Solve 6-dof equations of motion:

In this script we're going to see how we can **numerically** solve the 6-DOF equations of motion for a rigid body. We'll approach this task from three different angles:

1. We'll see how to write our own ODE solver using the classic "Runge Kutta" algorithm that is typically introduced in 2nd year Numerical methods classes.
2. We'll see how to use one of MATLAB's built in ODE solvers called `ode45()`
3. We'll see how to use Simulink to solve the 6-DOF system

Recall that our 6-DOF equations of motion are composed of or 3 translational equations of motion, and 3 angular equations of motion. We represent these equations as:

$${}^{B}F = m.\left({}^{B}\dot{v} \ + {}^{B}\omega \times {}^{B}v \ \right)$$

$${}^{B}M = {}^{B}I.{}^{B}\dot{\omega} \ + {}^{B}\omega \times ({}^{B}I.{}^{B}\omega)$$

where ${}^{B}X$ means that the components of X are expressed in the vehicles body fixed frame. To convert the body rates ${}^{B}\omega$ into Euler rates we'll use the following rotation sequence:

Rotation	**START frame**	**End frame**	**Angular rate vector Associated with Rotation**
R1Z(φ)	G-frame	a-frame	$\begin{pmatrix} 0 \\ 0 \\ \varphi\_DOT \end{pmatrix}$ `in G`
R2Y(θ)	a-frame	c-frame	$\begin{pmatrix} 0 \\ \theta\_DOT \\ 0 \end{pmatrix}$ `in a`
R3X(ψ)	c-frame	B-frame (the body frame)	$\begin{pmatrix} \psi\_DOT \\ 0 \\ 0 \end{pmatrix}$ `in c`

NOTE: a slightly more descriptive and verbose nomenclature for our 6-DOF equations of motion would be the following:

$${}^{B}F = m.\left({}^{B}_{-}\dot{v}_{C} \ + {}^{B}_{G}\omega_{B} \times {}^{B}_{G}v_{C} \ \right)$$

$${}^{B}M = {}^{B}I.{}^{B}_{-}\dot{\omega}_{B} \ + {}^{B}_{G}\omega_{B} \times ({}^{B}I.{}^{B}_{G}\omega_{B})$$

where:

- ${}^{B}_{G}v_{C}$: A vector representing the vehicles ***velocity*** of the centre of mass C. The vector is expressed in components of the B-frame. The G subscript indicates that the "measurement" of the velocity is as seen by the G-frame.

- ${}^{B}\underline{\dot{v}}_{C}$ = ${}_{B}\left(\frac{d\ {}^{B}_{G}v_{C}}{dt}\right)$: the derivative of ${}^{B}_{G}v_{C}$ as seen by the B-frame, and expressed in components of the B-frame. So if we integrate ${}^{B}\underline{\dot{v}}_{C}$, then we'll get ${}^{B}_{G}v_{C}$.

- ${}^{B}_{G}\omega_{B}$: the angular ***velocity*** of the B-frame as observed by the G-frame, and expressed in components of the B-frame.

- ${}^{B}\underline{\dot{\omega}}_{B}$ = ${}_{B}\left(\frac{d\ {}^{B}_{G}\omega_{B}}{dt}\right)$: the derivative of ${}^{B}_{G}\omega_{B}$ as seen by the B-frame, and expressed in components of the B-frame. So if we integrate ${}^{B}\underline{\dot{\omega}}_{B}$, then we'll get ${}^{B}_{G}\omega_{B}$.

- ${}^{B}I$: the Inertia of the body computed about the B-frame which is attached to the body's center of mass.

- *B-frame* : the body fixed frame attached to the body's center of mass.

- G-*frame* : the inertial reference frame.

- A . B : matrix A multiplied by matrix B.

- $a \times b$: vector CROSS product.

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

In MATLAB define the ODE system to solve:

To solve our 6-DOF system:

$${}^{B}F = m.\left({}^{B}\dot{v} + {}^{B}\omega \times {}^{B}v \right)$$

$${}^{B}M = {}^{B}I.{}^{B}\dot{\omega} + {}^{B}\omega \times \left({}^{B}I.{}^{B}\omega\right)$$

we need to write a MATLAB function that defines the "state derivatives" of the system of interest. ie: I need to write a MATLAB function that represents a general system:

$$\dot{q} = f(t, q)$$

For our system we're going to define the following 12 element state vector $\mathbf{q}$ and the corresponsing 12 element vector of state derivatives $\dot{\mathbf{q}}$:

- $\mathbf{q} = \left({}^{B}v_{x} , {}^{B}v_{y} , {}^{B}v_{z} , {}^{B}\omega_{x} , {}^{B}\omega_{y} , {}^{B}\omega_{z} , \phi , \theta , \psi , {}^{G}X , {}^{G}Y , {}^{G}Z \right)$
- $\dot{\mathbf{q}} = \left({}^{B}\dot{v}_{x} , {}^{B}\dot{v}_{y} , {}^{B}\dot{v}_{z} , {}^{B}\dot{\omega}_{x} , {}^{B}\dot{\omega}_{y} , {}^{B}\dot{\omega}_{z} , \dot{\phi} , \dot{\theta} , \dot{\psi} , {}^{G}\dot{X} , {}^{G}\dot{Y} , {}^{G}\dot{Z} \right)$

I've written a MATLAB function called **`bh_the_6dof_eoms.m`** that implements this. Here are the first 36 lines of this function:

```
dbtype('bh_the_6dof_eoms', '1:36')
```

```
function qDOT = bh_the_6dof_eoms( t, q, m, I, FbMb_at_t )
    % get the excitation Forces and Moments
    FbMb = FbMb_at_t(t);
    F    = FbMb(1:3);    % (N),   Body frame, Force  vector
    M    = FbMb(4:6);    % (N.m), Body frame, Moment vector

    % extract components from the STATE vector
    vB    = q( 1:3);   % (m/s),   Body frame, translational vel
    w     = q( 4:6);   % (rad/s), Body frame, angular velocity
    e     = q( 7:9);   % (rad),   Euler angles
    xyzE  = q(10:12);  % (m),     INERTIAL frame, position

    %  F = m*(vDOT + w_x_v)
    vDOT = F/m - cross(w,vB);

    % M  = I*wDOT + w_x_(I*w)
    wDOT = inv(I) * (M - cross(w, I*w));

    % euler rates from body rates
    eDOT = LOC_get_eDOT(w, e);

    % Inertial velocity
    bRg     = LOC_get_bRg(e);
    gRb     = bRg.';
    vE      = gRb * vB;
    xyzEDOT = vE;

    % assemble the final derivative vector
    qDOT = [     vDOT;
                 wDOT;
                 eDOT;
              xyzEDOT;
            ];
end
%_##########################################################################

```

Let's start preparing for the solution - part 1

Define vehicle Mass, Inertia and Initial Conditions

```
P_veh.I = [ ...
            0.005831943165131, 0,                 0;
            0,                 0.005831943165131, 0;
            0,                 0,                 0.011188595733333;
          ]; % (kg.m^2)

P_veh.mass  = 0.9272;    % (kg)

% Vehicle INITIAL conditions
P_veh.Vb_init     = [0;0;0]; % (m/sec)    Initial velocity in BODY axes
P_veh.wb_init     = [0;0;0]; % (rad/sec)  Initial body rates
P_veh.eul_init    = [0;0;0]; % (rad)      Initial EULER angles [yaw,pitch,roll]
P_veh.Xe_init     = [0;0;0]; % (m)        Initial position in INERTIAl axes

% state vector INITIAL conditions
q_init = [ P_veh.Vb_init;
           P_veh.wb_init;
           P_veh.eul_init;
           P_veh.Xe_init   ];
```

Define Excitation Forces and Moments (in BODY frame)

We'll stimulate our system with some forces and moments. This stimuli is defined as a collection of time series data stored in an EXCEL file. Let's read this data into MATLAB and plot it:

```
TFM_TAB = readtable('bh_some_FbMb_TS_data.xlsx', 'Sheet', 'D_short_XYZ');
```

Echo the first few lines of this table:

```
TFM_TAB(1:3,:)
```

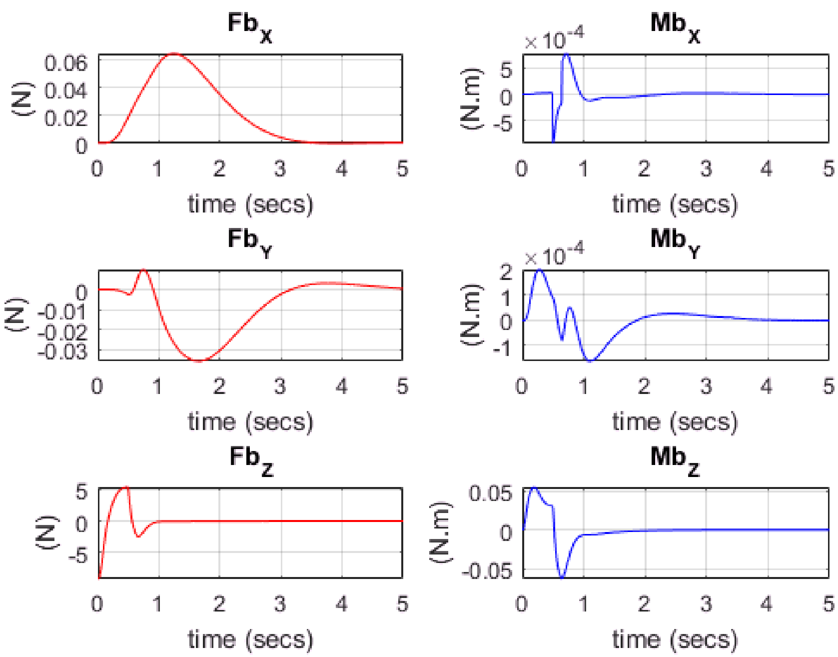
```
ans =
    Time     Fb_X     Fb_Y      Fb_Z       Mb_X     Mb_Y       Mb_Z
    ----     ----     ----     -------     ----     ----     ---------

       0     0        0        -9.0958     0        0                0
    0.01     0        0         -9.083     0        0        0.0001655
    0.02     0        0        -8.9508     0        0        0.0018196
```

There's lots of data in this log file, so just consider the first 5 seconds

```
TFM_TAB = TFM_TAB( TFM_TAB.Time <= 5, : );
```

Plot the excitation Forces and Moments that we will apply to our vehicle:

```
figure;
bh_plot_tfm(TFM_TAB);
```



Solve our ODEs using hand written ODE solver:

Let's focus on the general problem of:

$$\dot{y} = f(t, y)$$

Perhaps you want your students to explore one of the numerical methods for solving systems of ODEs. Perhaps you want them to implement their own version of the classic "Runge-Kutta" 4th oder algorithm. Recall that this RK4 algorithm looks like this (REF: *"Numerical Computing with MATLAB" - Cleve Moler*):

$$\begin{aligned}
s_1 &= f(t_n, y_n), \\
s_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}s_1\right), \\
s_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}s_2\right), \\
s_4 &= f(t_n + h, y_n + hs_3), \\
y_{n+1} &= y_n + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4), \\
t_{n+1} &= t_n + h.
\end{aligned}$$

We can implement this algorithm in MATLAB using only a few lines of code. For example here's one implmentation:

```
dbtype('bh_ode4.m', '23:31')
```

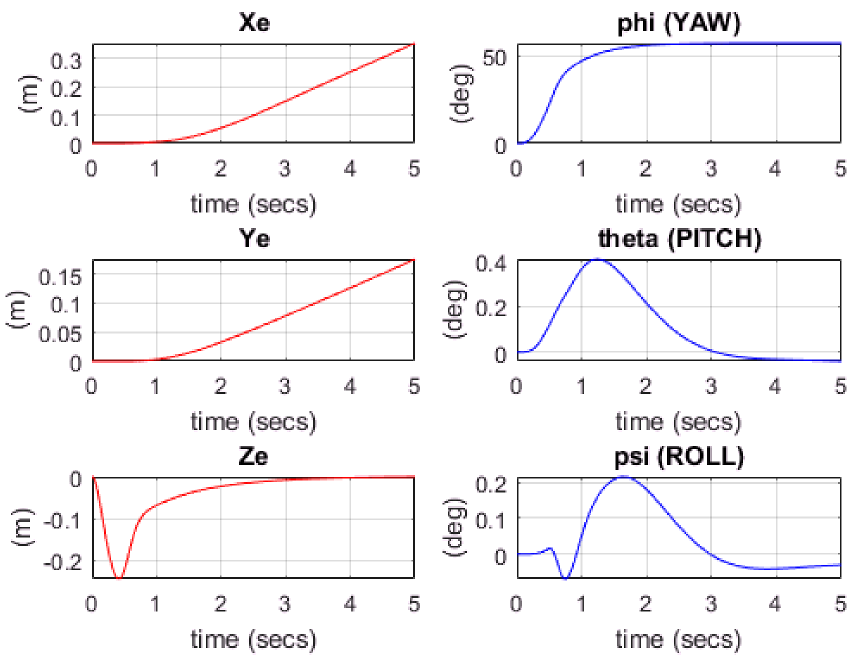
```
function y_next = LOC_rk4(f,t,y,h)
    s1 = f(t      ,  y);
    s2 = f(t + h/2,  y + h*s1/2 );
    s3 = f(t + h/2,  y + h*s2/2 );
    s4 = f(t + h  ,  y + h*s3   );

    y_next = y +  (h/6)*( s1 + 2*s2 + 2*s3 + s4);
end

```

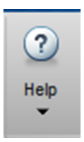
Solve our ODEs using hand written RK4 **ODE solver:**

```
% define OUR system that we want to solve
FM_at_t   = @(t)    bh_get_Fb_and_Mb_at_t( t, TFM_TAB );
dqdt_at_t = @(t, q) bh_the_6dof_eoms( t, q, P_veh.mass, P_veh.I, FM_at_t );

% Define some ODE solver settings
t_span = [0 5]; % (sec), [tstart, tend]
dt     = 0.01;    % (sec), fixed step size

% OK: let's use our ODE solver
[rk4_T,rk4_q]  = bh_ode4(dqdt_at_t, t_span, q_init, [], dt);
```

Plot the solution:

```
% plot our solution
figure;  bh_plot_6dof_solution(rk4_T,rk4_q)
```





Take a moment !

If you're interested in the broad topic of Numerically solving ODEs, don't forget how useful MATLAB's HELP browser can be. As an experiment, here's a simple text search to try with the HELP Browser:

```
doc Ordinary Differential Equations
```

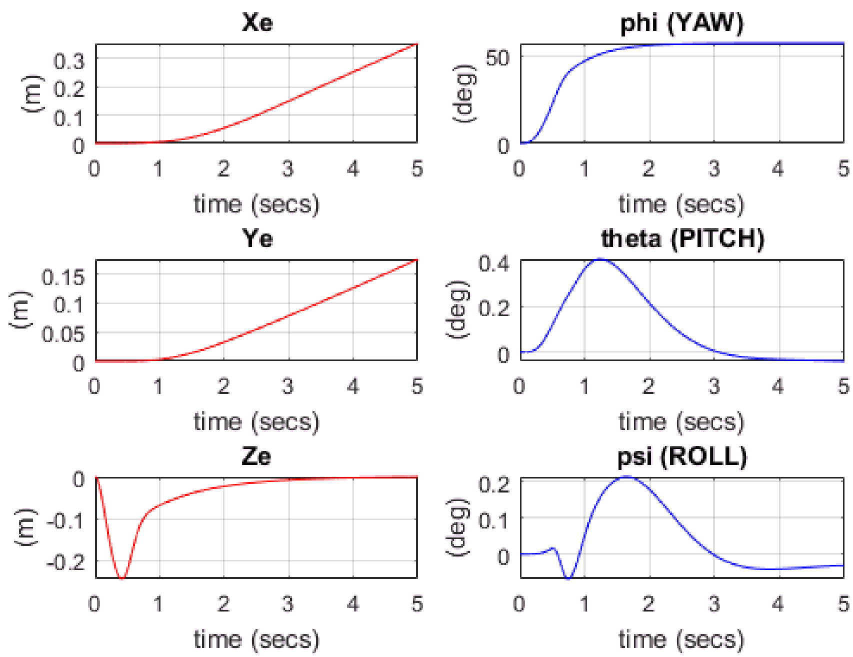
Solve our ODEs using MATLAB's `ode45()` **ODE solver:**

Perhaps you (or your students) don't want to reinvent a very round wheel. If this is your situation, then why not consider one of MATLAB's many ODE solvers. A good one to start with is the **ode45** solver.

```
% Define some ODE solver settings
t_span     = [0 5];   % (sec), [tstart, tend]
my_options = odeset('RelTol', 1e-7, 'AbsTol', 1e-7);

% OK: let's use our ODE solver
[T, Q]  = ode45(dqdt_at_t, t_span, q_init, my_options);
```

Plot the solution:

```
% plot our solution
figure;  bh_plot_6dof_solution(T, Q)
```



But .... why not use SIMULINK ?

If you're interested in solving ODEs, an alternate approach to using MATLAB is to use SIMULINK. In Simulink you express the problem using a "block diagram" language - which makes modelling systems with "feedback" very easy. In the following model, look at how we've implemented the 6-DOF equations of motion:

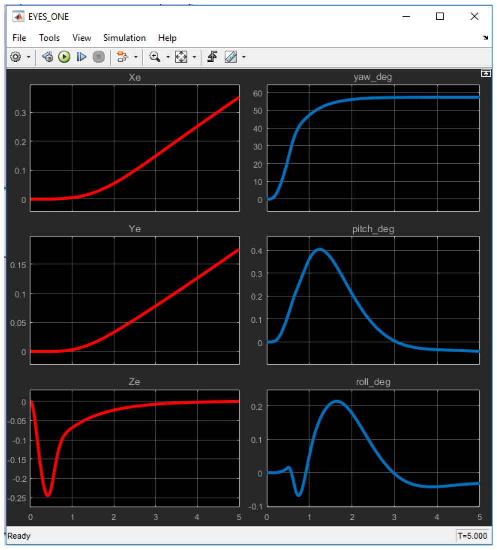
$$^{B}F = m.(\ ^{B}\dot{v}\ +\ ^{B}\omega \times {}^{B}v\ )$$

$$^{B}M = {}^{B}I.{}^{B}\dot{\omega}\ +\ ^{B}\omega \times (^{B}I.{}^{B}\omega)$$

Note that we can also define a "block's" behaviour by writing some MATLAB code - sometimes it just makes more sence to write a few lines of code. Some good examples of this are in the subsystem for computing the Direction Cosine Matrix (DCM) ... so check that out too !

```
open_system('bh_6dof')
```

When you run the Simulink model you get the following response:


EYES_ONE
File Tools View Simulation Help
Xe
yaw_deg
Ye
pitch_deg
Ze
roll_deg
Ready
T=5.000