z1, and dx = dx1

where the roots are z1, z2real ± i*z2imag.
Note that these scalar values may be plus or minus.

The above integrals are then converted, using the values for a1, b1, and c1, and executed by integral forms 14.265, 14.266, and 14.269 in Reference 8.

The fij for the conduit section are found using the same integrals as above.

© Marvin Liebler

APPENDIX 2

```
/*********************************************************************
 *                                                                   *
 * Ktprir1.h    : First find T0(x) thru T10(x) for I-beam (steps 1-4) *
 *                Then first equivalent power series (step 5)        *
 *                 Step 6 = find stiffness matrix of tapered beam-    *
 *                 column : 10 = largest non-zero label i of a2,a3 :  *
 *                 06-130-14 : ml                                     *
 *                                                                   *
 *********************************************************************/

void findKBlocal(int beam_no,double *bcoor_mtrx,int *beamconn_mtrx,
               double *beamprop_mtrx,double K[6][6],double *axial)
{
      int beamprop_no,i,j,k,n,node0,node1;
      double c0x,c0y,c1x,c1y,L,P;
      double bf,d0,d1,E,tf,tw;               /* inputs */
      double I00,I11,jj,kk,nn;
      double f1,f2,f3,f4,g1,g2,g3,g4,h1,h2,z1,z2;
      double sum1,sum2,sum3,sum4,sum5,sum6;
      double a,aa2,aa3,ax,b,den;
      double d,Iz,z;
      double pi       =       3.141592653589793;
      double *a2,*a3,*alpha,*cj,*I,*xk;
      void getcoefficients(double *,double *);

       P              =       *(axial+beam_no);

       node0          =       *(beamconn_mtrx+3*beam_no+0);
        node1         =       *(beamconn_mtrx+3*beam_no+1);
       beamprop_no    =       *(beamconn_mtrx+3*beam_no+2);

       c0x            =       *(bcoor_mtrx+2*node0+0);
       c0y            =       *(bcoor_mtrx+2*node0+1);
       c1x            =       *(bcoor_mtrx+2*node1+0);
       c1y            =       *(bcoor_mtrx+2*node1+1);

       bf             =       *(beamprop_mtrx+6*beamprop_no+0);
       d0             =       *(beamprop_mtrx+6*beamprop_no+1);
       d1             =       *(beamprop_mtrx+6*beamprop_no+2);
       E              =       *(beamprop_mtrx+6*beamprop_no+3);
       tf             =       *(beamprop_mtrx+6*beamprop_no+4);
       tw             =       *(beamprop_mtrx+6*beamprop_no+5);

       L              =       sqrt((c1x-c0x)*(c1x-c0x)+(c1y-c0y)*(c1y-c0y));

       a2             =       calloc(101,sizeof(double));
       a3             =       calloc(101,sizeof(double));
       alpha          =       calloc(11,sizeof(double));
       cj             =       calloc(11,sizeof(double));
       I              =       calloc(101,sizeof(double));
       xk             =       calloc(11,sizeof(double));

        for(k=0;k<=10;k++)
        {
                *(alpha+k)      =       0.0;
                *(cj+k)         =       0.0;
                *(xk+k)         =       0.0;
        }
       for(k=0;k<=100;k++)
       {
              *(a2+k)         =       0.0;
              *(a3+k)         =       0.0;
              *(I+k)          =       0.0;
       }
```

```
                /***********
                 *         *
                 * STEP 1. *
                 *         *
                 ***********/

for(k=1;k<=10;k++)
{
        kk      =       k;
        *(xk+k) =       cos(pi*(2.0*kk-1.0)/(2.0*10.0));
}


                /***********
                 *         *
                 * STEP 2. *
                 *         *
                 ***********/


for(k=1;k<=10;k++)
{
        z               =       *(xk+k);
       d                =       d0+(d1-d0)*z;
       Iz               =       tw*d*d*d/12.0+
                                  (d+tf)*(d+tf)*bf*tf/2.0+
                                  bf*tf*tf*tf/6.0;
        *(alpha+k)      =       Iz;
}

                /***********
                 *         *
                 * STEP 3. *
                 *         *
                 ***********/

for(k=1;k<=10;k++)
{
        *(cj+0)         +=      *(alpha+k);
}
*(cj+0)         *=      1.0/10.0;


                /***********
                 *         *
                 * STEP 4. *
                 *         *
                 ***********/

for(j=1;j<=10;j++)
{
        *(cj+j)          =       0.0;
        jj               =       j;
        for(k=1;k<=10;k++)
        {
                z        =      *(xk+k);
                *(cj+j)  +=     *(alpha+k)*cos(jj*acos(z));
        }
        *(cj+j)          *=       2.0/10.0;
}


                /***********
                 *         *
                 * STEP 5. *
                 *         *
                 ***********/
```

```
getcoefficients(cj,I);


                /**********
                *        *
                * STEP 6 *
                *        *
                **********/

I00     =       tw*d0*d0*d0/12.0+(d0+tf)*(d0+tf)*bf*tf/2.0+
                bf*tf*tf*tf/6.0;
I11     =       tw*d1*d1*d1/12.0+(d1+tf)*(d1+tf)*bf*tf/2.0+
                bf*tf*tf*tf/6.0;

*(a2+2)         =       1.0;
*(a3+3)         =       1.0;

for(n=4;n<=100;n++)
{
        nn              =       n;
        for(j=1;j<=n-2;j++)
        {
                jj      =       j;
                *(a2+n) +=      *(I+j)*(nn-jj)*(nn-jj-1.0)
                                *(*(a2+n-j));
        }
        *(a2+n)         +=      P*L*L*(*(a2+n-2))/E;
        *(a2+n)         *=      -1.0/((nn*(nn-1.0))*(*(I+0)));
}

for(n=4;n<=100;n++)
{
        nn              =       n;
        for(j=1;j<=n-2;j++)
        {
                jj      =       j;
                *(a3+n) +=      *(I+j)*(nn-jj)*(nn-jj-1.0)
                                *(*(a3+n-j));
        }
        *(a3+n)         +=      P*L*L*(*(a3+n-2))/E;
        *(a3+n)         *=      -1.0/((nn*(nn-1.0))*(*(I+0)));
}

sum1                    =       0.0;
sum2                    =       0.0;
for(n=4;n<=100;n++)
{
        sum1            +=      *(a2+n);
        sum2            +=      *(a3+n);
}
sum1                    +=      1.0;
sum2                    +=      1.0;

sum3                    =       0.0;
sum4                    =       0.0;
for(n=2;n<=100;n++)
{
        nn              =       n;
        sum3            +=      *(a2+n)*nn;
        sum4            +=      *(a3+n)*nn;
}

sum5    =       0.0;
sum6    =       0.0;
for(n=4;n<=100;n++)
{
        nn              =       n;
        sum5            +=      *(a2+n)*nn*(nn-1.0);
        sum6            +=      *(a3+n)*nn*(nn-1.0);
}
sum5                    +=      2.0;
```

```
        sum6                    +=      6.0;

        den                      =      +sum1*sum4-sum2*sum3;
        f1                       =      +sum4/den;
        f2                       =      -sum2/den;
        f3                       =      -sum3/den;
        f4                       =      +sum1/den;

        g1                       =      -2.0*E*I00*f1/(L*L);
        g2                       =      -2.0*E*I00*f2/(L*L);
        g3                       =      -E*I11*(f1*sum5+f3*sum6)/(L*L);
        g4                       =      -E*I11*(f2*sum5+f4*sum6)/(L*L);

        z1      =       E*(2.0*(d1-d0)*(tw*d0*d0/4.0+(d0+tf)*bf*tf))/(L*L*L);
        z2      =       6.0*E*I00/(L*L*L);


        h1      =       f1*z1+f3*z2;
        h2      =       f2*z1+f4*z2;

        for(i=0;i<=5;i++)
        {
                for(j=0;j<=5;j++)
                {
                        K[i][j] =       0.0;
                }
        }

        a       =       tw*(d1-d0)/L;
        b       =       2*bf*tf+tw*d0;
        ax      =       log((a*L+b)/b);
        K[0][0] = K[3][3] = +E*a/ax;
        K[0][3] = K[3][0] = -E*a/ax;

        K[1][1]   =     -h1;
        K[1][2]   =     -(h1+h2)*L;
        K[1][4]   =     +h1;
        K[1][5]   =     +h2*L;

        K[2][1]   =     -g1;
        K[2][2]   =     -(g1+g2)*L;
        K[2][4]   =     +g1;
        K[2][5]   =     +g2*L;

        K[4][1]   =     +h1;
        K[4][2]   =     +(h1+h2)*L;
        K[4][4]   =     -h1;
        K[4][5]   =     -h2*L;

        K[5][1]   =     +g3;
        K[5][2]   =     +(g3+g4)*L;
        K[5][4]   =     -g3;
        K[5][5]   =     -g4*L;
}

void getcoefficients(double *Tc,double *cof)
{
       double   c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10;
       c0       =       *(Tc+0);
       c1       =       *(Tc+1);
       c2       =       *(Tc+2);
       c3       =       *(Tc+3);
       c4       =       *(Tc+4);
       c5       =       *(Tc+5);
       c6       =       *(Tc+6);
       c7       =       *(Tc+7);
       c8       =       *(Tc+8);
       c9       =       *(Tc+9);
       c10      =       *(Tc+10);
       *(cof+0) =       +c0-c2+c4-c6+c8-c10;
       *(cof+1) =       +c1-3.0*c3+5.0*c5-7.0*c7+9.0*c9;
```

```
        *(cof+2) =      +2.0*c2-8.0*c4+18.0*c6-32.0*c8+50.0*c10;
        *(cof+3) =      +4.0*c3-20.0*c5+56.0*c7-120.0*c9;
        *(cof+4) =      +8.0*c4-48.0*c6+160.0*c8-400.0*c10;
        *(cof+5) =      +16.0*c5-112.0*c7+432.0*c9;
        *(cof+6) =      +32.0*c6-256.0*c8+1120.0*c10;
        *(cof+7) =      +64.0*c7-576.0*c9;
        *(cof+8) =      +128.0*c8-1280.0*c10;
        *(cof+9) =      +256.0*c9;
        *(cof+10)=      +512.0*c10;
}
```