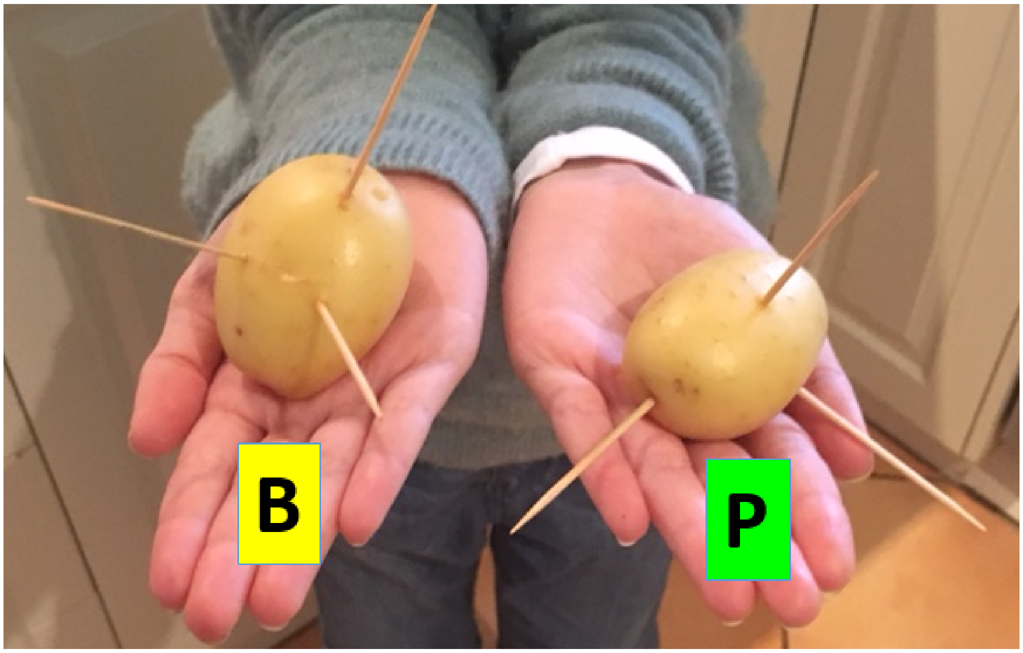
Principal Moments of Inertia

In this task we're going to look at how we can calculate the principal axes and principal moments of inertia, of a rigid body. Recall our well known angular momentum equation $^{B}\mathbf{L} = {}^{B}\mathbf{I} \times {}^{B}\boldsymbol{\omega}$, where the *B-frame* about which we are determining the angular momentum is both BODY fixed AND at the body's centre of mass:

$$\begin{pmatrix} {}^{B}L_X \\ {}^{B}L_Y \\ {}^{B}L_Z \end{pmatrix} = \begin{pmatrix} {}^{B}I_{XX} & {}^{B}I_{XY} & {}^{B}I_{XZ} \\ {}^{B}I_{XY} & {}^{B}I_{YY} & {}^{B}I_{YZ} \\ {}^{B}I_{XZ} & {}^{B}I_{YZ} & {}^{B}I_{ZZ} \end{pmatrix} * \begin{pmatrix} {}^{B}_{G}\omega_X \\ {}^{B}_{G}\omega_Y \\ {}^{B}_{G}\omega_Z \end{pmatrix} \quad \text{where} \quad \begin{aligned} {}^{B}I_{ZZ} &= \int (X^2 + Y^2)dm \\ {}^{B}I_{XY} &= \int (-X * Y)dm \end{aligned} \quad \text{etc.}$$

What we would like to do is to determine another co-ordinate frame, call it the *P-frame*, which is also at the body's centre of mass, but the inertia matrix about this new *P-frame* is diagonal. We call this *P-frame* the **PRINCIPAL** frame and the corresponding diagonal inertia matrix the **PRINCIPAL** moment of inertia matrix ^{P}I, ie:

$$^{P}L = {}^{P}I \times {}^{P}\omega \qquad where \qquad {}^{P}I = \begin{pmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{pmatrix}$$



What we're going to do:

Key topics that we'll cover in this task are:

- review Eigenvalues for a general matrix
- review Eigenvalues for a **SYMMETRIC** matrix
- review passive rotations and the DCM

After reviewing these topics we'll present the solution to determining the **PRINCIPAL** moments of inertia.

Bradley Horton : 01-Mar-2016, bradley.horton@mathworks.com.au

ATTENTION: before we start I want the following acknowledged:

In the MATLAB doc for the ***`eig()`*** *function it states that when you specifically use this calling syntax:*

\>> ***`[V,D] = eig(A)`***,

That the returned eigenvectors ***V*** *are normalised such that the 2-norm is 1. For other calling syntaxes, this normalization scheme is NOT necessarily used.*

So?

So in this task, we'll be using the above calling syntax which gives us nomalised eigenvectors of magnitude 1 !!

Eigenvalue problems

Recall the eigenvalue problem for a square matrix $A \in R^{n \times n}$:

$$A \times \vec{v}_i = \lambda_i . \vec{v}_i \Longrightarrow det(A - \lambda_i . I) = 0$$

```
syms I_xx I_xy I_xz I_yy I_yz I_zz lambda
% define the original inertia matrix
A  = [I_xx, I_xy, I_xz;
      I_xy, I_yy, I_yz;
      I_xz, I_yz, I_zz ]
```

A =

$$\begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{pmatrix}$$

Create the eigenvalue problem: $det(A - \lambda_i . I) = 0$

```
% create the eigenvalue problem
e_mat = A - lambda*eye(3)
```

e_mat =

$$\begin{pmatrix} I_{xx} - \lambda & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} - \lambda & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} - \lambda \end{pmatrix}$$

```
det_e = det(e_mat);
det_e = collect(det_e)
```

det_e

$$= -\lambda^3 + \left(I_{xx} + I_{yy} + I_{zz}\right)\lambda^2 + \left({I_{xy}}^2 + {I_{xz}}^2 + {I_{yz}}^2 - I_{xx} I_{yy} - I_{xx} I_{zz} - I_{yy} I_{zz}\right)\lambda - I_{zz} {I_{xy}}^2 + 2 I_{xy} I_{xz} I_{yz} - I_{yy} {I_{xz}}^2 - I_{xx} {I_{yz}}^2 + I_{xx} I_{yy} I_{zz}$$

After determining ALL of the eignvalue (λ_j) and eigenvector ($\vec{v}_i$) pairs , we can collect them into matrices and write them as a single matrix equation:

$$A \times V = V \times \Lambda$$

where

$$V = \left(\vec{v}_1\ ,\vec{v}_1\ ,\vec{v}_3\ ,\ \ldots\ ,\vec{v}_n\right)$$

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & 0 & 0\ldots \\ 0 & \lambda_2 & 0 & 0\ldots \\ 0 & 0 & \ddots & 0\ldots \\ 0 & 0 & \ldots & \lambda_n \end{pmatrix}$$

So you can see that we can covert A into a diaginal matrix using the matrix of A's eigenvectors:

$$A \times V = V \times \Lambda$$

$$\Lambda = V^{-1} \times A \times V$$

Let's have a look at an example using MATLAB's **`eig()`** function:

```
% a test matrix
A = [ 1, 2, 3;
      4, 0, 6;
      7, 8, 9];
% compute the eignvectors and eigenvalues
[V, lambda] = eig(A)
```

```
V = 3x3 double

    0.2496    0.8346    0.0334
    0.4198   -0.0387   -0.8553
    0.8726   -0.5495    0.5170
lambda = 3x3 double

   14.8509         0         0
         0   -1.0680         0
         0         0   -3.7829
```

```
% look at how we can diagonalise A:
A_diagonal = inv(V) * A * V
```

```
A_diagonal = 3x3 double

   14.8509    0.0000   -0.0000
    0.0000   -1.0680   -0.0000
   -0.0000    0.0000   -3.7829
```

Eigenvalue problems: When A is Symmetric

When A is symmetric, ie: $A = A^T$, the eigenvectors corresponding to the distinct eigenvalues have a cool property - the eigenvectors are actually ORTHOGONAL, ie:

$\vec{v}_j^{\,T} \times \vec{v}_k = 0$, for $j \neq k$ and

$\vec{v}_j^{\,T} \times \vec{v}_j = |\vec{v}_j|^2$, for $j = k$

If we normalise each of these eigenvectors so that their vector norm is 1 (ie: $\vec{v}_j^{\,T} \times \vec{v}_j = 1$), then we say that the eigenvectors are ORTHONORMAL, ie:

$$V \times V^T = I$$

$$V^T = V^{-1}$$

Therefore our diagonalization formula introduced earlier, can now be written as:

$$A \times V = V \times \Lambda$$

$$\Lambda = V^{-1} \times A \times V$$

$$\Lambda = V^T \times A \times V$$

Let's look at an example:

```
% here is a symmetric matrix A
A = [   1,   -9,  -17;
       -9,    2,   45;
      -17,   45,    3; ];

% compute the eignvectors and eigenvalues
[V, lambda] = eig(A)
```

```
V = 3x3 double

    0.1357    0.9341   -0.3302
   -0.6848    0.3293    0.6501
    0.7160    0.1379    0.6844
lambda = 3x3 double

  -43.2641         0         0
         0   -4.6830         0
         0         0   53.9471
```

```
% look at how we can diagonalise A:
A_should_be_diagonal = V' * A * V
```

```
A_should_be_diagonal = 3x3 double

  -43.2641   -0.0000   -0.0000
   -0.0000   -4.6830    0.0000
   -0.0000    0.0000   53.9471
```

```
% demonstrate that V is made up of orthonormal vectors
B_should_be_identity = V * V.'
```

```
B_should_be_identity = 3x3 double

    1.0000    0.0000   -0.0000
    0.0000    1.0000    0.0000
   -0.0000    0.0000    1.0000
```

OK, so we have 3 mutually orthogonal vectors ... which is awesome, but do these 3 vectors form a Right hand rule trio of vectors? We can ensure that they do by making the 3rd vector the cross product of the first two, eg: $V = [\vec{v}_1, \ \vec{v}_2, \ (\vec{v}_1 \times \vec{v}_2)]$

```
V = bh_we_have_a_RH_frame(V);
```

```
 ... yep V gives us a RH rule frame !
```

```
V * V.'
```

```
ans = 3x3 double

    1.0000    0.0000   -0.0000
    0.0000    1.0000    0.0000
   -0.0000    0.0000    1.0000
```

Principal moments of Inertia:

OK, now for the main event. Let's look at how we now calculate the principal axes of a rigid body. Recall our well known angular momentum equation ${}^{B}L = {}^{B}I \times {}^{B}\omega$ where the frame about which we are determining the angular momentum is both BODY fixed AND at the body's centre of mass:

$$\begin{pmatrix} {}^{B}L_X \\ {}^{B}L_Y \\ {}^{B}L_Z \end{pmatrix} = \begin{pmatrix} {}^{B}I_{XX} & {}^{B}I_{XY} & {}^{B}I_{XZ} \\ {}^{B}I_{XY} & {}^{B}I_{YY} & {}^{B}I_{YZ} \\ {}^{B}I_{XZ} & {}^{B}I_{YZ} & {}^{B}I_{ZZ} \end{pmatrix} * \begin{pmatrix} {}^{B}_{G}\omega_X \\ {}^{B}_{G}\omega_Y \\ {}^{B}_{G}\omega_Z \end{pmatrix} \text{ where } \begin{matrix} {}^{B}I_{ZZ} = \int (X^2 + Y^2)\, dm \\ {}^{B}I_{XY} = \int (-X * Y)\, dm \end{matrix} \text{ etc.}$$

Note: that the B-frame inertia matrix ${}^{B}I$ is **symmetric** - *are you thinking what I'm thinking ?*

What we would like to do is to determine another co-ordinate frame, call it the *P-frame*, which is also at the body's centre of mass, but the inertia matrix about this new *P-frame* is diagonal. We call this *P-frame* the **PRINCIPAL** frame and the corresponding diagonal inertia matrix the **PRINCIPAL** moment of inertia matrix ${}^{P}I$, ie:

$${}^{P}L = {}^{P}I \times {}^{P}\omega \qquad where \qquad {}^{P}I = \begin{pmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{pmatrix}$$

To determine this new **PRINCIPAL** frame we need to find a co-ordinate transformation ${}^{P}R_B$ that converts co-ordinates in the original body *"B-frame"* into their corresponding co-ordinates in the new **PRINCIPAL**

"P-frame". If this looks/sounds familiar, it should .... because what we've described is just a PASSIVE rotation, ie: we have a FIXED *B-frame*, and we will rotate a *P-frame* relative to B. And ${}^{P}R_{B}$ is just the PASSIVE rotation matrix which relates components in one frame to another. We'll define this relationship as ${}^{P}u = {}^{P}R_{B} \times {}^{B}u$. Consider then the following:

$$ {}^{P}\omega = {}^{P}R_{B} \times {}^{B}\omega \qquad \Longrightarrow \qquad {}^{B}\omega = {}^{P}R_{B}^{T} \times {}^{P}\omega $$

$$ {}^{P}L = {}^{P}R_{B} \times {}^{B}L \qquad \Longrightarrow \qquad {}^{B}L = {}^{P}R_{B}^{T} \times {}^{P}L $$

Now let's focus on the angular momentum described in the B-frame

$$ {}^{B}L = {}^{B}I \times {}^{B}\omega $$

$$ \left({}^{P}R_{B}^{T} \times {}^{P}L\right) = {}^{B}I \times \left({}^{P}R_{B}^{T} \times {}^{P}\omega\right) $$

$$ {}^{P}L = \left({}^{P}R_{B} \times {}^{B}I \times {}^{P}R_{B}^{T}\right) \times {}^{P}\omega $$

$$ {}^{P}L = {}^{P}I \times {}^{P}\omega \qquad where \qquad {}^{P}I = \left({}^{P}R_{B} \times {}^{B}I \times {}^{P}R_{B}^{T}\right) $$

The equation for ${}^{P}I$ has the form ($\mathbf{{}^{P}I = H \times {}^{B}I \times H^{T}}$) which is similar in shape to what we saw when we discussed eigenvectors of symmetric matrices. As observed earlier, we know that ${}^{B}I$ is **symmetric** , and as such we can use the eigenvectors of ${}^{B}I$ to construct a diagnonal matrix from ${}^{B}I$ ... and that's exactly what we want to do.

$$ {}^{B}I \times \vec{v}_{i} = \lambda_{i}.\vec{v}_{i} $$

$$ {}^{B}I \times V = V \times \Lambda \qquad \Longrightarrow \qquad \Lambda = {}^{P}I = V^{-1} \times {}^{B}I \times V $$

So the 2 equations that pull everything together are:

$$ {}^{P}I = \quad \left({}^{P}R_{B} \times {}^{B}I \times {}^{P}R_{B}^{T}\right) $$
$$ {}^{P}I = \quad V^{-1} \times {}^{B}I \times V \quad = \quad V^{T} \times {}^{B}I \times V $$

So ? - So we've finally converged on some useful results:

- The **PRINCIPAL** moments of inertia are the eigenvalues λ of ${}^{B}I$.
- The orientation of the **PRINCIPAL** *P-frame* relative to the initial *B-frame*, is given by the passive rotation matrix ${}^{P}R_{B} = V^{-1} = V^{T}$, where: ${}^{P}u = {}^{P}R_{B} \times {}^{B}u$.

RECALL the one small detail: *One small detail that we need to mention is that in order to interpret $[V]$ as a rotation matrix, we need to ensure that the 3 basis vectors in $[V]$ form a right handed co-ordinate frame, ie: just because 3 unit vectors are mutually orthogonal doesn't mean they form a RH frame. So an easy way to do this is to redefine $[V]$ as:*

$$ V = \left[\vec{v}_{1}, \quad \vec{v}_{2}, \quad \left(\vec{v}_{1} \times \vec{v}_{2}\right)\right] $$

Let's look at an example:

In the previous section we established that the PRINCIPAL moments of inertia could be found by solving an eigenvalue problem. The Principal moments of Inertia ${}^{P}I$ are the eigenvalues of the original inertia matrix ${}^{B}I$. SO let's solve this eigenvalue problem. In this example we're going to :

- use MATLAB's **`eig()`** to then solve for the eigenvectors and eigenvalues

```
format longG

bIn = [
        117.81      -59.685       56.046
       -59.685       171.95       14.243
        56.046       14.243       186.33
      ];
```

Find the ${}^{P}\boldsymbol{I}$:

Now calculate the Principal inertia values using the **`eig()`** function

```
% OK: let's use the EIG() function to solve for both eigenvalues AND eigenvectors
[V, Ip_mat] = eig(bIn)
```

```
V =
        -0.797353030582144        -0.111887356741905        -0.593050894968366
        -0.458290582862403         0.751631273019369          0.47436291073283
         0.392680386932056         0.650024344790711         -0.65059239535849

Ip_mat =
          55.9036216319413                         0                         0
                         0          193.152275905814                         0
                         0                         0          227.034102462245
```

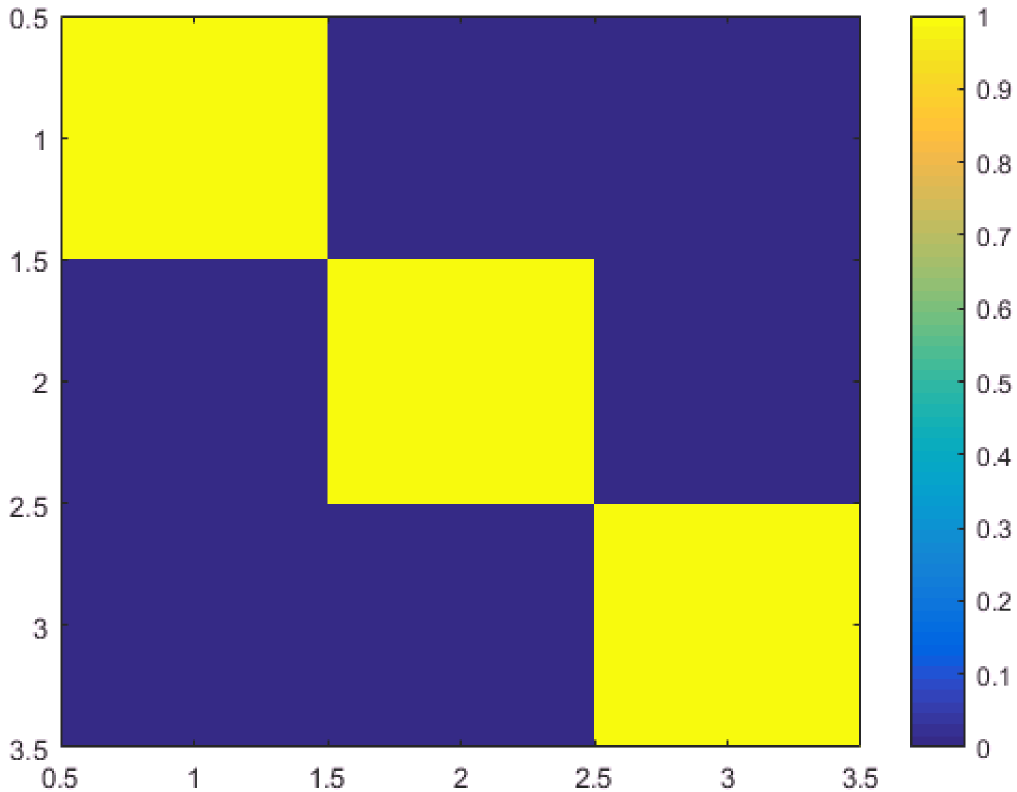
Check that we have a RH co-ordinate frame. We know that $\vec{i} \times \vec{j} = \vec{k}$

```
V = bh_we_have_a_RH_frame(V);
```

```
... yep V gives us a RH rule frame !
```

Demonstrate V is orthonormal:

```
% demonstrate that V is made up of orthonormal vectors
figure;
imagesc(V * V.');
colorbar
```



Summarise what we have so far

So we finally have our PRINCIPAL moments of inertia $({}^{P}I)$ AND we know how the PRINCIPAL axes are orientated relative to the B-frame $(V^{-1} \equiv {}^{P}R_{B})$ .. AND we've checked for a RH frame:

```
% so let's summarise what we've got
Ip_mat
```

```
Ip_mat =
      55.9036216319413                     0                     0
                     0      193.152275905814                     0
                     0                     0      227.034102462245
```

```
V
```

```
V =
    -0.797353030582144    -0.111887356741905    -0.593050894968366
    -0.458290582862403     0.751631273019369      0.47436291073283
     0.392680386932056     0.650024344790711     -0.65059239535849
```

Next step: Let's define our passive Rotation matrix $({}^{P}R_{B})$

Recall some of the formulaes mentioned earlier:

- ${}^{P}u = {}^{P}R_{B} \times {}^{B}u$
- ${}^{P}R_{B} = V^{-1} = V^{T}$

```
pRb = V.'
```

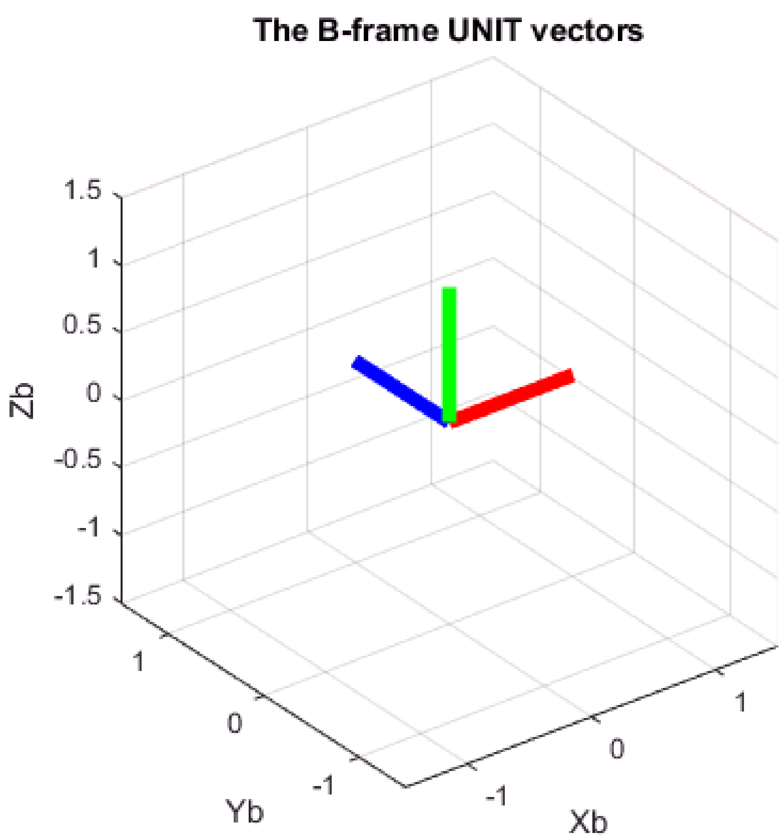
```
pRb =
        -0.797353030582144        -0.458290582862403         0.392680386932056
        -0.111887356741905         0.751631273019369         0.650024344790711
        -0.593050894968366          0.47436291073283         -0.65059239535849
```

Let's have a closer look at the passive Rotation matrix $({}^{P}R_{B})$

First I'd like to plot the unit vectors of our original B-frame:

```
figure
   % Plot the B-frame UNIT vectors
   bx_u = [1;0;0];
   by_u = [0;1;0];
   bz_u = [0;0;1];

   bh_plot_triad( gca, bx_u, by_u, bz_u );  view(3);
      title('The B-frame UNIT vectors')
```



What I'd now like to do is to plot the P-frame unit vectors, but I want to draw them in the B-frame. To compute the components of the P-frame expressed in terms of the B-frame we can just use our passive rotation matrices. Recall what our passive rotation $({}^{P}R_{B})$ matrix does:

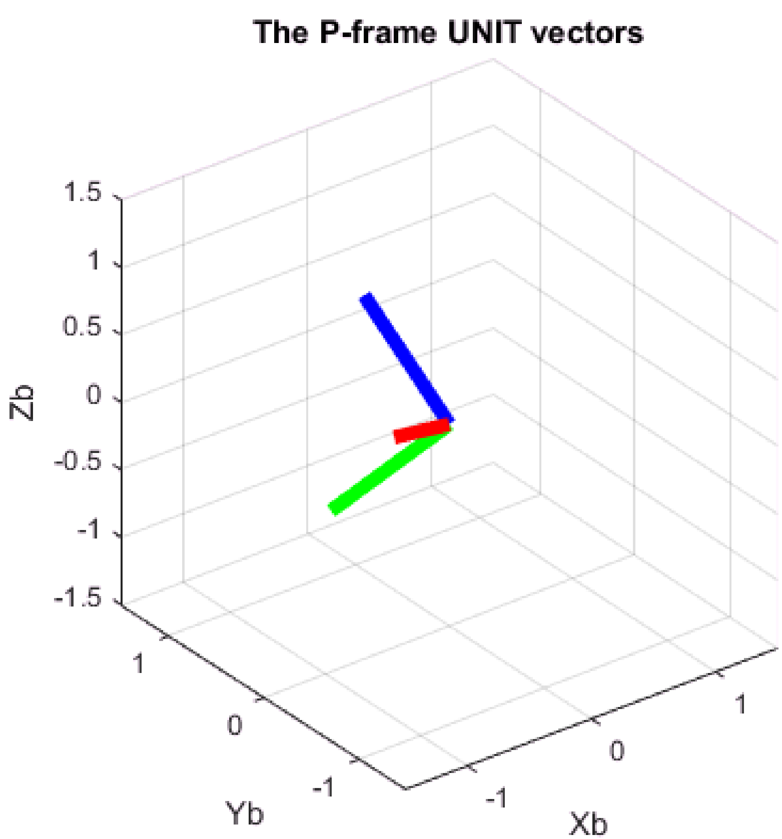
$$ {}^{P}u = {}^{P}R_{B} \times {}^{B}u $$

So to transform our P-frame unit vectors into their corresponding components in the B-frame we use the following PASSIVE rotation matrix:

$$ {}^{B}u = ({}^{P}R_{B})^{T} \times {}^{P}u $$

```
figure
   % express the unit vectors of P into their components in the B-frame
   b_xi_p = pRb.' * [1;0;0];
   b_yi_p = pRb.' * [0;1;0];
   b_zi_p = pRb.' * [0;0;1];

   bh_plot_triad( gca, b_xi_p, b_yi_p , b_zi_p );   view(3);
      title('The P-frame UNIT vectors')
```



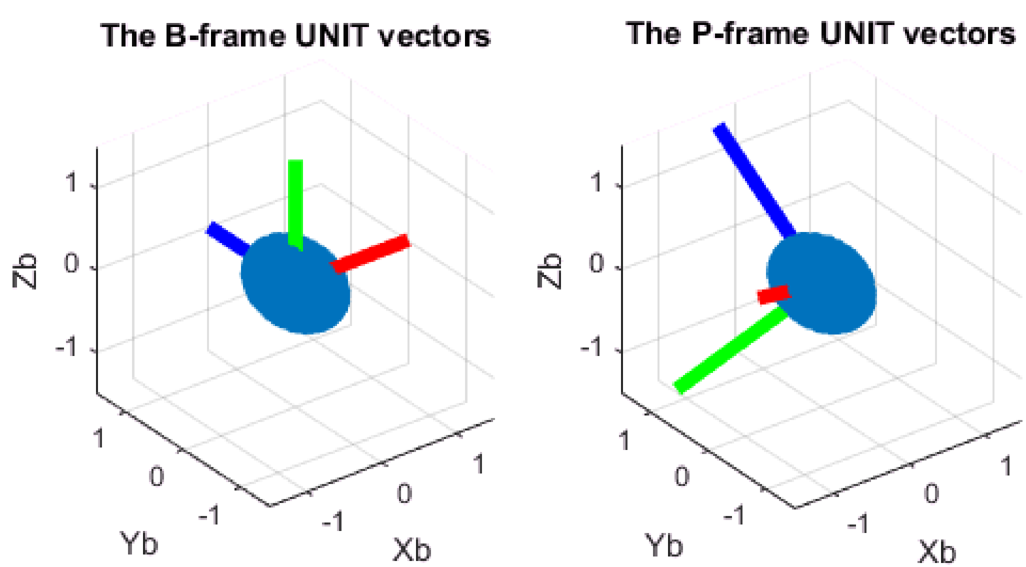
How about some more clarity ?

The Inertia matrix ${}^{B}I$ that we've just been looking at, was actually produced from a "cloud" that we've filled with tetrahedrons. We manully computed the system Inertia matrix from this "discretised" volume.

Here's what the cloud looks like ... with the original BODY axes and **PRINCIPAL** axes superimposed. It looks alot better when you interactively rotate the plots in MATLAB.

```
SRC_DATA = load('bh_saved_ellip_cloud.mat');

figure;
hax(1) = subplot(1,2,1);
        scatter3(SRC_DATA.new_x_col, SRC_DATA.new_y_col, SRC_DATA.new_z_col);
            % Plot the B-frame DOUBLE unit vectors
            bx_u = [2;0;0];
            by_u = [0;2;0];
            bz_u = [0;0;2];
            hold('on');
        bh_plot_triad( gca, bx_u, by_u, bz_u );   view(3); %view(-134,-34)
        title('The B-frame UNIT vectors')
hax(2) = subplot(1,2,2);
        scatter3(SRC_DATA.new_x_col, SRC_DATA.new_y_col, SRC_DATA.new_z_col);
            % express the DOUBLE unit vectors of P into their components in the B-frame
            b_xi_p = pRb.' * [2;0;0];
            b_yi_p = pRb.' * [0;2;0];
            b_zi_p = pRb.' * [0;0;2];
            hold('on');
        bh_plot_triad( gca, b_xi_p, b_yi_p, b_zi_p );   view(3); %view(-134,-34)
            title('The P-frame UNIT vectors')
```




```
% to see the axes rotate together you'll need to
% "evaluate selection in command window"
hlink = linkprop([hax(1),hax(2)],{'CameraPosition','CameraUpVector'});
```

Who new a story about **potatoes** could be so much **fun** ?