

MATLAB Structural Analysis Code for String Wing Box Structure

A Senior Project

Presented to

The Faculty of the Aerospace Engineering Department
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

by

Sean Brown

June 14, 2010

© 2010 Sean Brown

MATLAB Structural Analysis Code

Sean M. Brown

California Polytechnic State University San Luis Obispo, CA 93401

This report outlines the method used for analysis of a wing box structure for the experimental “stringy” wing structure often used in light RC aircraft. This code is able to find the displacement of each joint, and the stress and forces in each member of the truss structure. It also has the features of load and structure visualization, Kevlar string removal, user-defined point and distributed loading functions, and user-defined failure criteria function. This method and resulting code is meant for use in the support of the Human Powered Helicopter project being undertaken by the Aerospace and Mechanical Engineering Department at California Polytechnic State University San Luis Obispo. This code may also be used for the Design Build Fly team at Cal Poly.

Nomenclature

A	=	cross-sectional area of bar
E	=	Young’s modulus
K	=	system stiffness matrix
R	=	Reaction at the end of each bay
L	=	Length
c	=	cosine
d	=	displacement matrix
ff	=	forces vector
h	=	index number
i	=	index number
j	=	index number
k	=	element stiffness matrix
kk	=	stiffness matrix
l	=	index number
s	=	sine
$sdof$	=	system degrees of freedom

I. Introduction

The structure of aircraft wings are usually composed of ribs in the stream-wise direction, spars and stringers in the span wise direction. The skin that covers the wing is also used as a main structure. Each of these structures is used to carry any load applied to the aircraft wing. The spars are mainly used for bending loads applied to the wing such as weight and lift, while it can carry some torsion loads, the skin are the main carriers of the torsion loads. Stringers are used to stiffen the skin and carry loads from the skin to the ribs, which carry the load to the spars.

The human powered helicopter being designed and built at California Polytechnic University San Luis Obispo is part of the Human Powered Helicopter competition put on by the American Helicopter Society. From 1981 to 1989 the Da Vinci series of HPHs were built by Cal Poly, with the last of which, the Leonardo III had a 100 foot rotor span, tip to tip. The entire structure without the rider weighed 97 lbs, which emphasizes the importance of light-weight main rotor construction (Patterson, 2001). When considering the weight of the wing as the driving factor, it becomes important to look at alternatives to the normal rigid materials used in large aero vehicles. The new wing design being looked at for the competition is dubbed the “Stringy” wing, for its use of strings between ribs in order to increase strength without increasing weight significantly. Weight is such important part of all areas of this project that it is necessary to test and optimize the wing structure before it is built to full scale. Creation of a structural analysis code will optimize the sizes and strengths of the load bearing members of the wing box. The main bending load bearing members specifically, the spars were looked at the structure analysis and optimization. The simplified model is shown as a MATLAB figure in Figure 1.

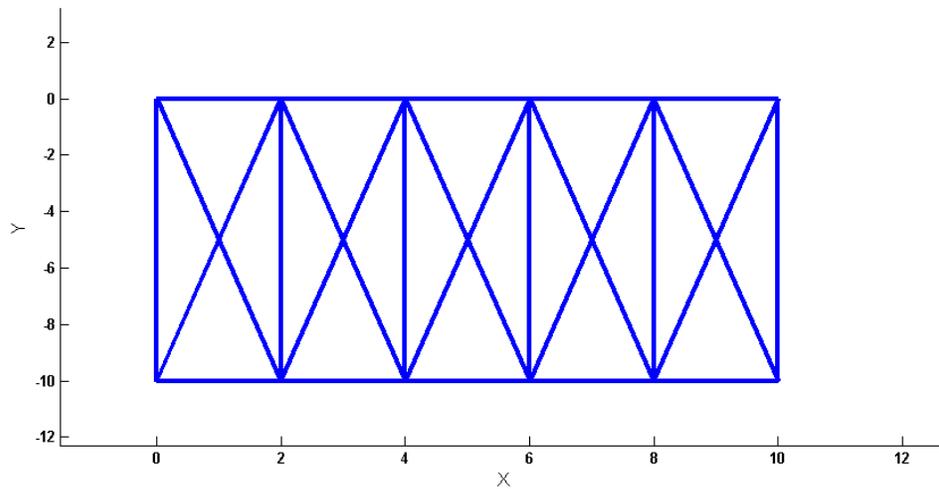


Figure 1. Simplified Wing Box Model

II. The Code

A. Finite Element Method

Formally known as Matrix Structural analysis, Finite Element Method was developed during the initial investigations of aeroelastic behavior. The method remained in the Aerospace community for a long time. In the earlier years, matrix formulation and inversion were all done using slide rulers, and pocket calculators, but with the invention of electronic computers, the method began to reach maturity. The finite element methods are classified by two main categories, displacement method and force method. Other names for these are stiffness method and flexibility method, respectively (Felippa, 2000).

The displacement method is the most common method used in commercial structural analysis programs today (Kwon & Bang, 1997). These methods differ in the choice of the primary unknowns (Felippa, 2000). The amount of documentation and information available relating to the stiffness method made it the obvious choice for application to the wing box structural analysis problem.

This document assumes a general knowledge of the MATLAB conventions. It is important to be familiar with function handles, evaluating functions and scripts, and general MATLAB notation.

A. TrussFEA

The code is organized into one m-file, a function called TrussFEA. All functions that are called within TrussFEA, with the exception of the ones specified by the user, are after the end of the main function.

The inputs to TrussFEA are as follows: "Joints", "Bars", "Kevlar", "Loading", "figuretoggle", "DistLoads", "PtLoads", and "FailureCrit".

"Joints" needs to be an i -by-3 matrix that specifies the joints on the truss, where " i " is the number of joints in the truss. The first column is the x -coordinate of each joint, the second column is the y -coordinate of each joint, and the third column is the type of boundary conditions that are applied to this joint. If the joint is constrained in the x -direction only, the third column is a 1, if constrained in the y -direction the third column is a 2, if constrained in the x - and y -direction the 3rd column is a 3.

The "Bars" input is expecting a j -by-4 matrix, where j is the number of bars in the truss. The first column of the matrix is the initial joint of the bar and the second column is the final joint of the bar. The numbering of the joints is done first in increasing x -coordinate and then in increasing y -coordinate, so the first joint will be the one that is in the bottom left. The third column in the bars matrix is the Young's modulus of the bar. The fourth column is the area of the cross-section of the bar. The units of the last two columns must have the same units, either English or SI; the units used will determine the units which displacement and stress are output.

The "Kevlar" input is used specifically for the stringy wing. This input has the same format as "Bars," however it specifies strings which can take no compressive load.

The "Loading" input is expecting an h -by-3 matrix, where h is the number of loads applied to the structure. The loads can only be applied at the joints, so the first column is the joint where the load is applied, using the same numbering scheme used in the Bars input.

The "figuretoggle" input is a single number that if one displays all figures produced by the code and if zero, suppresses all figures.

"DistLoads" input is a function that describes the distributed loading applied to the truss. The "PtLoads" input is a function that is used to specify the point loads applied to the truss. "FailureCrit" is a function that specifies the failure criteria for the truss. "DistLoads", "PtLoads", and "FailureCrit" format and features will be described later in the Loading Functions section. If the "DistLoads" and "PtLoads" function are specified, the code will ignore any loading matrix that is input and overwrite it with the one

generated using the “DistLoads” and “PtLoads” function. A variable argument input is also used which will be transferred to the “FailureCrit” function later in the code. This variable argument input will be used to capture any additional variable that are necessary for the “FailureCrit” function to evaluate.

The outputs of the TrussFEA function are “Stresses”, “displacement”, “AxialForces”, “Failure.” The Stresses contain a breakdown of the stresses experienced by each bar. Stresses output is formatted as an i-by-2 matrix with the first column being the number of the bar, and the second column is the stress experience by the bar. The displacement output is a matrix of the displacement of each joint. Each joint has two degrees of freedom, so each joint has two displacements, one in the x-direction and one in the y-direction. For example, if there are 10 joints, displacement will be a 20-by-2 matrix. The AxialForces output lists the axial forces experienced by each bar. AxialForces is in the same format as the Stresses output. The Failure output is the direct output from the FailureCrit function that was input.

TrussFEA is broken into commented section as follows: Settings, Loading, Truss Plotting, Matrix building, Boundary Conditions, Solving for displacements, Solving for stresses, Failure Criteria, and plotting Deflection. Each section is commented out with a double percent sign and is followed by a rough description of the contents of each section.

1. Settings

Settings contain any settings that aren’t changed in the code very often. For example, the settings section contains the actual loading plot option, explained later in the features section.

2. Loading

The Loading section takes the point loads function and distributed loads function can creates the loading matrix for the joints. In this section the joints matrix will be sorted by the position of the joint in the span wise direction. The truss that is described by the joints matrix is split into bays, where a bay is the space between joints in a truss. The position of the start and end of the bays are used to find the reactions at each joint. To find the reactions that act at each joint, the code steps through each bay treating the individual bays as fixed end bar problems with a loading scheme between them. By splitting the bays up in this way, the reactions at the joints can be found. The matrix of the reactions is kept in the variable “R”.

The next section in loading is the actual building of the loading matrix, the format of which is described earlier. The loading matrix is built so that loads are applied to the top and bottom joints of the truss. If two truss joints are located in the same y-coordinate, the load that is experience there is split in half between the top and bottom. If the load that the joint experiences is negative the load will be in the negative y-direction, where if it is positive it will be in the positive y-direction.

3. Truss Plotting

This feature is useful when constructing the truss and loading that it experiences. The joints, bars and loading are represented on the plot. Green arrows at each joint show the load experienced at that joints. The numbers that are displayed next to the truss joints are used for the user reference when constructing the bars matrix. Figure 1 shows an example of the truss plot. The truss plot section will also plot the actually loading that the structure is experiencing, this is described further in the features section.

4. Matrix building

The matrix building section cycles through each bar and builds an element stiffness matrix, which is then used to influence the overall system stiffness matrix. The variable “k” represents the element system matrix, while “K” represents the system stiffness matrix. Please refer to reference 3 for a detailed description of how the element and system stiffness matrices are built.

5. Boundary Conditions

This section creates the “bcdof” and “bcval” matrices, which are used to edit the system stiffness matrix to represent the boundary conditions, applied on the truss. The creation the force vector, “ff,” is also put in this section. The boundary conditions are created from the joints matrix, which describes how each joint is constrained, i.e. x-direction, y-direction or both. The function “feaplyc2” is then used to “zero-out” the proper rows in the system stiffness matrix using the boundary conditions specified. The function “feaplyc2” will be described more later.

6. Solving for displacements.

All of the system and force matrix building culminates in this section with the division of the system stiffness matrix by the force matrix giving the displacement of each degree of freedom of each joint.

7. Solving for the stresses

Using the displacements that where found at each joint, the stresses are computed from the element stiffness matrices. The element stiffness matrices are built as shown in the matrix building section.

8. Failure Criteria

This section takes the failure criteria function that is input and runs it with the inputs of stresses, bars and displacements. The outputs, “Failure” and “msg” are also outputs of the TrussFEA function, so the user can handle the failure of a bar as they wish.

9. Plotting Deflection

Using the displacements given for each degree of freedom, the new position of each joint is found and plotted against the original truss.

B. Support Functions

The TrussFEA m-file contains several functions. The functions other than the TrussFEA function are used to support the main function. The support functions are fetruss2, feaplyc2, feeldof, and feasmb11. All of these support functions were found in reference 3.

1. *fetruss2*

This function is used to generate the element stiffness matrix of each bar. In this function the bar is defined by its angle with the horizontal, as well as the area of the bar cross section, “A”, the Young’s Modulus of the Bar, “E” and the length of the bar, “L.” The matrix described in Equation X shows the cosine-sine matrix, which is used to describe the orientation of each element.

$$k = \frac{AE}{L} * \begin{bmatrix} c^2 & c * s & -c^2 & -c * s \\ c * s & s^2 & -c * s & -s^2 \\ -c^2 & -c * s & c^2 & c * s \\ -c * s & -s^2 & c * s & s^2 \end{bmatrix}$$

2. *feaplyc2*

This function uses the boundary condition values and their corresponding degrees of freedom to apply the boundary conditions to the system stiffness matrix. The degree of freedom corresponds to the rows in the system stiffness matrix. The full function is shown in the appendix

3. *feeldof*

This function is used when generating the system stiffness matrix, K. Using the element index and the degree of freedom index, the index is created for where the element stiffness matrix fits into the system stiffness matrix. This function is a major part of the code, and to understand it is to understand the entire process that is being used.

4. *Feasmbll*

This function takes in the index generated in feeldof function and adds the element stiffness matrix to the system stiffness matrix in the proper place. The function is used for each bar in the matrix.

C. Loading Functions

Using the loading functions as inputs give the user a great deal of freedom when it comes to integration with other applications. The distributed loading function and the point loading function have a general format that allows easy use by the operator. Originally, it would be dependent on the user to find the appropriate forces at each joint and then input them into the algorithm. Using the loading functions, however, move the algorithm toward a more model specific focus.

1. *Point Loading Function*

This function is held in TrussFEA as the variable “PtLoads.” An example point loading function is given in the appendix. The point loading function has inputs of “Xmin” and “Xmax”, and has the output of “ptloads.” “Xmin” and “Xmax” describe a range that is connected to the bay that is being evaluated. So when the inputs of “Xmin” and “Xmax” are entered, the ptloads variable must be a i-by-2 matrix of the point loads that lie within that range. The first column is the location of the point load in the span wise direction. The second column is the magnitude of the load being applied at the described point. The sign convention described earlier applies. negative is in negative y-direction and positive is in positive y-direction.

2. *Distributed Loading Function*

The distributed load function follows the point loading function’s style of simplicity. The input for the function is a span wise location, or x location. The output for the function is the load experienced at input location. An example of an elliptical loading is shown in the appendix.

Procedures

The following section outlines the procedures that can be used with this code. These are only general guidelines and the user should take the time to understand the format in order to efficiently tailor the code to their specific problem or application.

A. Example

The full script used for these examples is shown in Appendix A. The first example deals with a standard truss that is show plotted in Figure 2. The joints matrix is shown in (3). The bar connectivity used for this example are shown in (4). Two Kevlar strings were added across the diagonal, 1 to 4 and 2 to 5. A young’s modulus of 20e6, and an area of .01867 were used for the third and forth columns of the bars matrix. The Kevlar had young’s modulus of 20e5 and the same area. Pass the loading and failure functions into the TrussFEA function as function handles. Pass an empty matrix in for the loading input. The script can now be run, the solved truss is shown in Figure 3, while the displacements, and stresses are shown Table 1.

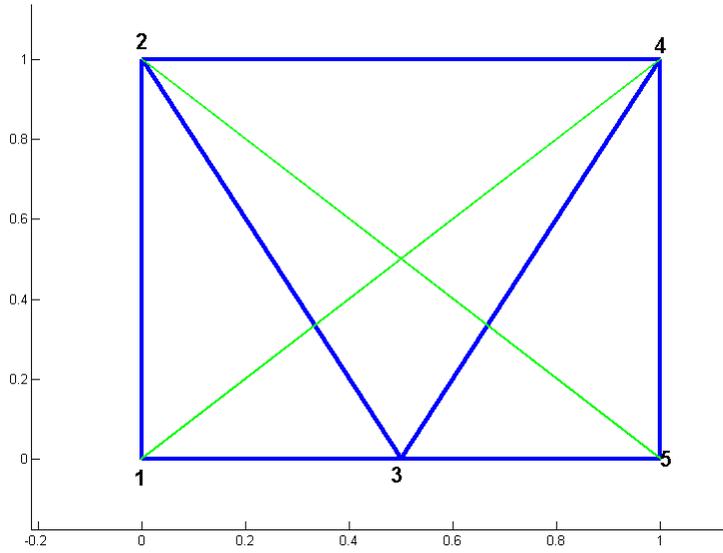


Figure 2. Standard Truss with Kevlar string

$$JOINTS = \begin{bmatrix} 0 & 0 & 1 & 1 & .5 \\ 0 & 1 & 1 & 0 & 0 \\ 3 & 3 & 0 & 0 & 0 \end{bmatrix}^T \quad (3)$$

$$BARS = \begin{bmatrix} 1 & 3 & 1 & 2 & 2 & 3 & 4 \\ 2 & 5 & 3 & 3 & 4 & 4 & 5 \end{bmatrix}^T \quad (4)$$

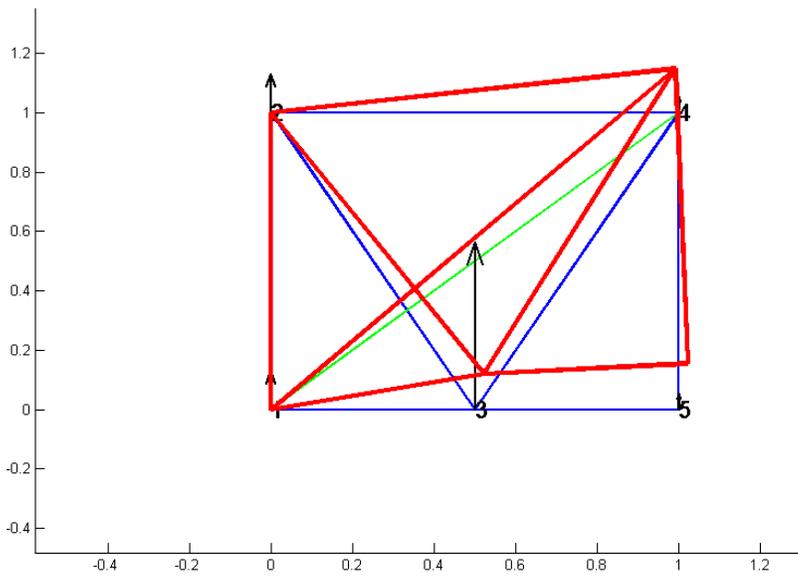


Figure 3. Solved Truss

Table 1. Stresses and Displacements

Degree of Freedom	Displacement	Bar	Stress
1	-3.7042e-17	1	1.6649e-09
2	8.1756e-16	2	-1.2176e-11

3	1.4612e-16	3	86920.331
4	-1.489e-17	4	-175500.5317
5	0.0217	5	18859.238
6	0.1205	6	-18237.824
7	-0.0091	7	13335.966
8	0.147	8	13864.548
9	0.0217		
10	0.1544		

From Figure 3, you can see that the Kevlar string that spanned from joint 2 to joint 5 was removed because it was in compression. Looking at the table above, we see that the first four displacements are getting very close to zero, which demonstrates that they are the boundary conditions that were constrained.

B. Features

This code has a few features that will make it more useful for the user. The code has the following functionalities: Actual Load Plotting, Failure Criteria, and Kevlar Functionality.

The actual load plotting can be activated by changing the variable “ActualLoadPlot” from a zero to a one. This will create a second plot. The truss structure with bars will be plotted, along with the distributed load and point loading given by the black and red arrows, respectively. This is meant to be used when constructing the loading functions. The scale on the arrows is changed so the length of the arrow only relates to the amount of force in relation to the arrows around it. The feature is shown in Figure 4. An elliptical loading distribution along with some point loads is plotted. The loading functions are shown in Appendix C.

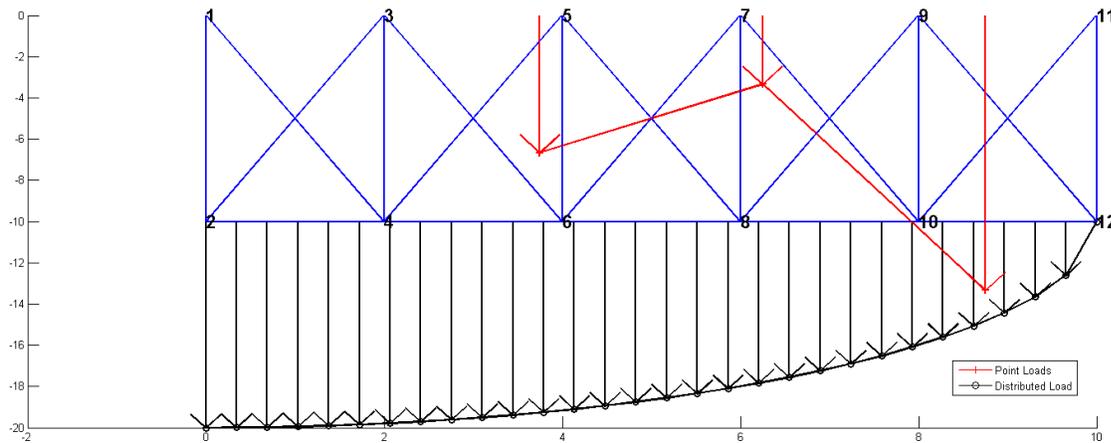


Figure 4. Actual Loading Feature

The Failure criteria feature is used to allow for more flexibility on the part of the user. The failure of a material is dependent on a nearly infinite amount of factors. So the failure criteria function allows the user to create criteria that they deem necessary to solve their problem. The code that uses the failure criteria function is shown in the Failure Criteria section of TrussFEA.

The Kevlar Functionality is specific to the design and development of light weight structures. The Kevlar input has the same format as the Bars input, however any connects specified by the bar can only support tensile stress, just as a Kevlar string would. The TrussFEA algorithm will calculate the stresses in all connections including the Kevlar connections, and then it will check for any negative stresses (compression) in the Kevlar connection. If any exist the connections will be removed and the analysis will be run again without it. The process of removing strings will continue until there are no Kevlar strings in compression. The Kevlar strings will be displayed in the truss plot as green lines, compared to the blue lines of the normal bars. If the Kevlar functionality is not desired, making an empty “Kevlar” matrix to satisfy the “Kevlar” input will allow the user to run the algorithm without this feature.

III. Discussion

This algorithm although extremely simple has much potential for use in the design and optimization of wing structures. This algorithm should only be treated as the first step in analysis of a structure. After narrowing down designs, a high fidelity FEA analysis should commence to further design, optimize and verify.


```
Loading=[];
```

```
[Stress,Displacements,AxialForces,Failure]=TrussFEA(Joints,Bars,Kevlar,  
Loading,figuretoggle,DistLoads,PtLoads,FailureCrit);
```

B. TrussFEA

```
function
```

```
[Stresses,displacement,AxialForces,Failure]=TrussFEA(Joints,Bars,Kevlar,Loading,...  
figuretoggle,DistLoads,PtLoads,FailureCrit,varargin)
```

```
% JOINTSOLVER uses method of joints to solve for all the forces in a truss structure
```

```
% Inputs:
```

```
% Joints specifies the joints of the truss, must be a i-by-3 matrix. First column  
specifies  
% the x coordinate of each joint, and the 2nd column specifies the y-coordinate. The  
3rd column is  
% used to designate the type of constraint that it has, no constraint(=0), x-  
constraint (=1), y-constraint (=2)  
% x and y constraint(=3).
```

```
% Bars specifies the connections between the points and must be a i-by-8  
% matrix, where the first column is the first joint of the bar and the  
% second column is the second joint. The joints are numbered right to left,  
% top to bottom. Third column represents the Modulus of elasticity of the  
% bar, column 4 represents Area of each bar. For checking the Buckling of  
% the bars, going to add the thickness, Length, and radius at the end.
```

```
% Truss Loading:
```

```
% Can either be done using the Loading matrix:  
% Loading is a j-by-3 matrix of the forces with the format of joint, angle (measured  
from the positive x-axis), force (N).
```

```
% OR can use the distloads and the PtLoads:
```

```
% The dist load is a function that given a X-value, or the distance from the  
% root to X, the function will output the load due to the distributed  
% loading at the point given by X.  
% The PtLoads is a function which has two inputs, Xmin and Xmax. Given Xmin  
% and Xmax, PtLoads will output a matrix of point loads. This matrix is be  
% in the form of [distance from root, load;...]
```

```
% If you provide a DistLoads and PtLoads Function, it will ignore the  
% original Loading Condition and opt for the new one.
```

```
%% Settings:
```

```
ActualLoadPlot=1;  
LoadFunction=1;
```

```

%% Kevlar

NumberOfBars=size(Bars,1); % Number of Bars in the Structure
NumberOfKevlar=size(Kevlar,1); % Number of Kevlar Strings
BarsFull=Bars; % A complete Bar matrix for when it iterates

CompressionString=1; % Initializing the logical matrix: if true, string in
compression

while ~any(CompressionString==1)==0 % Used for taking out the Kevlar strings
% Robustness for kevlar Functionality
    if isempty(Kevlar)==0
        Bars=cat(1,BarsFull,Kevlar); % Putting bars together for run through
algorithm
    else
        Bars=BarsFull; % If there are no strings left, just use the bars matrix
    end

    % Clearing previous data- so no overwrite residuals
    clear Stress StringStress CompressionString AxialForce Angle k K index BayEnd
BayFull BayStart Distance E I Loading P U V X Xc bcdof bcval eldisp ptload mu index
ff t r L

    %% Loading
    % Will change the Distloads or PtLoads into the j-by-3 loading matrix

    if LoadFunction==1
        Loading=[]; % Deciding to use the Loading Functions
    end

    if isempty>Loading)==1
        Loading=nan;

        DistMoment=@(x,Xmin) DistLoads(x).*(x-Xmin); % Moment at any given point
between Xmin and Xmax

        [Joints,index]=sortrows(Joints,1); % Sorting the row of the joints so they
are in order.

        % Making the different bays: A bay specifies two different x-axis joints
        BayStart=0;
        h=1;
        l=1;

        for i=1:size(Joints,1)
            % Assumes no built-in dihedral
            if Joints(i,1)~=BayStart(h)
                h=h+1;
                BayStart(h)=Joints(i,1);

                if h==2
                    BayEnd(1)=BayStart(h);

                    elseif exist('BayEnd')==1

                        if Joints(i,1)~=BayEnd(end)
                            BayEnd(length(BayEnd)+1)=Joints(i,1);
                        end
                    end
                end
            end
        end
    end

```

```

        end
    end
end

Bays=h-1; % Number of bays is the number of different
BayFull=BayStart;
BayStart=BayStart(1,1:length(BayStart)-1);

R=zeros(Bays+1,1);
for i=1:Bays
    Xmin=BayStart(i); % defining the minimum point for this section
    Xmax=BayEnd(i); % Defining the maximum point for this section
    ptload=PtLoads(Xmin,Xmax); % Getting the point loads for this section
    PMoment=(ptload(:,1)-Xmin)'*ptload(:,2); % Point Moment about Xmin
    DMoment=quad(DistMoment,Xmin,Xmax,[],[],Xmin); % Distributed Load Moment
    about Xmin
    Mt=PMoment+DMoment; % Total Moment
    Lt=sum(ptload(:,2))+quad(DistLoads,Xmin,Xmax); % Total Load
    Xc=Mt/Lt; % Determining the point where the total load is applied
    R(i)=R(i)+(Xc*Lt)/(Xmax-Xmin); % Reaction at Xmax
    R(i+1)=R(i+1)+(Lt-R(i)); % Reaction at Xmin
end

% Find the same joints in vertical dir
P=find(diff(Joints(:,1))==0);
P=P+1; % Use this so when iterating in the for loop it will stop on the
second in the pair of repeated vector

for i=1:size(Joints,1)
    Loading(i,1)=i; % Documenting the joint for the loading matrix
    I=BayFull==Joints(i,1); % Making a logical where the bay starts and
ends==the Joints.
    Loading(i,3)=R(I); % Using the reactions to fill in the correct Loading
matrix
    if any(i==P)==1 % Checking if the current Joint matches any of the
repeated joints
        Loading(i,3)=Loading(i,3)/2; % Splitting the force between the top and
bottom joints
        Loading(i-1,3)=Loading(i-1,3)/2; % Splitting the force between the
top and bottom joint
    end

    if Loading(i,3)<0;
        Loading(i,2)=-90; % if the force is negative, make it point down
    elseif Loading(i,3)>0
        Loading(i,2)=90; % if the force is positive, make it point up.
    end
    Loading(:,3)=abs(Loading(:,3));
end

end

%% Truss Plotting

if figuretoggle==1
    figure(1)

```

```

clf
hold on
% Adding the bars
for i=1:NumberOfBars
    plot([Joints(Bars(i,1),1) Joints(Bars(i,2),1)], [Joints(Bars(i,1),2)
Joints(Bars(i,2),2)], 'b');
end

% Plot The Kevlar
if isempty(Kevlar)==0
    for i=1:size(Kevlar,1)
        plot([Joints(Kevlar(i,1),1)
Joints(Kevlar(i,2),1)], [Joints(Kevlar(i,1),2) Joints(Kevlar(i,2),2)], 'g');
    end
end

% Adding the joint numbers
for i=1:size(Joints,1)

text(Joints(i,1),Joints(i,2),num2str(i), 'fontsize',16, 'fontweight', 'bold')
end
axis([min(Joints(:,1))-min(Joints(:,1))/3-
1,max(Joints(:,1))+max(Joints(:,1))/3,min(Joints(:,2))+min(Joints(:,2))-
1)/2,max(Joints(:,2))+max(Joints(:,2))/3]);
% Initializing the Quiver matrices
X=zeros(size(R,1),1);
Y=zeros(size(R,1),1);
U=zeros(size>Loading,1),1);
V=zeros(size>Loading,1),1);

% Generating the
for i=1:size>Loading,1)
    X(i)=Joints>Loading(i,1),1); % X-Coordinate
    Y(i)=Joints>Loading(i,1),2); % Y-coordinate
    U(i)=Loading(i,3)*cos>Loading(i,2)*pi/180); % X-load
    V(i)=Loading(i,3)*sin>Loading(i,2)*pi/180); % Y-load
end

LoadingHeight=abs(max(Joints(:,2)))-abs(min(Joints(:,2)));
% Dimensionalizing the Loads for plotting
ReductionFactor=LoadingHeight/max(U);
U=U.*ReductionFactor;

% Plotting the Load forces at the joints
quiver(X,Y,U,V,'k') % Plotting the loading arrows a the joints

hold off

if ActualLoadPlot==1
    figure(2)
    hold on

    % Adding the bars
    for i=1:size(Bars,1)
        plot([Joints(Bars(i,1),1) Joints(Bars(i,2),1)], [Joints(Bars(i,1),2)
Joints(Bars(i,2),2)]);
    end
    % Adding the joint numbers
    for i=1:size(Joints,1)

```

```

text(Joints(i,1),Joints(i,2),num2str(i),'fontsize',16,'fontweight','bold')
end
axis([min(Joints(:,1))-min(Joints(:,1))/3-
1,max(Joints(:,1))+max(Joints(:,1))/3,min(Joints(:,2))+
(min(Joints(:,2))-1)/2,max(Joints(:,2))+max(Joints(:,2))/3]);

DensityofPoints=3; % Number of points per unit for the distributed
loading plot
NumberofPoints=DensityofPoints*abs(max(Joints(:,1))-min(Joints(:,1))); %
Number of points

ActualLoadingX=linspace(min(Joints(:,1)),max(Joints(:,1)),NumberofPoints); % x
location of dist loading

for i=1:length(ActualLoadingX)
    ActualLoadingY(i)=DistLoads(ActualLoadingX(i)); % Getting Actual Dist
loading
end

ReductionFactor=LoadingHeight/abs(max(ActualLoadingY)-
min(ActualLoadingY)); % Finding load reduction
ActualLoadingY=ActualLoadingY.*ReductionFactor+max(Joints(:,2)); %
reducing the load

Y_actual(1:length(ActualLoadingY))=LoadingHeight; % Actual loading y-loc
for quiver

quiver(ActualLoadingX,Y_actual,zeros(1,length(Y_actual)),(ActualLoadingY),0,'k') %
distributed loads arrows

h2=plot(ActualLoadingX,ActualLoadingY+LoadingHeight,'ko-');

% Getting the Actual Point loading
ActualPtloads=PtLoads(ActualLoadingX(1),ActualLoadingX(end)); % Calling
Point load function
ReductionFactor=LoadingHeight/abs(max(ActualPtloads(:,2))-
min(ActualPtloads(:,2))); % finding reduction of load for display
ActualPtloads(:,2)=ActualPtloads(:,2).*ReductionFactor+max(Joints(:,2));
% reducing the load
ActualPtloads=sortrows(ActualPtloads,1); % organizing the point load

ptloadY=zeros(size(ActualPtloads,1),1);
ptloadY(:,:)=0;

h1=plot(ActualPtloads(:,1),ActualPtloads(:,2),'r+-'); % Plotting the
point load lines

quiver(ActualPtloads(:,1),ptloadY,zeros(1,length(ptloadY))',(ActualPtloads(:,2)),0,'r')
% Point load arrows
legend([h1,h2],'Point Loads','Distributed Load')
axis auto
hold off
end

```

```

end
%% Matrix Building
% Builds the element stiffness matrix, which builds system stiffness matrix

nnel=2; % number of nodes per element
ndof=2; % degrees of freedom of each element

% initializing variables
index=zeros(nnel*ndof,1); % Vector used to determine the number of each dof
sysdof=size(Joints,1)*ndof; % System Degrees of freedom
K=zeros(sysdof,sysdof); % Stiffness matrix
E=Bars(:,3); % Modulus of elasticity for the elements
A=Bars(:,4); % Area for the elements

for i=1:length(Bars(:,1))
    Distance(i,1)=Joints(Bars(i,1),1)-Joints(Bars(i,2),1); % Distance in the x-
dir of each element
    Distance(i,2)=Joints(Bars(i,1),2)-Joints(Bars(i,2),2); % Distance in the y-
dir of each element

    if (Distance(i,1))==0
        Angle(i)=2*atan(1); % Angle Calculation
    else
        Angle(i)=pi-atan(Distance(i,2)/-Distance(i,1)); % Angle Calculation
        if Angle(i)==pi
            Angle(i)=0;
        end
    end
end

L(i)=sqrt(Distance(i,1)^2+Distance(i,2)^2); % Length of Bars

% Element stiffness matrix & System Stiffness matrix
% Using the intialized stiffness matrix and filling it in for
% each element and just overwriting it

% Creating the index for the specific bar
l=0;
for j=1:nnel
    start=(Bars(i,j)-1)*ndof;
    for h=1:ndof
        l=l+1;
        index(l)=start+h;
    end
end

AEL=(A(i)*E(i)/L(i));
[k]=fetruss2(AEL,Angle(i)); % Building the element stiffness matrix

```

```

%      filling in the system stiffness matrix

eldof=nnel*ndof;

for j=1:eldof
    jj=index(j);
    for h=1:eldof
        hh=index(h);
        K(jj, hh)=K(jj, hh)+k(j, h); % Building the System Stiffness matrix
    end
end

end

%% Boundary conditions
% Uses the boundary conditions specified by the joints matrix to edit the
% system stiffness matrix

% Constructing the Boundary condition vectors
k=0;
for i=1:length(Joints)
    if Joints(i,3)==1
        c=2*i-1;
        k=k+1;
        bcdof(k,1)=c;
        bcval(k,1)=0;
    elseif Joints(i,3)==2
        c=2*i;
        k=k+1;
        bcdof(k,1)=c;
        bcval(k,1)=0;
    elseif Joints(i,3)==3
        c=2*i-1;
        k=k+1;
        bcdof(k,1)=c;
        bcval(k,1)=0;
        c=2*i;
        k=k+1;
        bcdof(k,1)=c;
        bcval(k,1)=0;
    end
end

end

% System Force
% Consists of a vector of the forces in the x and y for each joint

% Assumes that the loads are in spanwise order
ff=zeros(sysdof,1);
l=0;
for i=1:size>Loading,1)
    l=l+1;
    ff(2*Loading(i,1)-1)=ff(2*Loading(i,1)-
1)+Loading(i,3)*cos>Loading(i,2)*pi/180);
    ff(2*Loading(i,1))=ff(2*Loading(i,1))+Loading(i,3)*sin>Loading(i,2)*pi/180);
end

```

```

if l~=sysdof/2
    keyboard
end

% Applying the boundary conditions
[K,ff]=feaplyc2(K,ff,bcdof,bcval);

%% Solve for displacements
% Uses matrix division to calculate the displacements

Sing=det(K);
if Sing==0
    display('>>>>>Matrix is badly conditioned---Check Manual<<<<<<<')
    keyboard
end

d=K\ff;

%% Solve for reactions
% Uses the displacements to solve for the stresses and axial forces.

for i=1:length(Bars(:,1))
    % Element stiffness matrix & System Stiffness matrix
    % Using the intialized stiffness matrix and filling it in
    % for each element and just overwriting it

    % Creating the index for the specific bar
    l=0;
    for j=1:nnel
        start=(Bars(i,j)-1)*ndof;
        for h=1:ndof
            l=l+1;
            index(l)=start+h;
        end
    end

    AEL=(A(i)*E(i)/L(i));
    [k]=fetruss2(AEL,Angle(i));

    % filling in the system stiffness matrix

    eldof=nnel*ndof;
    for j=1:eldof
        eldisp(j)=d(index(j));
    end

    %
    elforce=k*eldisp'; % give the forces at a node on a bar
    AxialForce(i)=sqrt(elforce(1)^2+elforce(2)^2);
    Stress(i)=sqrt(elforce(1)^2+elforce(2)^2)/A(i);
end

```

```

    % Checking if the element is in tension or compression
    if (Distance(i)*elforce(3))>0
        Stress(i)=-Stress(i);
        AxialForce(i)=-AxialForce(i);
    end

end
if NumberOfKevlar>0
    % Checking Kevlar Compression
    StringStresses=Stress(NumberOfBars+1:NumberOfKevlar+NumberOfBars);
    clear CompressionString
    CompressionString=zeros(1,length(StringStresses)); % Initializing the
variable while will record if there is any strings in compression

    for i=1:length(StringStresses)
        if StringStresses(i)<0
            CompressionString(i)=1;
        end
    end

    Kevlar=Kevlar(CompressionString==0,:);
    NumberOfKevlar=size(Kevlar,1);
else
    CompressionString=0;
end

end

% Printing The solution

num=1:1:sysdof;
d=[num' d];

num=1:1:size(Bars,1);
AxialForces=[num' AxialForce'];

num=1:1:size(Bars,1);
Stresses=[num' Stress'];

displacement=d;

%% Failure Criteria
% Checks for failure base on the criteria specified.

[Failure,msg]=FailureCrit(Bars,Stress,d,varargin);

%% Plotting deflection
if figuretoggle==1
    for i=1:length(Joints)

```

```

        Joints(i,1)=Joints(i,1)+d(2*i-1,2);
        Joints(i,2)=Joints(i,2)+d(2*i,2);
    end

    figure(1)
    hold on

    for i=1:size(Bars,1)
        plot([Joints(Bars(i,1),1) Joints(Bars(i,2),1)], [Joints(Bars(i,1),2)
Joints(Bars(i,2),2)], 'r');
    end

    hold off
end

end

```

C. Loading Functions

1. Point Loading Test Function

```

function [ptloads]=PointLoadingTestcase(Xmin,Xmax)
%% Point Loading Test function for the input conversion code.
% This is where you need to make sure that you dont double count the point
% loads, it all depends on the logical signs in the if statment

%% Point loads
span=1;

ptload(1,:)=[3,1000]; % Engine
ptload(2,:)=[7,2000]; % Armament
ptload(3,:)=[5,500]; % Engine
ptload(4,:)=[8,1000]; % Armament

reductionfactor=span/abs(max(ptload(:,1)));
ptload(:,1)=ptload(:,1).*reductionfactor;

X=zeros(1,1);
L=zeros(1,1);
c=0;
for i=1:size(ptload,1)
    if ptload(i,1)<Xmax && ptload(i,1)>=Xmin
        c=c+1;
        X(c)=ptload(i,1);
        L(c)=ptload(i,2);
    end
end
X=X';
L=L';
ptloads=cat(2,X,L);

end

```

2. *Distributed Loading Test Function*

```
function [L]=DistributedTestCase(x)
%% Distributed Loading Function Test case
% Simulate an elliptical loading distribution with a weight loading.
Span=1; % ft
maxforce=2000;
% L=-maxforce*(x-Span/2).^2+maxforce*(Span/2)^2;
L=maxforce*sqrt(1-(x/Span).^2);% elliptical lift distribution

% if x<Span/2
%     L=L-500; % Uniform load for anything less than half span
% end

end
```

D. Support Functions

1. *Feaplyc2*

```
function [kk,ff]=feaplyc2(kk,ff,bcdof,bcval)

%-----
% Purpose:
%     Apply constraints to matrix equation [kk]{x}={ff}
%
% Synopsis:
%     [kk,ff]=feaplybc(kk,ff,bcdof,bcval)
%
% Variable Description:
%     kk - system matrix before applying constraints
%     ff - system vector before applying constraints
%     bcdof - a vector containing constrained d.o.f
%     bcval - a vector containing constrained value
%
%     For example, there are constraints at d.o.f=2 and 10
%     and their constrained values are 1.0 and 2.5,
%     respectively. Then, bcdof(1)=2 and bcdof(2)=10; and
%     bcval(1)=1.0 and bcval(2)=2.5.
%-----

n=length(bcdof);
sdof=size(kk);

for i=1:n
    c=bcdof(i);
    for j=1:sdof
        kk(c,j)=0;
    end

    kk(c,c)=1;
    ff(c)=bcval(i);
end

end
```

2. *Fetruss2*

```
function [k]=fetruss2(AEL,Angle)
```

```

c=cos(Angle);
s=sin(Angle);

k=AEL*[ c^2,c*s,-c^2,-c*s;...
        c*s, s^2, -c*s, -s^2;...
        -c^2,-c*s,c^2,c*s;...
        -c*s, -s^2, c*s, s^2];
end

```

References

- Bruhn, E. (1973). *Analysis and design of Flight Vehicle Structures*. Indiana: Jacobs Publishing, INC.
- Felippa, C. A. (2000). *A historical Outline of Matrix Structural Analysis: A Play in Three Acts*. Boulder,CO: University of Colorado.
- Kwon, Y. W., & Bang, H. (1997). *The Finite Element Method using MATLAB*. Boca Raton, London, New York, Washington, D.C.: CRC Press.
- Langford, J. (1989, August 2). The Daedalus Project: A Summery of Lessons Learned. *American Institute of Aeronautics and Astronautics*, p. 6.
- Patterson. (2001, January 16). *The History of the Human Powered Helicopter Project*. Retrieved June 6, 2010, from Cal Poly Mechanical Engineering Department: <http://www.calpoly.edu/~wpatters/davi.jpg>