Lab 7 Lecture

Through the course of today’s lecture and this handout, the four functions required for the completion of laboratory #7 will be gone over. Ideally, the information contained in this document, when combined with the lecture and the given problem statement, will guide you to successfully complete the first portion of writing your own 3D frame analysis code.

General Report Format

Please refer to the final project description provided in the lecture by Prof. Paulino for additional information concerning this project. Note: these labs should be professional in appearance and organization.

Comments About Completing Laboratories #7-10

For each laboratory, you will be required to provide functions that complete the tasks required for the laboratory. We will be providing you with a suggested form and approach for these functions however, you are free to follow our suggestions or adopt your own method. The details of your approach are no concern to us other than if the approach you take works and satisfies the laboratory requirements. No matter what approach you take, you must provide clear evidence that your function works as intended and that the laboratory requirements are satisfied.

Grading

Laboratories #7-9 will be out of 20 points each. The point distribution will be 40% for the functions and 60% for discussion / verification of the functions. This means that if you do not do the verifications / discussions for the functions, the maximum you may receive for the assignment will be 8 points. The last laboratory will be out of 40 points and have a similar point distribution.

Verifications

The format and requirements of the function verifications will remain consistent with the previous laboratories. However, we will not be providing set problems for you to test your program against until laboratory #10. You are free to determine what approach you wish to take to clearly demonstrate that your function works as intended and all laboratory requirements are satisfied. One approach to complete these verifications is to come up with your own quality example where you solve the example using your function and also using some other method not including your function. Quality examples include but are not limited to examples that fully test the function and also demonstrate your understanding of the logic behind the function, the function's limitations and uses of the function.
Solutions

The solutions to a laboratory will be posted relatively soon after the due date. The functions submitted in a laboratory are often needed in the following laboratory therefore it is prudent to check your function against the solution as soon as possible. This will help avoid any potential "snowball" effect caused by an error in a submitted function.

Function 1: ud_webvect.m: finds the local y-axis

NOTE: please name all the files that you create for the remaining labs with the same first three characters, “ud_” (represents that the function was user defined).

Before we talk about ud_webvect.m specifically, lets review a couple of things about coordinate systems.

2 key properties of the x, y, z coordinate system

• Each axis is defined by a unit vector
  
  \[ \hat{x} = (1, 0, 0) \]
  
  \[ \hat{y} = (0, 1, 0) \]
  
  \[ \hat{z} = (0, 0, 1) \]

• All the axes are perpendicular to each other, i.e. orthogonal. This means that the dot product between any two of the unit vectors will be zero. Correspondingly, the cross product between any two of the unit vectors will equal ± the third unit vector.

\[ \hat{i} \times \hat{j} = \hat{k} \]
\[ \hat{j} \times \hat{i} = -\hat{k} \]
\[ \hat{k} \times \hat{i} = \hat{j} \]
\[ \hat{i} \times \hat{k} = -\hat{j} \]
\[ \hat{j} \times \hat{k} = \hat{i} \]
\[ \hat{k} \times \hat{j} = -\hat{i} \]

The signs of the answer follows the “right-hand” rule
Cross Product

Question: what physical meaning is associated with the cross product of two vectors?

Answer: a vector that is perpendicular to the plane defined by the 2 vectors. If the vector is perpendicular to the plane, it is also perpendicular to the 2 vectors. The direction of the vector is found by obeying the “right-hand” rule. Just keep this in mind for later.

Let’s talk 2D for a minute

In the 2D case, we would assign the local coordinate system for each element by aligning the local x-axis with the element and the local y-axis perpendicular to the local x-axis.

Finding \( x' \) is simple enough but how do we find \( y' \)? There is a simple but clever way to find \( y' \) without having to calculate the angle \( \theta \) between the local and global coordinate systems.

What happens when you take the cross product between the normal to the xy plane, i.e. \( \hat{k} \), and the local \( x' \)?

You get a vector perpendicular to the plane defined by the z-axis and the \( x' \)-axis and in the xy plane. This vector is \( y' \) !!!

Note: lower case letters with apostrophes will symbolize the local system, capital letters will symbolize the global system.
Now for the good stuff: the 3D case

Is x’ always going to be in the xy plane? No for a 3D structure, x’ can be in any direction. So how do we modify the procedure we just used to find the local coordinate system in 2D to find the local coordinate system in 3D?

The First Step

- By definition, an element’s cross section is perpendicular to the x’ axis and is therefore defined in the y’z’ plane.

The process to define the local coordinate system starts off the exact same in 3D. First, we define x’ as the unit vector aligned with the element. Then, let’s take the cross product between x’ and Y and lets call it \( \vec{z} \). The “~” notation is adopted here to signify a temporary variable. Where in the 2D case, these “~” vectors defined the local coordinate system, we still have one more step before we can define the 3D local coordinate system for a given element. Anyway, the vector \( \vec{z} \) is a vector perpendicular to x’ and Y even though x’ and Y are not perpendicular to each other (remember the earlier discussion about the physical meaning to the cross product). Also note that \( \vec{z} \) will always be in the XZ plane.

\[
\begin{align*}
x’ &= A\hat{i} + B\hat{j} + C\hat{k} \\
Y &= D\hat{j}
\end{align*}
\]

\[
(x’) \times (Y) = \begin{vmatrix} i & j & k \\ A & B & C \\ 0 & D & 0 \end{vmatrix} = -CD\hat{i} + AD\hat{k}
\]

Now that we know x’ and \( \vec{z} \), can we find \( \vec{y} \) where \( \vec{y} \) is a vector perpendicular to both x’ and \( \vec{z} \)?

Yes, just take the cross product between \( \vec{z} \) and x’ and you get the desired vector \( \vec{y} \). Now all three vectors are perpendicular to each other. Note: if you know 2 of the coordinate axes, you can always find the third axis using the cross product.
The Next Step

We know the vector $z'$ will not be constrained to the XY plane, but our $\tilde{z}$ vector is constrained to the XZ plane. How do we remove this limitation and then determine $z'$ and $y'$?

The idea is that by finding $\tilde{y}$ and $\tilde{z}$, we have defined the plane that is perpendicular to $x'$. Remember that two vectors define a plane. Thus, to find $z'$ and $y'$, we will just rotate the vectors $\tilde{z}$ and $\tilde{y}$ an angle $\beta$ about the $x'$ axis and call the resulting two vectors $z'$ and $y'$.

RESULT

\[
y' = \tilde{y}\cos(\beta) + \tilde{z}\sin(\beta)
\]
\[
z' = -\tilde{y}\sin(\beta) - \tilde{z}\cos(\beta)
\]

This process is exactly what ud_webvect should do. The function takes the element coordinates, finds the unit vector $x'$, finds the unit vector $\tilde{z}$, finds $\tilde{y}$, and then applies a rotation $\beta$ to find $y'$. The unit vector $y'$ is called the web vector because it is aligned with the beam’s web. What should be outputted from ud_webvect.m is a matrix of “nele” rows and three columns where “nele” is the number of elements in the structure. The first column is the x component of $y'$, the second column is the y component of $y'$ and the third column is the z component of $y'$.
Problems?

Question: So when does the process just described not work?

Answer: when \( x' \) is the same as \( Y \)!!!

When you take the cross product of 2 parallel vectors, you get zero. So how do we deal with this case? Earlier, we just decided to use \( Y \) to find \( \tilde{z} \). We could have used any vector in the \( xy \) plane that is not aligned with \( x' \). If \( x' \) is the same as \( Y \), let's make the vector that we use in the cross product with \( x' \) be \((-1,0,0)\).

Once you know \( \tilde{z} \) and \( x' \), we can find \( y' \) using the exact same process as before.

In code:

First, find the unit vector that defines \( x' \). Then, find the vector to cross with \( x' \) to determine \( \tilde{z} \).

```matlab
x = % to be determined by you
if abs(x(2))<0.9999 % the process to determine Y
    Y = [0 1 0];
else
    Y = [-1 0 0];
end
```

Function 2: ud_order.m: dof numbering

This function numbers the degrees of freedom (dofs) in a structure. The convention used is different than used for previous laboratories. In the previous laboratories, we would assign the dofs based on the node number and the number of dofs per node. For example, the dof numbering for node 4 would be 10, 11 and 12 in a frame. These numbers would be assigned without consideration to whether or not the dof was supported or free. For the remaining four laboratories, we are going to take a different approach.
Instead of numbering the dofs based on the node number, we will go around the structure numbering the free dofs with a positive number and the fixed or supported dofs with a negative number. We will start the numbering scheme at node 1 and then progress to node 2.

**Example**

Question: What would the dof numbering be for the given figure if we only had three dofs per node? (note: the structures we will be coding for will require 6 dofs per node)

Answer:

![Diagram](ud_order.m)

**In code**

For this function, you have two inputs: nnodes and fixity. The variable “nnodes” is the number of nodes in the structure and “fixity” is a matrix that contains the information necessary to determine if a given dof is free or supported. The output is the number of free and the number of supported dofs and also a matrix named NORDER. The matrix NORDER contains the free and supported dof #’s corresponding to each nodes 6 dofs.

What you want to do is write a code that reads the components of fixity. If a component is NaN (not a number), then that dof is free and should be numbered with the next available positive number in the dof numbering scheme. If the component has a value, the dof is supported and the specified displacement is equal to that value.

**Example**

Question: Given that fixity = 

\[
\begin{bmatrix}
0 & -2 & 1 & 0 & 0 & 0 \\
NaN & NaN & NaN & NaN & NaN & NaN \\
0 & 0 & 1 & NaN & NaN & NaN
\end{bmatrix}
\]

What is NORDER?
Answer: \[ \text{NORDER} = \begin{bmatrix}
-1 & -2 & -3 & -4 & -5 & -6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
-7 & -8 & -9 & 7 & 8 & 9
\end{bmatrix} \]

Matlab Hint: use the built-in function \texttt{isnan()} to evaluate whether or not a number is a “number” or “not a number”.

**Function 3: ud\_pfdisps.m:** determines the free dof load vector and the supported displacement vector

The input matrix “\texttt{concen}” contains the value of any applied loads. Specifically, the “ith” row of the matrix contains the magnitude of any loads applied to the corresponding ”ith” node. The dof in which the load is applied determines which column of the six that the value is placed. Likewise, the “ith” row of the input matrix “\texttt{fixity}” contains the value of any prescribed displacement to the corresponding “ith” node.

**Example**

Question: Given the above matrices “\texttt{fixity}” and NORDER, what is the output vector \( \text{DISP}_S \)?

Answer: \( \text{DISP}_S = (0 \ -2 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)^T \)

The same approach is used to find the output vector \( \text{PF} \) but using “\texttt{concen}” instead of “\texttt{fixity}”.

**Function 4: ud\_trans.m:** defines the transformation matrix for a frame element

\[
[T] = \begin{bmatrix}
[R] & [0] & [0] & [0] \\
[0] & [R] & [0] & [0] \\
[0] & [0] & [R] & [0] \\
[0] & [0] & [0] & [R]_{12\times12}
\end{bmatrix}
\]

where \( [R] = \begin{bmatrix} x(1) & y(1) & z(1) \\
x(2) & y(2) & z(2) \\
x(3) & y(3) & z(3) \end{bmatrix}_{3\times3} \) and \( [0] = \begin{bmatrix} 0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \end{bmatrix}_{3\times3} \)

The vectors “x”, “y” and “z” are the unit vectors that define the local coordinate system of the element.