



Matrix Structural Analysis of Plane Frames using Scilab

by

Satish Annigeri Ph.D.

Professor of Civil Engineering

B.V. Bhoomaraddi College of Engineering & Technology, Hubli
satish@bvb.edu

Department of Civil Engineering
B.V. Bhoomaraddi College of Engineering & Technology, Hubli
www.bvb.edu

Table of Contents

Preface	ii
Introduction	1
Tutorial 1 – Data Organization for Matrix Analysis of Plane Frames	2
Tutorial 2 – Location Matrix for a Plane Frame	4
Tutorial 3 – Local Stiffness Matrix of a Plane Frame Element	6
Tutorial 4 – Rotation Matrix of a Plane Frame Member	8
Tutorial 5 – Global Stiffness Matrix of a Plane Frame Member	9
Tutorial 6 – Assembling the Plane Frame Structure Stiffness Matrix	10
Tutorial 7 – Assembling the Load Vector	12
Tutorial 8 – The Super Function	14
Tutorial 9 – Extracting Member End Forces	15
References	16

Preface

Scilab is a software for numerical mathematics and scientific visualization. It is capable of interactive calculations as well as automation of computations through programming. It provides all basic operations on matrices through built-in functions so that the trouble of developing and testing code for basic operations are completely avoided. Its ability to plot 2D and 3D graphs helps in visualizing the data we work with. All these make Scilab an excellent tool for teaching, especially those subjects that involve matrix operations. Further, the numerous toolboxes that are available for various specialized applications make it an important tool for research. Being compatible with Matlab®, all available Matlab® M-files can be directly used in Scilab. Scicos, a hybrid dynamic systems modeler and simulator for Scilab, simplifies simulations. The greatest features of Scilab are that it is multi-platform and is free. It is available for many operating systems including Windows, Linux and MacOS X. More information about the features of Scilab are given in the Introduction.

Scilab can help a student understand all intermediate steps in solving even complicated problems, as easily as using a calculator. In fact, it is a calculator that is capable of matrix algebra computations. Once the student is sure of having mastered the steps, they can be converted into functions and whole problems can be solved by simply calling a few functions. Scilab is an invaluable tool as solved problems need not be restricted to simple examples to suit hand calculations.

The first part of this tutorial gives a hands on introduction to a beginner and is available at <http://www.lulu.com/content/419922>. This second part continues from where the first left off and demonstrates how to use Scilab to develop a simple program for the Matrix Structural Analysis of Plane Frames using the Direct Stiffness Method (DSM). The DSM is the method used in the computer analysis of structures and is the precursor to the more general Finite Element Method. Developing the program in Scilab is straight forward since matrix operations are built into Scilab. Attention can thus be focused on the DSM rather than on developing and debugging an entire set of functions for matrix operations.

The tutorial presumes that the reader is familiar with the direct stiffness method (DSM) of structural analysis and the associated terminology. No effort is made in this tutorial to explain the DSM in depth, instead the focus is on writing the functions to implement it.

This is the first version of this document and will certainly contain errors, typographical as well as factual. You can help improve this document by reporting all errors you find and by suggesting modifications and additions. Your views are always welcome. I can be reached at the email address given on the cover page.

Acknowledgments

It goes without saying that my first indebtedness is to the developers of Scilab and the consortium that continues to develop it. I must also thank Mr. A.B. Raju, Assistant Professor, E&EE Department, BVBCET who first introduced me to Scilab and forever freed me from using Matlab®.

Finally, I would like to thank my wife Uma, daughter Deepti and son Prajwal for letting me take so much time away that I should in fact be spending with them. Their forbearance is gratefully acknowledged.

Satish Annigeri

Introduction

Scilab is a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific applications. Developed since 1990 by researchers from [INRIA](http://www.inria.fr/index.en.html) (French National Institute for Research in Computer Science and Control, <http://www.inria.fr/index.en.html>) and [ENPC](http://www.enpc.fr/english/int_index.htm) (National School of Bridges and Roads, http://www.enpc.fr/english/int_index.htm), it is now maintained and developed by [Scilab Consortium](http://scilabsoft.inria.fr/consortium/consortium.html) (<http://scilabsoft.inria.fr/consortium/consortium.html>) since its creation in May 2003.

Distributed freely and open source through the Internet since 1994, Scilab is currently being used in educational and industrial environments around the world.

Scilab includes hundreds of mathematical functions with the possibility to add interactively programs from various languages (C, Fortran...). It has sophisticated data structures (including lists, polynomials, rational functions, linear systems...), an interpreter and a high level programming language.

Scilab has been designed to be an open system where the user can define new data types and operations on these data types by using overloading.

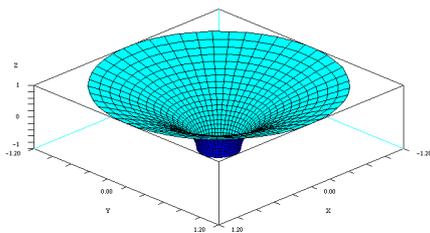
A number of toolboxes are available with the system:

- 2-D and 3-D graphics, animation
- Linear algebra, sparse matrices
- Polynomials and rational functions
- Simulation: ODE solver and DAE solver
- [Scicos](#): a hybrid dynamic systems modeler and simulator
- Classic and robust control, LMI optimization
- Differentiable and non-differentiable optimization
- Signal processing
- Metanet: graphs and networks
- Parallel Scilab using PVM
- Statistics
- Interface with Computer Algebra (Maple, MuPAD)
- Interface with Tck/Tk
- And a large number of contributions for various domains.

Scilab works on most Unix systems including GNU/Linux and on Windows 9X/NT/2000/XP. It comes with source code, on-line help and English user manuals. Binary versions are available.

Some of its features are listed below:

- Basic data type is a matrix, and all matrix operations are available as built-in operations.
- Has a built-in interpreted high-level programming language.
- Graphics such as 2D and 3D graphs can be generated and exported to various formats so that they can be included into documents.



To the left is a 3D graph generated in Scilab and exported to GIF format and included in the document for presentation. Scilab can export to Postscript and GIF formats as well as to Xfig (popular free software for drawing figures) and LaTeX (free scientific document preparation system) file formats.

Tutorial 1 – Data Organization for Matrix Analysis of Plane Frames

We shall design, implement and test a series of functions that, when put together, can analyze plane frames using the matrix method of structural analysis. Although it is possible to generalize the program so as to be able to analyze all kinds of skeletal structures, both 2D and 3D, the set of functions described here is targeted only at analyzing plane frames.

The first task in developing such a program is to clearly define the way data is organized within the program. The input data for a plane frame shall be organized as follows:

xy	Coordinates matrix. Size $n \times 2$, where 'n' is the number of nodes in the frame. It is assumed that row number corresponds to the node number. Thus row 'i' stores the coordinates of node number 'i'. Columns 1 and 2 correspond to the x and y coordinates of the node, respectively.
conn	Connectivity matrix. Size $m \times 3$, where 'm' is the number of members in the frame. Row number corresponds to the member number. Columns 1 and 2 correspond to the start and end node numbers of the member, respectively. Column 3 corresponds to Member Property ID, referring to the row number of the material property stored in the matrix 'mprop'.
bc	Boundary Constraint matrix. Size $ns \times 3$, where 'ns' is the number of supports (nodes with constrained degrees of freedom). Columns 1, 2 and 3 correspond to the constraint code for displacement along x-axis (u_x), displacement along y-axis (u_y) and rotation about z-axis (r_z) respectively. Constraint code 1 indicates that the degree of freedom (dof) is constrained and 0 indicates unconstrained.
mprop	Member Property matrix. Size $np \times 3$, where 'np' is the number of member properties required, one for each type. Columns 1, 2 and 3 represent Modulus of elasticity (E), Area of cross section (A) and Second moment of area with reference to the neutral axis of the section (I) respectively.
jtloads	Joint Loads matrix. Size $njl \times 4$, where 'njl' is the number of joint loads. Column 1 corresponds to the node number where the joint load is applied. Columns 2, 3 and 4 correspond to component of load along x-axis (F_x), component of load along y-axis (F_y) and Moment about z-axis (M_z) respectively. Joint load components must be in defined in global axes.
memloads	Member Load matrix. Size $nml \times 7$, where 'nml' is the number of member loads. Column 1 corresponds to the member number on which member loads are applied. Columns 2 to 7 correspond to F_x , F_y and M_z with reference to the global axes, at the start and end nodes of the member. The loads are calculated as the fixed end forces due to the applied member loads and are defined in member axes. Sign convention follows the right-hand rule.

Let us consider the example frame shown in Fig. 3 (Weaver & Gere, p. 243).

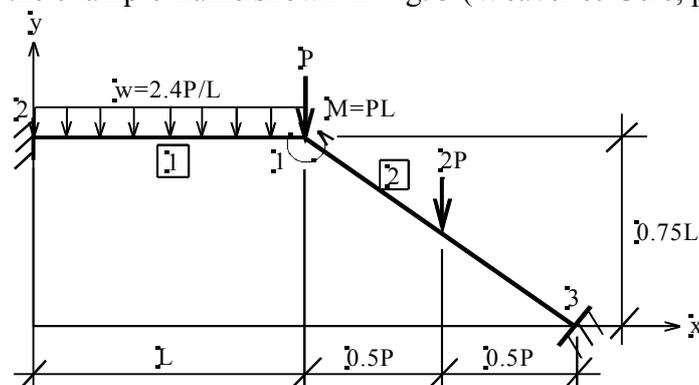


Fig. 3 Example plane frame (Weaver & Gere, p. 243)

The input data for the example in Fig. 3 is given below. The numbers within the tables constitute the data and are shown shaded. The rest are labels to help you interpret the data.

L=100 in
P=10 kips
E=10,000 ksi
A=10 in²
I=1000 in⁴

<i>xy</i>	<i>x</i>	<i>y</i>
1	100	75
2	0	75
3	200	0

<i>conn</i>	<i>Node i</i>	<i>Node j</i>	<i>Member Property ID</i>
1	2	1	1
2	1	3	1

<i>bc</i>	<i>Node</i>	<i>u_x</i>	<i>u_y</i>	<i>r_z</i>
1	2	1	1	1
2	3	1	1	1

<i>mprop</i>	<i>E</i>	<i>A</i>	<i>I</i>
1	1x10 ⁴	10	1x10 ³

<i>jtloads</i>	<i>Node</i>	<i>F_x</i>	<i>F_y</i>	<i>M_z</i>
1	1	0	-10	-1000

<i>memloads</i>	<i>Member</i>	<i>F_x</i>	<i>F_y</i>	<i>M_z</i>	<i>F_x</i>	<i>F_y</i>	<i>M_z</i>
1	1	0	12	200	0	12	-200
2	2	-6	8	250	-6	8	-250

Note that the units for the above data are kips (kilo pounds) and inches, as a result, E is in ksi (kilo pounds per square inch) and I is in in⁴. **The units used must be consistent and it is the responsibility of the user to ensure this.** The functions work well for any consistent system of units.

One thing that must be clearly understood is the definition of the local axis of the member. The local axis is defined with its origin at the start node of the member and the positive direction of the local x-axis is defined by the end node of the member. For example, member 1 in the above example has its x-axis starting from 1 and going towards 2, that is from right to left. The local y-axis is taken perpendicular to the x-axis and the local z-axis is kept the same as the global z-axis in the case of a plane frame.

The sign convention for the member fixed end forces is dependent on the member local x-axis, and the right-hand rule is used in deciding the positive sign convention for the forces and moments at the ends of the member.

Tutorial 2 – Location Matrix for a Plane Frame

In a plane frame, the number of degrees of freedom (dof) per node is 3, namely u_x , u_y and r_z , and therefore the total number of degrees of freedom is 3 times the number of nodes ('n'). However, every structure has supports with certain degrees of freedom constrained (otherwise the structure would be a free body capable of undergoing rigid-body motion). These constrained dof have zero displacements or non-zero prescribed displacements (such as support settlement problems). As a result, the number of unknown degrees of freedom ('ndof') is smaller than $3 \times n$.

In the direct stiffness matrix method, the portion of the stiffness matrix corresponding to zero displacements is not assembled as it does not affect the calculation of the nodal displacements. In case of non-zero prescribed displacements, the stiffness matrix must include the rows and columns corresponding to non-zero prescribed displacements, but before solving the stiffness equation, a rearrangement of the matrices is necessary. **This however is not attempted here and is left as an exercise for the student.**

A generalized representation of the dof numbering is called the location matrix. It has a size $n \times 3$, where 'n' is the number of nodes in the plane frame. Columns 1 to 3 store the constraint code for displacements along the 3 possible dof, namely, u_x , u_y and r_z for each node. The code is 0 for unconstrained dof and 1 for constrained dof.

The location matrix 'lm' is compiled in 2 stages. In the first stage, 'lm' is initialized to a zero matrix, implying that all dof are unconstrained. The number of nodes with constrained dof is available in column 1 of matrix 'bc'. For each of these nodes, the zeros in the corresponding rows of the location matrix are replaced by the constraint codes from 'bc'.

In the second stage, the variable representing the number of dof of the plane frame ('ndof') is initialized to 0 and 'lm' is processed row by row. For each constraint code 0 (unconstrained dof), 'ndof' is incremented by 1 and stored in the place of the 0. If the constraint code is 1 (constrained dof), it is set to 0 indicating the corresponding displacement to be 0.

At the end of this stage, 'ndof' will be the total number of dof of the plane frame and 'lm' will have zeros for a dof with zero displacement or a unique dof number for an unconstrained dof. The size of structure stiffness matrix must therefore be $ndof \times ndof$. The interface for a function to compute the location matrix is given below:

Interface: [lm, nd]=pf_calclm(n, bc)

Input Parameters: n=number of nodes in the plane frame, bc=Boundary constraint matrix.

Output Parameter: lm=Location matrix of a plane frame, nd=number of dof of the plane frame.

The function to carry out this computation is given below:

```
function [lm, nd]=pf_calclm(n, bc)
    lm=zeros(n,3);
    [ns dummy]=size(bc); // ns=number of supports
    for i=1:ns
        nn=bc(i,1); // node number of ith support
        lm(nn, 1:3) = bc(i, 2:4); // constraint codes of ith support
    end;
    nd=0; // initialize number of dof to zero
    for i=1:n
        for j=1:3
            if lm(i,j) == 1 then // constrained dof
                lm(i,j) = 0;
            else // unconstrained dof
                nd = nd + 1;
                lm(i,j) = nd;
            end
        end
    end
end
endfunction
```

This function must be loaded into Scilab and tested with the following commands:

```
-->[nodes dummy]=size(xy)
-->[lm, ndof]=pf_calclm(nodes, bc)
```

The first command determines the size of 'xy' matrix and puts the number of rows in 'nodes' and number of columns in 'dummy' (which is subsequently ignored). The second command invokes the function which returns the location matrix in 'lm' and the total number of dof is 'ndof'. The input to the function is the boundary condition matrix 'bc', which for the given structure is:

```
[ 2   1   1   1
  3   1   1   1
```

The location matrix is initialized to all zeros implying that the nodes are unconstrained. The rows corresponding to the supports, nodes 2 and 3 in this case, are then altered as defined in the matrix 'bc'. At this, the location matrix looks as shown below:

```
[ 0   0   0
  1   1   1
  1   1   1   ]
```

Before commencing the second pass, 'ndof' is initialized to zero and as each element is processed row by row, zeros are counted as permitted degrees of freedom ('ndof' is incremented and the incremented value is put in the place of the element) and a constrained dof (indicated by 1) is replaced by 0 (implying zero displacement at that dof). The output after the second pass, for our example problem is **ndof=3** and **lm** as follows:

```
[ 1   2   3
  0   0   0
  0   0   0   ]
```

Tutorial 3 – Local Stiffness Matrix of a Plane Frame Element

$$k = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix}$$

The stiffness matrix of a plane frame member with reference to its local axes is of size 6x6 and is given as shown on the left.

This can be generated by first defining a zero matrix of size 6x6, then defining the elements of the upper triangular matrix and finally copying the upper triangle into the lower triangle. The description of the function that will be developed for this purpose is as follows:

Interface: `[k]=pf_stiff(E, A, I, L)`

Input Parameters: **E** = Modulus of elasticity, **A** = Cross sectional area of the member, **I** = Second moment of area of the cross section about the neutral axis and **L** = Length of the member.

Output Parameter: **k** = Local stiffness matrix of a plane frame element of size 6x6.

The code for the function is as follows:

```
function [k] = pf_stiff(E, A, I, L)
    k = zeros(6,6);
    k(1,1) = E*A/L; k(1,4) = -k(1,1);

    k(2,2) = (12*E*I)/L^3; k(2,3) = (6*E*I)/L^2;
    k(2,5) = -k(2,2); k(2,6) = k(2,3);

    k(3,3) = (4*E*I)/L;
    k(3,5) = -k(2,3); k(3,6) = k(3,3)/2;

    k(4,4) = k(1,1);

    k(5,5) = k(2,2); k(5,6) = -k(2,3);

    k(6,6) = k(3,3);

    for i = 2:6
        for j = 1:i-1
            k(i,j) = k(j,i);
        end
    end
endfunction
```

Load this function into Scilab and test it with the following input:

```
-->k = pf_stiff(1,1,1,1)
```

The output is given on the next page. Another test is the following input, which gives the local stiffness matrix of member 2:

```
-->k = pf_stiff(1e4,10,1e3,125)
```

The output of this second command is also shown on the next page.

$$k = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 12 & 6 & 0 & -12 & 6 \\ 0 & 6 & 4 & 0 & -6 & 2 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -12 & -6 & 0 & 12 & -6 \\ 0 & 6 & 2 & 0 & -6 & 4 \end{bmatrix}$$

This is the stiffness matrix generated by the first command where E, A, I and L are all set to 1.

$$k = \begin{bmatrix} 800 & 0 & 0 & -800 & 0 & 0 \\ 0 & 61.44 & 3840 & 0 & -61.44 & 3840 \\ 0 & 3840 & 320000 & 0 & -3840 & 160000 \\ -800 & 0 & 0 & 800 & 0 & 0 \\ 0 & -61.44 & -3840 & 0 & 61.44 & -3840 \\ 0 & 3840 & 160000 & 0 & -3840 & 320000 \end{bmatrix}$$

This is the stiffness matrix generated by the second command where E=1x10⁴, A=10, I=1x10³ and L=125.

You will note that this function takes the length of the member as one of the input. We therefore need a function to calculate the length of a member. Later on, we will also need to do a related calculation, the direction cosines of the member. We will write a function that can do both these things. Its interface is as follows:

Interface: [l, cx, cy]=pf_calclen(imem, xy, conn)

Input Parameters: imem = number of the member whose length and direction cosines are to be calculated, xy = Coordinates matrix, conn = Connectivity matrix.

Output Parameters: l = Length of the member, cx = x direction cosine, cy = y direction cosine.

The code for the function is given below and the comments (which need not be typed when you try out the function yourself) are self-explanatory.

```
function [l, cx, cy]=pf_calclen(imem, xy, conn)
    n=conn(imem,:); // start and end nodes of member number i
    p1 = xy(n(1),:); // x,y coordinates of start node
    p2 = xy(n(2),:); // x,y coordinates of end node
    dxdy=p2-p1; // x,y projections of member
    dxdy2=dxdy.^2; // square of the projections
    l=sqrt(sum(dxdy2)); // length of member

    cx = dxdy(1) / l; // x-direction direction cosine of member
    cy = dxdy(2) / l; // y-direction direction cosine of member
endfunction
```

With this function available, we can compute the local stiffness matrix of any member by first calculating its length, extracting its material properties from 'mprop' and then computing its local stiffness matrix. For member 1, the commands would be:

```
-->[L cx cy]=pf_calclen(1,xy,conn)
-->iprop=conn(1,3)
-->E=mprop(iprop,1);A=mprop(iprop,2);I=mprop(iprop,3);
-->k=pf_stiff(E,A,I,L)
```

Tutorial 4 – Rotation Matrix of a Plane Frame Member

The local stiffness matrix is computed with reference to the local axis of a member, which may or may not be parallel to the global axes. Therefore it is necessary to transform the member stiffness matrix from local axes to global axes before it is superposed with stiffness matrices of other members in order to assemble the structure stiffness matrix. This transformation requires the rotation matrix of the member and is expressed in terms of the direction cosines of the member. The rotation matrix for a plane frame member is as given below:

$$r = \begin{bmatrix} C_x & C_y & 0 & 0 & 0 & 0 \\ -C_y & C_x & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_x & C_y & 0 \\ 0 & 0 & 0 & -C_y & C_x & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where C_x and C_y are the direction cosines of the member and are calculated as $C_x = \frac{dx}{l} = \frac{x_2 - x_1}{l}$ and $C_y = \frac{dy}{l} = \frac{y_2 - y_1}{l}$ where $l = \sqrt{dx^2 + dy^2}$.

The interface for the function is as follows:

Interface: `[r]=pf_calcrot(cx, cy)`

Input Parameters: `cx` = x direction cosine, `cy` = y direction cosine.

Output Parameters: `r` = Rotation matrix.

The code for the function is as follows:

```
function [r] = pf_calcrot(cx, cy)

    r = zeros(6,6); // initialize rotation matrix to zero

    r(1,1) = cx;
    r(1,2) = cy;
    r(2,1) = -r(1,2);
    r(2,2) = r(1,1);
    r(3,3) = 1;

    r(4:6, 4:6) = r(1:3, 1:3); // copy rows and columns 1:3, 1:3 into 4:6, 4:6
endfunction
```

Note that the direction cosines of the member must first be calculated using the `pf_calclen()` function and only then should the function to calculate the rotation matrix must be called. The commands to calculate the rotation matrix for member 2 are as follows:

```
-->[l cx cy]=pf_calclen(2,xy,conn)
-->r=pf_calcrot(cx,cy)
```

The output of these commands must be as follows:

$$r = \begin{bmatrix} 0.8 & -0.6 & 0 & 0 & 0 & 0 \\ -0.6 & 0.8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & -0.6 & 0 \\ 0 & 0 & 0 & 0.6 & 0.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Tutorial 5 – Global Stiffness Matrix of a Plane Frame Member

The global stiffness matrix of a plane frame member is calculated as $k = r' k^e r$ where k^e is local stiffness matrix and r is the rotation matrix of the member. Thus writing a function to calculate the global stiffness matrix of a plane frame member is straight forward once the functions to calculate the local stiffness matrix and rotation matrix are available. The interface for such a function is given below:

Interface: `[K]=pf_gstiff(imem, xy, conn, mprop)`

Input Parameters: `imem` = number of member whose global stiffness matrix is to be calculated, `xy` = coordinates matrix, `conn`=connectivity matrix and `mprop` = material property matrix.

Output Parameters: `K` = global stiffness matrix.

The code for the function is as follows:

```
function [K] = pf_gstiff(imem, xy, conn, mprop)

    [L cx cy] = pf_calclen(imem, xy, conn); // length and direction cosines
    r = pf_calcrot(cx, cy); // calculate rotation matrix

    iprop = conn(imem,3); // property id of imem th member
    E = mprop(iprop,1); // Modulus of elasticity
    A = mprop(iprop,2); // Area of cross section
    I = mprop(iprop,3); // Second moment of area of cross section about NA

    k = pf_stiff(E, A, I, L); // local stiffness matrix of ith member

    K = r' * k * r; // global stiffness matrix of imem th member
endfunction
```

The comments make the function self explanatory. To calculate the global stiffness matrices of the 1st and 2nd members, the function is used as follows:

```
-->k1=pf_gstiff(1,xy,conn,mprop)
-->k2=pf_gstiff(2,xy,conn,mprop)
```

The global stiffness matrix of member 2 must be as given below:

$$r = \begin{bmatrix} 534.1184 & -354.5088 & 2304 & -534.1184 & 354.5088 & 2304 \\ -354.5088 & 327.3216 & 3072 & 354.5088 & -327.3216 & 3072 \\ 2304 & 3072.0000 & 320000 & -2304 & -3072 & 160000 \\ -534.1184 & 354.5088 & -2304 & 534.1184 & -354.5088 & -2304 \\ 354.5088 & -327.3216 & -3072 & -354.5088 & 327.3216 & -3072 \\ 2304 & 3072 & 160000 & -2304 & -3072 & 320000 \end{bmatrix}$$

Tutorial 6 – Assembling the Plane Frame Structure Stiffness Matrix

The structure stiffness matrix of a plane frame is obtained by computing the global stiffness matrix for an individual member of the plane frame and superposing it with the structure stiffness matrix. To be able to do so, one needs to determine a mapping between the rows and columns of the global stiffness matrix of a member and those of the structure stiffness matrix. This is done by writing the dof numbers for the rows and columns of the member global stiffness matrix and superposing the elements with the corresponding elements of the structure stiffness matrix. Thus, we need to know the numbers of the start and end nodes of a member and the corresponding dof numbers from the location matrix 'lm'.

Interface: `[ssm]=pf_assemssm(imem, xy, conn, mprop, lm, ssm)`

Input Parameters: `imem` = number of member whose global stiffness matrix is to be calculated, `xy` = coordinates matrix, `conn` = connectivity matrix and `mprop` = material property matrix, `lm` = location matrix of the member and `ssm` = structure stiffness matrix of the plane frame.

Output Parameters: `ssm`=structure stiffness matrix of the plane frame.

The code for the function is as follows:

```
function [ssm] = pf_assemssm(imem, xy, conn, mprop, lm, ssm)

    K = pf_gstiff(imem, xy, conn, mprop);

    nj = conn(imem,1); nk = conn(imem,2);
    dof(1:3) = lm(nj,1:3);
    dof(4:6) = lm(nk,1:3);

    for i=1:6
        ii = dof(i);
        if ii == 0 then

        else
            for j=1:6
                jj = dof(j);

                if jj == 0 then

                else
                    tmp = ssm(ii,jj) + K(i,j);
                    ssm(ii,jj) = tmp;
                end
            end
        end
    end
endfunction
```

Note that `ssm` appears both as an input as well as an output parameter, as this function superposes the global stiffness matrix of one member onto the existing structure stiffness matrix, which is initialized to zero to start with. To compute the structure stiffness matrix considering all members, we need to call this function inside a `for` loop. The interface to the function that does this is shown below:

Interface: `[K]=pf_ssm(xy, conn, mprop, lm, ndof)`

Input Parameters: `xy` = coordinates matrix, `conn` = connectivity matrix and `mprop` = material property matrix, `lm` = location matrix of the member and `nd` = number of degrees of freedom of the plane frame.

Output Parameters: `K` = structure stiffness matrix of the plane frame.

The code for the function is given below:

```
function [K]=pf_ssm(xy, conn, mprop, lm, ndof)
    K=zeros(ndof,ndof);
```

```

[nmem dummy]=size(conn);
for imem = 1:nmem
    K = pf_assemssm(imem, xy, conn, mprop, lm, K);
end
endfunction

```

The input data **xy**, **conn**, **mprop** must first be read and the number of degrees of freedom (**ndof**) and location matrix (**lm**) must be computed before the function **pf_ssm()** is called. This function returns the structure stiffness matrix (**K**) of the plane frame.

To illustrate the steps for member 1, the start end end nodes are 2 and 1 respectively, as can be seen from the '**conn**' matrix. The dof numbers for these nodes are obtained from the location matrix '**lm**', and are seen to be [0 0 0] and [1 2 3] for nodes 2 and 1 respectively. This implies that rows and columns of the global stiffness matrix of member 1 correspond to rows [0 0 0 1 2 3] of the global stiffness matrix of the structure. Rows and columns with dof number zero correspond to zero displacement and therefore need not be assembled into the structure stiffness matrix. The elements of the member global stiffness matrix where both the row and column dof numbers are non-zero are superposed on the elements in the corresponding row and column of the structure stiffness matrix.

Tutorial 7 – Assembling the Load Vector

The load vector has the same number of rows as the number of degrees of freedom (dof) of the structure ('ndof'). Each element of the load vector represents the load applied corresponding to a dof number. Loads applied on a plane frame can be either loads applied at the nodes of the frame or loads applied on the members. The two different types of loads have to be processed differently before assembling the load vector.

Joint loads are an input data to the program and are specified as components in global coordinate system (components along x, y axes and moment about z axis). Knowing the node number at which the load is applied, it is easy to identify the dof numbers corresponding to the node from the location matrix. The components of the load at a node are then superposed with the corresponding element in the load vector.

The interface to the function that assembles the load vector for loads applied at the joints is given below:

Interface: `[P]=pf_assemloadvec_jl(lm, jtloads, P)`

Input Parameters: `lm` = location matrix of the member and `jtloads` =matrix of joint loads, `P` = load vector.

Output Parameters: `P` = load vector.

This function assembles the load vector for a plane frame from the loads applied directly on the nodes.

```
function [P]=pf_assemloadvec_jl(lm, jtloads, P)
[nloads dummy] = size(jtloads);
printf('Loads = %d\n', nloads);
for i=1:nloads
    n=jtloads(i,1);
    printf('Load applied at joint %d', n);
    dof=lm(n,:);
    disp(dof, ' with dof = ');
    for j=1:3
        jj = dof(j);
        if jj == 0 then

            else
                tmp = P(jj) + jtloads(i, j+1);
                P(jj) = tmp;
            end
        end
    end
end
endfunction
```

This function returns a 1x6 vector containing the dof numbers of the start and end nodes of member number `imem`. The degree of freedom (dof) numbers in the first 3 rows correspond to the start node and the last 3 to the end node.

```
function [dof]=pf_getdof(imem, conn, lm)
dof=zeros(1,6);
n1 = conn(imem,1);
dof(1:3) = lm(n1,:);
n2 = conn(imem,2);
dof(4:6) = lm(n2,:);
endfunction
```

This interface to the function that assembles the load vector for a plane frame from the loads applied on the members is given below:

Interface: `[P]=pf_assemloadvec_ml(iload, xy, conn, lm, memloads, P)`

Input Parameters: `iload` = number of the member load being processed, `xy` = coordinates matrix, `conn` = connectivity matrix, `lm` = location matrix of the member and `memloads` = matrix

of member loads, \mathbf{P} = load vector.

Output Parameters: \mathbf{P} = load vector.

Since member loads are in member axes, they are transformed to global axes using the rotation matrix. This function calculates the load vector due to one of the member loads and superposes it on to the previous load vector.

```
function [P]=pf_assemloadvec_ml(iloal, xy, conn, lm, memloads, P)
    imem = memloads(iloal,1);
    [L cx cy]=pf_calclen(imem,xy,conn);
    r = pf_calcrot(cx, cy);
    am=-r' * memloads(iloal,2:7)';
    dof=pf_getdof(imem, conn, lm);
    for i=1:6
        ii = dof(i);
        if ii ~= 0 then
            P(ii) = P(ii) + am(i);
        end
    end
endfunction
```

To assemble the load vector due to all member loads, this function has to be called once for each member load, within a `for` loop, as follows:

```
[nmemlds dummy]=size(memloads);
for iloal=1:nmemlds
    [P]=pf_assemloadvec_ml(iloal, xy, conn, lm, memloads, P);
end
```

The number of member loads `nmemlds` is the number of rows in the member load matrix `memloads`. The load vector \mathbf{P} has to be initialized to zero at the start. The contribution to the load vector due to joint loads may first be assembled and after that the contribution due to the member loads may be assembled.

Tutorial 8 – The Super Function

The functions developed until now accomplish the task of assembling the structure stiffness matrix of a plane frame and assembling the load vector. The next task is to write a single function that calls these functions in the right sequence and solves the stiffness equation for the nodal displacements.

The interface to such a function is given below:

Interface: `[P]=pf(xy, conn, bc, mprop, jtloads, memloads)`

Input Parameters: `xy` = coordinates matrix, `conn` = connectivity matrix, `bc` = boundary condition matrix, `mprop` = member property matrix, `jtloads` = joint load matrix, and `memloads` = member load matrix.

Output Parameters: `ssm` = structure stiffness matrix, `x` = nodal displacements and `P` = load vector.

This is the super function that performs the complete task of assembling the structure stiffness matrix, assembling the load vector and solving the stiffness equation to obtain the nodal displacements.

```
function [ssm,x,P]=pf(xy,conn,bc,mprop,jtloads,memloads)

    [nodes dummy]=size(xy);

    [lm ndof]=pf_calclm(nodes,bc);
    ssm=pf_ssm(xy,conn,mprop,lm,ndof);

    P=zeros(ndof,1);
    x=zeros(ndof,1);

    P=pf_assemloadvec_jl(lm, jtloads, P);

    [nmemlds dummy]=size(memloads);
    for iload=1:nmemlds
        [P]=pf_assemloadvec_ml(iload, xy, conn, lm, memloads, P);
    end

    x=zeros(ndof,1);
    x=inv(ssm)*P;
endfunction
```

All the input data must be gathered before calling this function. After this function is called, the nodal displacements are known and the task that remains is to extract the member end forces from the calculated nodal displacements.

Tutorial 9 – Extracting Member End Forces

Once the nodal displacements are available, calculating the member end forces in global axes is possible if we know the displacements pertaining to the nodes of the member concerned. This information is available in the location matrix and can easily be obtained. To express the member end forces in local axes, the member end forces are transformed from global to local axes. The interface to the function that computes the member end forces in local axes for a member with number `imem` is given below:

Interface: `[P]=pf_memendforces(imem, xy, conn, mprop, lm, x, memloads)`

Input Parameters: `imem` = number of member for which member end forces are to be calculated, `xy` = coordinates matrix, `conn` = connectivity matrix, `mprop` = member property matrix, `lm` = location matrix, and `x` = nodal displacement vector.

Output Parameters: `f` = member end force vector of size 6x1.

This function computes the end forces for member number '`imem`' in local axes.

```
function [f] = pf_memendforces(imem, xy, conn, mprop, lm, x)
    iprop=conn(imem,3); // Property ID for member imem
    E=mprop(iprop,1); A=mprop(iprop,2); I=mprop(iprop,3);
    [L cx cy]=pf_calclen(imem,xy,conn); // Length and direction cosines
    r=pf_calcrot(cx,cy); // Rotation matrix

    k = pf_stiff(E,A,I,L); // Local stiffness matrix

    u = zeros(6,1); // Initialize member end displacements
    dof = pf_getdof(imem, conn, lm); // Get DOF numbers for the ends of member
    for i = 1:6
        idof = dof(i);
        if idof ~= 0 then
            u(i) = x(idof); // Copy global displacement into u
        end
    end
    uu=r*u; // Displacements in local axes

    f = zeros(6,1);
    f = k * uu; // Member end forces in local axes

    [nmemloads,dummy] = size(memloads);
    for i = 1:nmemloads
        if memloads(i,1) == imem
            f = f + memloads(i,2:$)';
        end
    end
endfunction
```

This function has to be called once for each member and the results stored. This can be done at the Scilab command prompt.

```
-->[k,x,P]=pf(xy,conn,bc,mprop,jtloads,memloads)
-->[nodes dummy]=size(xy);
-->lm=pf_calclm(nodes,bc);
-->f1=pf_memendforces(1,xy,conn,mprop,lm,x,memloads)
-->f2=pf_memendforces(2,xy,conn,mprop,lm,x,memloads)
```

Data for the problem shown has been stored in a file called `pf.bin`, and you can load it with the command:

```
-->load('pf.bin')
```

Ensure that `pf.bin` is located in your working directory. The output of the Scilab functions is compared with the results from the book:

Displacements

Node	u_x		u_y		r_z	
	Scilab	Book	Scilab	Book	Scilab	Book
1	-0.0202608	-0.02026	-0.0993600	-0.09936	-0.0017976	-0.001797

Member End Forces

Member		F_{x1}	F_{y1}	M_{z1}	F_{x2}	F_{y2}	M_{z2}
1	Scilab	20.260769	13.137825	436.64755	-20.260769	10.862175	-322.86504
	Book	20.26	13.14	436.6	-20.26	10.86	-322.9
2	Scilab	28.72592	-4.5332787	-677.13496	-40.72592	20.533279	-889.52488
	Book	28.72	-4.53	-677.1	-40.73	20.53	-889.5

There is another data file `hall.bin` containing the input data to Example 15.1 from Hall & Kabaila (Hall & Kabaila, p. 300) which you can try. You could try working backwards from the input data and determine its geometry in case you can't get hold of the book.

References

1. Hall, A. and Kabaila, A.P., Basic Concepts of Structural Analysis, Pitman Publishing, London, 1977.
2. Kassimali, A., Matrix Analysis of Structures, Brooks/Cole Publishing Company, Pacific Grove, 1999.
3. Weaver Jr., W. and Gere, J.M., Matrix Analysis of Framed Structures, 2ed., CBS Publishers and Distributors, New Delhi, 1986.

Notes