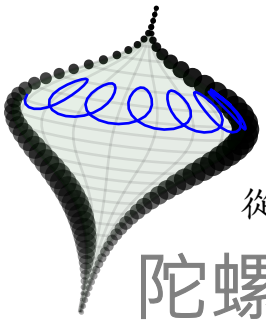


小良，台南，西元 2015 年 3 月

從剛體的轉動尤拉方程到轉動向量於姿態計算的實際應用

陀螺運動的電腦模擬

Part I
導言

當我們談到陀螺的三維運動，多數動力學的書上推導出貼體角速度的尤拉方程之後，就會求諸 Euler angles 來得到 Lagrangian，接著用 elliptical integral 解出解析解。或者，以數值方法解其尤拉角的尤拉方程 ODE，然後再以尤拉角模擬其運動。不過，若對尤拉方程透徹理解，只要有貼體角速度，也可以利用轉動向量（rotation vector），用姿態估測的方法直接簡單地數值模擬剛體轉動，這樣就不需要用到 Lagrangian 或 Euler angles。這個方法屬於姿態估測學中的方向餘弦遞推法（iteration of direction cosine matrix, DCM）。這裡將姿態（orientation 姿態 or attitude 航向，這裡統稱姿態。）動力學與姿態估測應用在剛體轉動的尤拉運動方程上，以姿態估測中的方向餘弦法來積分尤拉方程的貼體角速度轉動向量，以此達到模擬剛體的轉動，並且應用在陀螺的三維運動上。一般會覺得貼體角速度並沒有多大的用處，不過事實上在姿態動力學中，正是以正確地積分貼體角速度來得到姿態的運動。這邊不只介紹公式，還以淺顯易懂的方式給出姿態的運動方程，所以只要有基礎的線性代數矩陣知識，就可以掌握此方法。這裡詳述的這些理論在書籍上比較少見，在航太領域或電機的姿態控制領域有些講解，但也缺乏清楚的解釋，只給出複雜的公式，這樣在應用層面的時候會發生比較多不必要的試誤與嘗試。這裡我將這觀念連結錯綜複雜的東西清楚地寫下來，並且提供一個實作的例子，以供自己以及別人參考。

這裡詳述的方法可以廣泛的運用於任何剛體轉動或其尤拉方程。因此，此處所涉及的方向餘弦遞推的完整理解還可應用上其他相關領域。舉例來說，這裡在作陀螺模擬的姿態演算程式可以用在陀螺儀這類型的綁附式慣性感測器 [6, Ch 3.6.4] 上，積分測得的貼體角速度來作姿態估測，這將在第四章作介紹。仿間姿態演算法林立，但多數是沒有經過驗證的。這邊從根本的推導開始，一步步進階到實用的程式碼，每一步推導都有合理的邏輯，並且轉動矩陣都有標明轉動方向是遵守左手還是右手定則，以及標明轉動矩陣是取其主動性轉向量或是被動性轉坐標軸，這一點是所有書上或網路上的程式所共同缺乏的，而這在實際應用的時候尤其重要，這樣子推導出來的公式我們可以很方便的檢查其正確性，也可以很放心地應用在其他地方。另外，這邊提供與第三方理論及程式碼的驗證，證實我們公式及程式的正確性。這裡介紹的尤拉方程的數值化也可以應用上剛體物理模擬（simulation of rigid body），這在電腦遊戲的物理引擎 [7, Ch 2.3] 中也有其廣泛的應用性。這邊我們呈現以這些方法數值模擬三維陀螺運動。這邊提供的詳細解說也適合當作大學物理，電機，機械或航太系在剛體運動的衍伸教材。

也有其他人做過類似的陀螺模擬，舉例如 Wolfgang Christian 教授 [8]，以及 Eugene Bukitov 教授 [10]，不過都沒有我們這邊完整。比如說，Christian 教授的程式無法改變 ω_2 的參數，因此

無法觀察到波紋及正圓兩種章動進動運動。Bukitov 教授的參數只能在某區間裡以拉桿調整，而我們的程式沒有參數的限制，可以做任意運動狀態的模擬。Christian 教授使用的是四元數遞推法。四元數法與我這裡的方向餘弦法在數學上是同義的。兩位教授都沒有提供跟其他方法做比較驗證的選項，我這裡是有跟 Lagrange 法的數值解做驗證比較。我下一步也希望能跟四元數法這類的數值解能做比較。Bukitov 教授沒有公開他的演算法，因此不曉得他是用哪種方法模擬。很感謝 Christian 教授有提供 java 的 source code，不過我還在思考怎麼將我的 Python 結果跟他的 Java 結果比較。值得一提的是，他有把他的 Java 程式寫成一個獨立的 pc 端執行檔，公開在 comPADRE 網路上，此程式是互動性的，使用起來非常的方便，這也是我的目標之一。四元數及方向餘弦法的理論可以在 [9] 文獻書中找到，姿態估測演算法的深度理論可參考文章 [11]，不過裡面並沒有提供實際的模擬。

這裡的實際應用例子也提供了一個，以尤拉方程來驗證姿態估測演算法的正確性的一個實例。即以尤拉方程解析解或數值解給出的貼體角速度來做姿態估測，結果再與原本的尤拉角解析結果直接比較，這樣比較的即是姿態估測演算法的估測能力，這會在文中詳述。這邊似乎是第一個完整結合尤拉方程與姿態估測並實際應用的例子。

這篇文章的架構是：

1. 第一部分為最完整的剛體轉動運動方程 -牛頓尤拉（Newton-Euler or Euler）方程的推導證明。因為要做轉動軸的轉動向量數值積分必須要對尤拉方程有最正確的理解。這邊補充了 Goldstein Classical Mechanics[1] 中的證明觀念跳來跳去的缺失，以及大多數的書上推導尤拉方程解釋模糊的地方。轉動理論多數的講述通常過於複雜，要不就是過於簡化，缺乏與基本原理的結合，並且大部分也無提供實際實例操作這非常重要的一環，這裡則提供完整的陀螺模擬實例與完整公開的 code 供練習。

2. 接著藉由第一段尤拉方程的推導來嚴謹的證明 Euler equation 中的貼體角速度（angular velocity along body frame）可直接用於建立剛體特徵軸與 lab frame 間的主動與被動轉動矩陣 ，並以此推導出方向餘弦法的主要遞推原理來積分剛體轉動，追蹤每一時刻的剛體特徵軸在 lab frame 的位置。並以 python 程式編寫演算法，並以 python 繪圖庫 matplotlib 來作 3D 動畫。

3. 接著說明了以貼體轉動向量來近似 t 到 t+dt 時間的微小轉動時我們將以在 t+dt 時間的貼體角速度來近似，即以 $\vec{\omega}_b\,(t+dt)$ 來建立 dt 時間內的轉動矩陣 [1]。這裡也比較一般常見的 $\vec{\omega}_b\,(t)$ 方法的結果來證明此優化所帶來精度上的提升。最後將以上方法應用上陀螺的三維運動，模擬結果將與文獻 [5] 的尤拉角法做比較。

4. 最後說明如何以以上的基礎來建立三種姿態估測演算法。其中一種即為慣性角速度感測器（IMU）的姿態解算基礎。另一種為利用力矩項的姿態估測演算法，適合用於電腦姿態模擬。最後一種為傳統的尤拉角加 Lagrangian 法。我們的程式可以在同一個動畫平台上同時比較這三種方法的正確性，因此相當於是一個姿態估測能力的比較平台。

[1] The path order exponential of $\vec{\omega}_b$ from time t to t+dt. This will be discussed more in the text.

Part II
尤拉方程與姿態動力方程的推導

若你有認真思考過轉動的問題，你應該會曾經有過下面的疑問。

1. 牛頓尤拉方程17中的力矩項，明明在推導的時候是沿著固定的 space 座標取分量，為什麼在計算在應用的時候反而要沿著貼體的 body 座標取分量 ?（如 Goldstein classical mechanics 第二版書中 4-124 公式）
2. 若我隨著轉動座標系運動，那麼我觀察到此座標系的轉動角速度應該是零，因為我隨著此座標系運動，但是 body angular velocity 偏偏又不是零？到底怎麼回事？那角度感測器 angular rate sensor 量到的也不是零，那它量到的到底又是甚麼東西，我們怎麼用它？
3. 貼體角速度（body angular velocity）明明是 angular velocity 沿著 body 投影的分量，為什麼在帶入公式的時候它會等於剛體特徵軸的軸轉動角速度？body angular velocity 是不是等於轉動座標軸中觀測到的 angular velocity？
4. 與上一點非常相關，一般認為貼體角速度無太大用處，事實上在剛體運動中，貼體角速度因姿態動力學而有許多廣泛的應用性。

這邊很少有書上會去詳細解釋清楚，這裡從最根本出發一次解決你的疑惑。

首先我們必須再次證明向量變化量在不同觀測座標中的關係，即公式1，這次你會發現在應用此公式的時候有不少地方是要注意的。由於當我實際在解這問題時我發現 Goldstein classical mechanics 書中還有幾點證明在應用層面還不夠清楚，讓我在應用的時候花了好多時間去釐清，因此這邊寫上我認為可以補充書上的推導證明。首先我們從以下公式開始

$$\left(\frac{d\vec{L}}{dt}\right)_s = \left(\frac{d\vec{L}}{dt}\right)_b + \vec{\omega} \times \vec{L} \tag{1}$$

對此公式的理解將是整篇文章最重要的基礎。此公式如何而來？此公式為一隨時間變動的向量在恆定座標與非恆定座標 (此例為轉動中座標) 之間線性變換的結果。以下是此公式的推導。

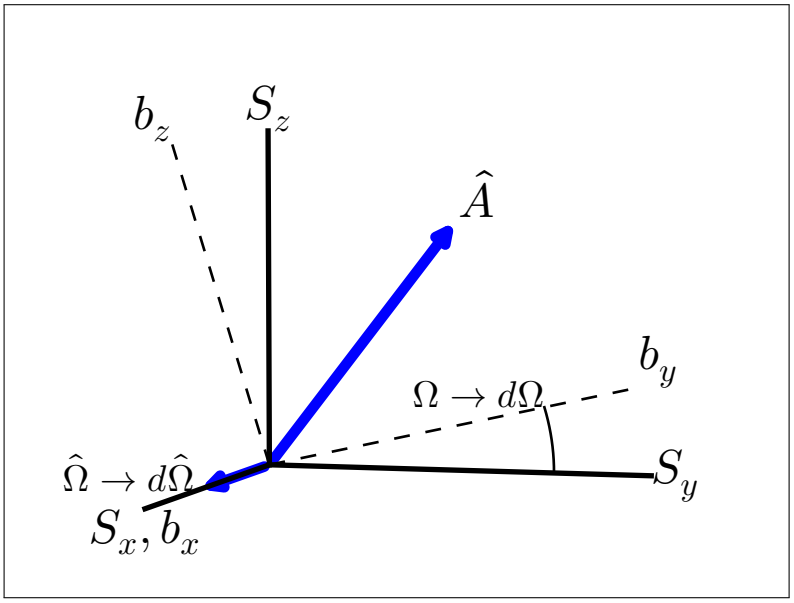
首先考慮一恆定座標 S(space)，一轉動座標 b(body)，為了方便討論矩陣轉換的主被動性與座標轉換的左右手法則，我們這邊方便的先假設 $\hat{S}_x$,$\hat{b}_x$ 兩軸重合，因此圖中顯示了 body frame 沿著 $+\hat{S}_x$ 遵守右手定則逆時針轉了 Ω 角度，依右手定則此角位移向量 $\hat{\Omega}$ 會在 $+\hat{S}_x$ 方向。但是接下來的推導以及所有公式都適用任意的座標旋轉，這邊是為了方便討論矩陣的主動被動的方向性，以及在之後的推導方便我們追蹤正負號以及矩陣主動被動意義的改變，因此在圖中做了一個方便我們思考的情形。另外，大部分書上在討論座標轉換時有時候給的公式是遵守左手定則，但這與物理定律所採納的右手定則相反，因此這邊我寫下完整的右手定則的推導，希望之後的人不需要像我一樣花了大半時間在轉換不同公式間左手右手定則帶來的正負號的改變。

依照圖1所示，我們可以寫下 $\vec{A}$ 向量在 S,b 座標間的關係

$$\left(\vec{A}\right)_b = \underset{\text{passive, r.h.}}{\Omega} \left(\vec{A}\right)_s$$

其中 Ω 是 s frame 到 b frame 的座標轉換矩陣，因為是轉換座標軸，因此矩陣取被動含意，並且我們採用右手定則，因此逆時針方向為正方向。接下來只要有用到矩陣的運算我都會標明主被

Figure 1:



動及左右手 (r.h. right-hand or l.h. left-hand)，這樣我們可以很快速對照圖表來理解轉動方向，這很重要。

若我們考慮 Ω 的角度很小 $\Omega \to d\Omega$(infinitesimal rotation)，則 $d\Omega$ 矩陣與 unity matrix 相去不遠，可以寫成 1(unity matrix) +ϵ(infinitesimal matrix)，ϵ 具有 antisymmetric matrix 的特性 [1, p. 169]，帶入上式

$$\left(\vec{A}\right)_b = \underset{\text{passive, r.h.}}{(1+\epsilon)} \left(\vec{A}\right)_s$$

infinitesimal matrix 有個特性，很容易自行驗證，

$$\underset{\text{r.h., passive or active}}{\epsilon} = \begin{bmatrix} 0 & \epsilon_3 \geq 0 & -\epsilon_2 \leq 0 \\ -\epsilon_3 & 0 & \epsilon_1 \geq 0 \\ \epsilon_2 & -\epsilon_1 & 0 \end{bmatrix}, \underset{\text{l.h., passive or active}}{\epsilon} = \begin{bmatrix} 0 & -\epsilon_3 \leq 0 & \epsilon_2 \geq 0 \\ +\epsilon_3 & 0 & -\epsilon_1 \leq 0 \\ -\epsilon_2 & \epsilon_1 & 0 \end{bmatrix}$$

現在我們考慮 $\vec{A}$ 是 $+\hat{b}_y$ 軸的狀況，不過考慮相同矩陣 $(1+\epsilon)$ 的主動特性，也就是主動轉向量，這樣的話轉動方向會與原本的方向相反，變左手定則，我們會得到

$$\left(\hat{S}_y\right)_s = \underset{\text{active, l.h.}}{(1+\epsilon)} \times \left(\hat{b}_y\right)_s$$

整理一下

$$\left(\hat{b}_y\right)_s = \underbrace{[(1+\epsilon)]^T}_{\text{active, r.h.}} \times \left(\hat{S}_y\right)_s = \underset{\text{active, r.h.}}{(1-\epsilon)} \times \left(\hat{S}_y\right)_s$$

代入上面 r.h. ϵ 的公式 (因 ϵ 還是原本的矩陣)，整理一下

$$\left(\hat{b}_y\right)_s - \left(\hat{S}_y\right)_s = -\begin{bmatrix} 0 & \epsilon_3 \geq 0 & -\epsilon_2 \leq 0 \\ -\epsilon_3 & 0 & \epsilon_1 \geq 0 \\ \epsilon_2 & -\epsilon_1 & 0 \end{bmatrix} \times \left(\hat{S}_y\right)_s$$

利用向量外積，上式也可寫成

$$\left(\hat{b}_y\right)_s - \left(\hat{S}_y\right)_s = \left(\vec{\epsilon}\right)_s \times \left(\hat{S}_y\right)_s$$

其中 $\vec{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{bmatrix}_s$ 為一向量，在 S frame 中的分量為 ϵ_1，ϵ_2，ϵ_3。

現在我們將上式跟微小轉動公式 Rodrigues rotation formula 比較

$$\vec{r}' - \vec{r} = d\vec{\Omega} \times \vec{r}$$

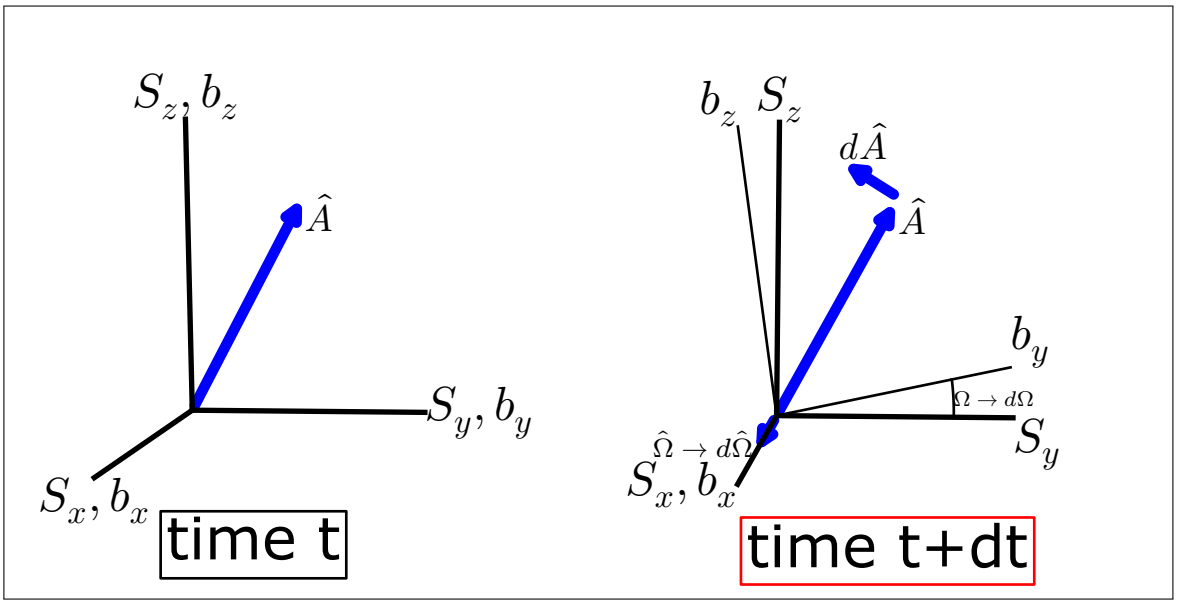
$d\vec{\Omega}$ 是 r 到 r' 的 r.h. 角位移向量，因此我們得到 $\vec{\epsilon} = d\vec{\Omega}$，$d\vec{\Omega}$ 就是 s frame 到 b frame 的角位移向量 (follow r.h. rule)

$$\left(\hat{b}_y\right)_s - \left(\hat{S}_y\right)_s = \left(d\vec{\Omega}\right)_s \times \left(\hat{S}_y\right)_s$$

這一點很重要，因為我們將證明此 $\left(d\vec{\Omega}\right)_s$ 跟接下來我們要推導的尤拉公式中的貼體角速度 $\vec{\omega}$ 有直接相關性，並且以此來做我們模擬剛體轉動的基礎。

以上的討論是考慮 $\vec{A}$ 向量不隨時間變動的情況，接下來我們必須討論 $\vec{A}$ 以及 b frame 皆隨時間變動的狀況。

Figure 2: Rate change of a vector observed in a inertial and non-inertial frame.



在時間 t 時我們令 S 與 b frame 重合，過了 dt 時間原本的 $\vec{A}$ 向量加了一改變量 $d\vec{A}$ 變成 $\vec{A}'$，並且 b frame 依右手定則轉動了一微小角度 (infinitisemal rotation)，在此前提下，向量 $\vec{A}$ 在 t 時間符合

$$\left(\vec{A}\right)_{s(t)} = \left(\vec{A}\right)_{b(t)} \tag{2}$$

接著，在 t+dt 時間 $\vec{A} + d\vec{A}$ 向量在 s 與 b frame 間的關係為

$$\left(\vec{A}'\right)_{b(t+dt)} = \underset{\text{passive, r.h.}}{\Omega} \left(\vec{A}'\right)_{s(t+dt)} \tag{3}$$

Ω 為 s, b frame 轉動矩陣 (passive r.h.)，此 Ω 矩陣與上一段 $\vec{A}$ 不變動的情況的 Ω 矩陣完全相同，我們取 s 到 b frame 的轉動為微小量，$\Omega \to d\Omega$，上式依之前所述的原理可寫成

$$\left(\vec{A}'\right)_{b(t+dt)} = \underset{\text{passive, r.h.}}{(1+\epsilon)} \left(\vec{A}'\right)_{s(t+dt)}$$

要強調這邊的 ϵ 矩陣跟之前上一段的 ϵ 矩陣是完全相同的，代入3式重新整理上式

$$\left(\vec{A}'\right)_{s(t+dt)} = \left(\vec{A}'\right)_{b(t+dt)} - \epsilon \left(\vec{A}'\right)_{s(t+dt)} \tag{4}$$

接著我們用4式減去2式，

$$\begin{aligned}
\left(\vec{A}'\right)_{s(t+dt)} - \left(\vec{A}\right)_{s(t)} &= \left(d\vec{A}\right)_s \text{, (the change in observable A in space frame)} \\
&= \left(\left(\vec{A}'\right)_{b(t+dt)} - \left(\vec{A}\right)_{b(t)}\right) - \epsilon \left(\vec{A}'\right)_{s(t+dt)} \\
&= \left(d\vec{A}\right)_{body} - \epsilon \left(\vec{A}'\right)_{s(t+dt)} \\
&= \left(d\vec{A}\right)_{body} - \epsilon \left(\vec{A} + d\vec{A}\right)_{s(t+dt)}
\end{aligned} \tag{5}$$

注意，由於 s frame 式恆定座標因此 s frame 不變動，$s(t+dt) = s(t)$。忽略高階項 $\epsilon \left(d\vec{A}\right)_{s(t+dt)}$，重新整理成

$$\left(d\vec{A}\right)_s = \left(d\vec{A}\right)_b - \underset{\text{r.h.}}{\epsilon} \left(\vec{A}\right)_{s(t)}$$

接下來我們只要記得我們的下標 b frame 總是在 t+dt 時的 frame，s frame 總是指在 t 時間的 frame，我們將不再寫出 frame 所對應的時間。依之前所述原理代入 r.h. ϵ 的公式，並且利用向量外積

$$\begin{aligned}
\left(d\vec{A}\right)_s &= \left(d\vec{A}\right)_b - \begin{bmatrix} 0 & \epsilon_3 \geq 0 & -\epsilon_2 \leq 0 \\ -\epsilon_3 & 0 & \epsilon_1 \geq 0 \\ \epsilon_2 & -\epsilon_1 & 0 \end{bmatrix} \left(\vec{A}\right)_s \\
&= \left(d\vec{A}\right)_b - \left(\vec{A}\right)_s \times \left(d\vec{\Omega}\right)_s \\
&= \left(d\vec{A}\right)_b + \left(d\vec{\Omega}\right)_s \times \left(\vec{A}\right)_s
\end{aligned} \tag{6}$$

因為這裡的 ϵ 矩陣與上一段的 ϵ 矩陣是一樣的，因此我們也可以用上之前轉動公式所推導的微小轉動矩陣 ϵ 所對應的轉動向量 $\left(d\vec{\Omega}\right)$，這樣我們就得到了 rate of change of a vector/observable in a rotating frame 公式

$$\left(d\vec{A}\right)_s = \left(d\vec{A}\right)_b + \left(d\vec{\Omega}\right)_s \times \left(\vec{A}\right)_s \tag{7}$$

我們連結了不同觀測座標觀測到的物理變化量，並且所用到都是已知的物理量 $\left(d\vec{\Omega}\right)_s$ 與 $\left(\vec{A}\right)_s$。這邊要強調，因為這裡的 ϵ 矩陣與上一段的 ϵ 矩陣是一樣的，所以證明了 $d\vec{\Omega}$ 所對應的向量就是 s frame 轉到 b frame 的角位移向量 (r.h.)，這樣強調的目的是，接下來 $d\vec{\Omega}$ 所導出的貼體角速度，就是 s frame 轉到 b frame 的角速度，因為這跟一般我們對貼體角速度的定義與認知並不一樣，這裡再次強調，我們必須考慮了 t 時間 s 與 b frame 重合，才能得到這結果。接下來會說明，也是因為如此，我們才能利用貼體角速度來作座標軸的轉動追蹤，因此此觀念至關重要。另外要注意的是 $\vec{A}$ 與 $d\vec{\Omega}$ 是沿著 t 時間的 s frame 取的投影量。這邊值得一提的是，傳統公式大多寫成

$$\left(d\vec{A}\right)_s = \left(d\vec{A}\right)_b + \left(d\vec{\Omega}\right)_b \times \left(\vec{A}\right)_b$$

為什麼這邊的 $d\vec{\Omega}$ 是沿著 body 取分量呢？注意6式中當我們將微小轉動矩陣 ϵ 寫成向量 $d\vec{\Omega}$ 時，我們並沒有侷限此向量是定義在哪一個觀測座標，因此沿著 s 或 b frame 取都是可以的。另外5式中我們忽略了高階項 $\epsilon\left(d\vec{A}\right)_{s(t+dt)}$，因此留下了 $\epsilon\left(\vec{A}\right)_{s(t+dt)} = \epsilon\left(\vec{A}\right)_{s(t)}$，但其實我們也可以取另一個近似

$$-\epsilon\left(\vec{A}+d\vec{A}\right)_{s(t+dt)} = -\epsilon\left(\vec{A}\right)_{b(t+dt)}$$

若我們考慮 s 到 b frame 的轉動非常的微小，我們只是做了一個不一樣的忽略方法。這樣的話，我們就得到傳統公式。這裡也是 Goldstein[1] 裡面的附註說明 $\left(d\vec{\Omega}\right)\times\left(\vec{A}\right)$ 沿著 s 或 b frame 取分量都是可以的，只要外積矩陣運算後出來的結果是一樣的就可以，不過他並沒有給出背後的原因。

7式取微分即得到一般常見的形式，這邊我們取沿 s frame 給出的公式，方便我們之後作數值模擬

$$\left(\frac{d\vec{A}}{dt}\right)_s = \left(\frac{d\vec{A}}{dt}\right)_b + \left(\vec{\omega}\right)_s \times \left(\vec{A}\right)_s \tag{8}$$

其中 $\left(\vec{\omega}\right)_s$ 為 $\left(\frac{d\vec{\Omega}}{dt}\right)_s$，根據我們之前對 $d\vec{\Omega}$ 的定義與強調，我們知道 $\left(\vec{\omega}\right)_s$ 即為 s frame 到 b frame 的瞬時角速度。

嚴謹的定義了 $d\vec{\Omega}$ 與 $\left(\vec{\omega}\right)_s$ 後，我們接著需要討論如何從 $\left(\vec{\omega}\right)_s$ 求回相對應的轉動矩陣，這邊你會認為，不是將 $\left(\vec{\omega}\right)_s$ 的 xyz 分量帶入之前 $1+\epsilon$ 矩陣中的 $\epsilon_1\epsilon_2\epsilon_3$ 就可以了嗎，這樣是不行的，因為從之前微小轉動的推導可以看出，$\epsilon_1\epsilon_2\epsilon_3$ 是符合特定的 antisymmetric matrix properties 的，但任意的角速度向量 $\left(\vec{\omega}\right)_s$ 可不然。這邊我們利用 Calvin Klein parameter 來近似原本的轉動矩陣（CK parameters 矩陣可從轉動公式 Rodrigues rotation formula 推導，它們數學上是同源的，參考 [1]），這邊我們給他一個代號 $CK(d\vec{\Omega})$，當然，接下來只要是矩陣運算我們都會寫上 CK 的主被動及左右手性質，方便我們與圖對照與思考[2]。

$$\underset{\text{active, r.h.}}{CK(d\vec{\Omega})} = \begin{bmatrix} a^2+b^2-c^2-d^2 & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & a^2+c^2-b^2-d^2 & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & a^2+d^2-b^2-c^2 \end{bmatrix},$$

$$\text{with } a = \cos\left(\frac{\left|d\vec{\Omega}\right|}{2}\right), \text{ b, c, d = component of } d\hat{\Omega}\cdot\sin\left(\frac{\left|d\vec{\Omega}\right|}{2}\right)$$

現在，我們一再強調 $d\vec{\Omega}$ 所對應的是 s frame 轉動到 b frame，因此我們建立的 $CK(d\vec{\Omega})$ 矩陣具有以下的特性，根據圖2我們可以寫下，

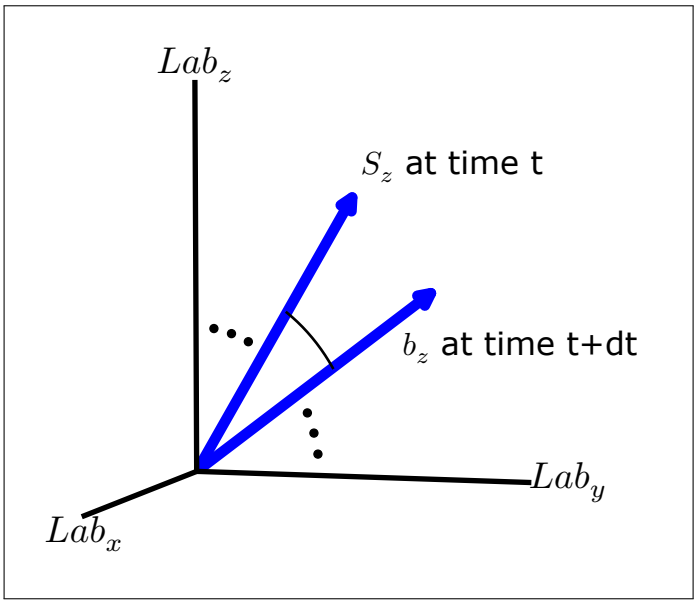
$$\left(\vec{A}\right)_s = \underset{\text{passive, l.h.}}{CK(d\vec{\Omega})}\times\left(\vec{A}\right)_b \tag{9}$$

$$\left(\hat{b}_y\right)_s = \underset{\text{active, r.h.}}{CK(d\vec{\Omega})}\times\left(\hat{S}_y\right)_s \tag{10}$$

這代表了，若我們知道 s frame 到 b frame 的轉動角度，我們就可以求出 t+dt 時間的 xyz 軸在 t 時間 xyz 軸的投影量。若我們知道的是 s 到 b frame 的瞬時角速度 $\left(\vec{\omega}\right)_s$ 則可帶入 $CK\left(\left(\vec{\omega}\right)_s\cdot dt\right)^T$ 來得到轉矩矩陣。以上兩式就是模擬或追蹤剛體的 body frame 的 x,y,z 軸轉動的基礎。

[2]物理上力矩給出的角速度是遵守右手定則 (counterclockwise，不過說順或逆時針常常會造成困擾，因順逆時針方向是軸方向正對著我們自己的時候給出的方向。)，所以 CK 矩陣必須使用其 active r.h. sense 才能描述正確向量轉動，要小心，因大部分書上 (如 Goldstein) 給的公式都是 active l.h. (舉例如書上的 Caley Klein parameter rotation matrix)，因此差一個負號，這裡我也花了許多時間把文獻上所有公式轉成了正確的右手定則。

Figure 3: How to apply rate-of-change-of-a-vector equation to numerically simulate a true rotation. Here $S_xS_yb_xb_y$ are not shown.



上述的微小轉動是只考慮 t 到 t+dt 時間內的變化，現在我們將用遞推並且 discrete 的方式，求出 body frame 在實驗者處在的靜止座標的變化。因此現在我們設定一個真正的靜止座標 Lab frame ，見圖3，此為真正的觀測者所處在的 inertial frame。考慮任意一段微小轉動 t 到 t+dt，在 t 時刻時我們將剛體的 principle axes 設定為 S frame，再將 t+dt 時刻剛體的 principle axes 設定為 b frame，這樣代表 s frame 到 b frame 就是剛體 t 到 t+dt 的轉動，並且 s 到 b frame 的瞬時角速度也是剛體轉動的瞬時角速度。我們重新將10式寫成

$$\hat{y}_s\left(t+dt\right) = \underset{\text{active, r.h.}}{CK\left(\vec{\omega}_s\left(t\right)dt\right)} \times \hat{y}_s\left(t\right) \tag{11}$$

$\hat{y}_s\left(t\right)$ 現在為 t 時間剛體特徵軸 $\hat{y}$ 在 s frame(也是 t 時刻) 的投影，這代表 $\hat{y}_s\left(t\right)$ 即為單位向量 $\left[\begin{array}{ccc} 0 & 1 & 0 \end{array}\right]$。現在我們再將上式寫成

$$\hat{y}_s\left(t_{i+1}\right) = \underset{\text{active, r.h.}}{CK\left(\vec{\omega}_s\left(t_i\right)\Delta t\right)} \times \left[\begin{array}{ccc} 0 & 1 & 0 \end{array}\right] \tag{12}$$

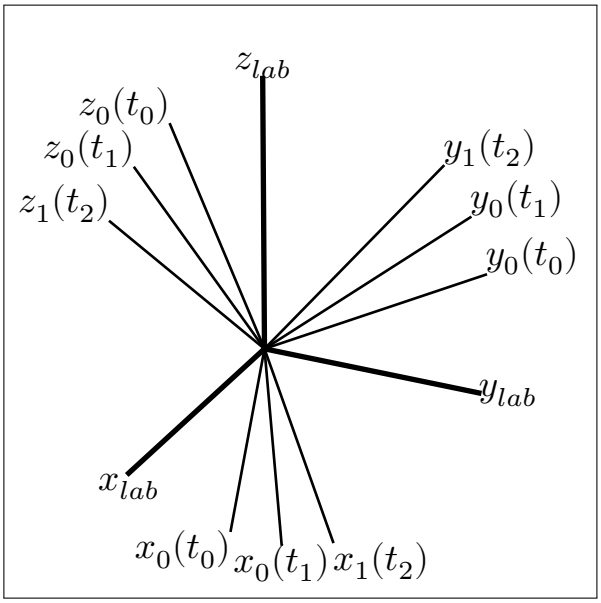
假設陀螺特徵軸在 lab frame 的起始位置已知 $\hat{x}\hat{y}\hat{z}_{lab}(t_0)$，初始貼體角速度已知 $\vec{\omega}_s(t_0)$，$\hat{x}\hat{y}\hat{z}$ 的下標代表的是觀測的 frame。首先我們將 s frame 放在 $(\hat{x}\hat{y}\hat{z}_{lab}(t_0))$，這樣依照圖3及其所述原理，b frame 的軸就是我們要求的 $\hat{x}\hat{y}\hat{z}_{lab}(t_1)$。我們先看 $\hat{z}$ 軸，12式告訴我們

$$\hat{z}_0\left(t_1\right) = \underset{\text{active, r.h.}}{\left[CK\left(\vec{\omega}_s\left(t_0\right)dt\right)\right]} \times \hat{z}_0\left(t_0\right) = \underset{\text{active, r.h.}}{\left[CK\left(\vec{\omega}_s\left(t_0\right)dt\right)\right]} \times \left[\begin{array}{ccc} 0 & 0 & 1 \end{array}\right]$$

其中 $\hat{z}_0\left(t_0\right)$，$\hat{z}_0(t_1)$ 代表時間為 t_0 與 t_1 的ẑ軸在 t_0 時間的座標軸 (也就是 s frame) 的投影，因此 $\hat{z}_0\left(t_0\right)$ 為單位向量 $\left[\begin{array}{ccc} 0 & 0 & 1 \end{array}\right]$。這樣子我們求得下一個 z 軸的位置在 t_0 的投影，不過我們得轉回 lab frame，我們假設 lab frame 的 xyz 軸到陀螺初始位置 $\hat{x}\hat{y}\hat{z}_{lab}(t_0)$ 的轉動向量是 $\vec{\Omega}_0$, 這樣我們可以用 $\vec{\Omega}_0$ 輕易的改變陀螺初始位置，運用上9式

$$\hat{z}_{lab}\left(t_1\right) = \underset{\text{passive, l.h.}}{\left[CK\left(\vec{\Omega}_0\right)\right]} \times \hat{z}_0\left(t_1\right)$$

Figure 4: Boby 軸在每一分段 t 到 t+dt 的追蹤示意圖。



注意這邊矩陣就取被動含意，結合以上兩式得到

$$\begin{aligned}\hat{z}_{lab}(t_1) &= \underset{\text{passive, l.h.}}{\left[CK\left(\vec{\Omega}_0\right)\right]}\underset{\text{active, r.h.}}{\left[CK\left(\vec{\omega}_s(t_0)\,dt\right)\right]} \times \hat{z}_0(t_0) \\ &= \underset{\text{passive, l.h.}}{\left[CK\left(\vec{\Omega}_0\right)\right]}\underset{\text{active, r.h.}}{\left[CK\left(\vec{\omega}_s(t_0)\,dt\right)\right]} \times \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

這樣我們就從 t_0 時間得到 t_1 時間陀螺 z 軸的位置。

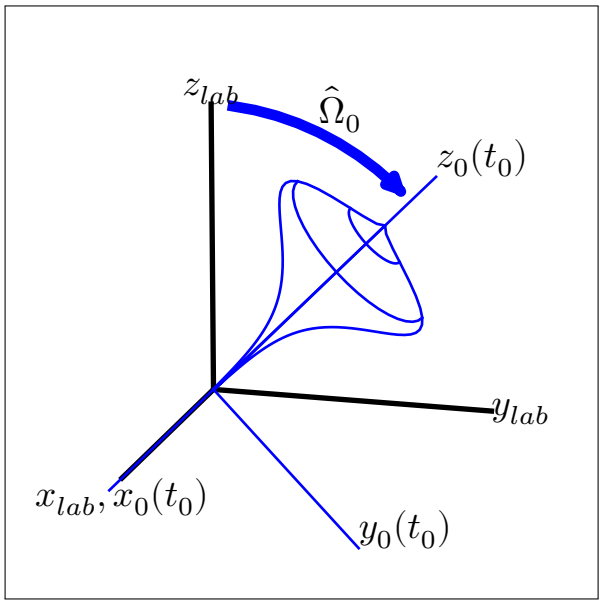
接著若我們知道 $\hat{z}_{lab}(t_i)$，再從尤拉方程數值法解出的 $\vec{\omega}_s(t_0, t_1, \cdots, t_i)$(接下來會說明)，我們同樣可以求得 $\hat{z}_{lab}(t_{i+1})$，首先用10式

$$\begin{aligned}\hat{z}_i(t_{i+1}) &= \underset{\text{active, r.h.}}{\left[CK\left(\vec{\omega}_s(t_i)\,dt\right)\right]} \times \hat{z}_i(t_i) \\ &= \underset{\text{active, r.h.}}{\left[CK\left(\vec{\omega}_s(t_i)\,dt\right)\right]} \times \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

再用9式轉回到 lab frame

$$\begin{aligned}\hat{z}_{lab}(t_{i+1}) &= \underbrace{\left[CK\left(lab \to t_i\right)\right]}_{\text{passive, l.h.}} \times \hat{z}_i(t_{i+1}) \\ &= \underbrace{\left[CK\left(\vec{\Omega}_0\right) \cdot CK\left(\vec{\omega}_s(t_0)\,dt\right) \cdot CK\left(\vec{\omega}_s(t_1)\,dt\right) \cdot \cdots \cdot CK\left(\vec{\omega}_s(t_{i-1})\,dt\right)\right]}_{\text{passive, l.h.}} \times \\ &\quad \hat{z}_i(t_{i+1}) \\ &= \underbrace{\left[CK\left(\vec{\Omega}_0\right) \cdot CK\left(\vec{\omega}_s(t_0)\,dt\right) \cdot CK\left(\vec{\omega}_s(t_1)\,dt\right) \cdot \cdots \cdot CK\left(\vec{\omega}_s(t_{i-1})\,dt\right)\right]}_{\text{passive, l.h.}} \times \\ &\quad \underbrace{\left[CK\left(\vec{\omega}_s(t_i)\,dt\right)\right]}_{\text{active, r.h.}} \times \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \qquad (13)\end{aligned}$$

Figure 5: 陀螺的初始值設定。此初始值設定對應的尤拉角初始值（以 $z-x'-z'$ 為轉軸順序分別以右手定則轉動 ϕ，θ，φ）為 $\phi_0=0$，$\theta_0=-\left|\hat{\Omega}_0\right|$，$\varphi_0=0$。在 Hasbun 教授的尤拉解法中需要此尤拉角初始值，來給出與我們方向餘弦法相同的初始條件。



這裡用上不同時間微小轉動矩陣的 commutive 性質 $(AB)C=A(BC)$。這樣我們就得到了所有時刻 $(t_0,t_1,\cdots,t_{i+1})$z 軸在 lab frame 位置的公式，同樣方法可求得 x,y 軸。可以看出上面所有 passive 的矩陣的作用只是再把坐標軸從 body frame 轉回到 lab frame。若我們再用上轉動矩陣相乘即等於轉動向量相加的事實 [1]，上式也可寫成

$$\hat{z}_{lab}(t_{i+1}) == \left[CK\left(\vec{\Omega}_0+\sum_{m=0}^{i}\vec{\omega}_s(t_m)\right)\right]\times\begin{bmatrix}0&0&1\end{bmatrix} \tag{14}$$

不過要注意，只有 CK 內包含的所有轉動向量都是要微小轉動向量此條公式才會成立。（在我們的 Python 程式中也包含有使用此公式的選項，讀者可以自由切換比較。用此公式的好處是，矩陣的相乘轉換成了矩陣的相加，在速度及記憶體的佔用上此近似會有優勢。不過，程式預設是使用13公式，因 $\vec{\Omega}_0$ 量值不小。）另外這邊我們就省略了矩陣的主被動左右手的標示，但注意，我們的 CK 矩陣是定義右手定則，遵守物理定律。這裡若不強調會產生問題，因為很多書上給的是 CK 是左手公式。並且，若要了解公式，主被動及左右手的標示是必須保留並且對照圖表才有辦法了解公式如何而來以及轉動方向是如何，而這也是大部分書上所缺乏的。這樣，我們就得到了剛體特徵軸的追蹤公式，並且只要知道" 貼體" 角速度（注意並不是角速度在 lab frame 的觀察值，當然如果知道這個，我就不用寫這麼多了。）就可以求得。同理可得 $\hat{x}_{lab}(t_{i+1})$，$\hat{y}_{lab}(t_{i+1})$。

若將 $\hat{x}_{lab}(t_{i+1})$，$\hat{y}_{lab}(t_{i+1})$，$\hat{z}_{lab}(t_{i+1})$ 的三個行組成一矩陣 C，我們會發現14式即為導航書籍上常看到的方向餘弦遞推理論的姿態矩陣動力方程 [6]

$$C(t_{i+1}) = C(t_i)\exp(\int\omega dt) \tag{15}$$

$$\dot{C}(t_{i+1}) = C(t_i)\times\omega(t) \tag{16}$$

不過似乎沒有人願意花時間把此公式解釋清楚，比如說，13式若不經過我們每一步轉動矩陣都紀錄並寫下主被動與左右手性質，實際應用的時候我們將不知道是否要套用 transpose 矩陣而胡亂套用 (很常發生)，轉動向量的主動還是坐標系的被動特性以及轉動方向遵守是左手還是右手定則，這樣將造成實際應用時極大的困擾。因此我們知道，只給出15公式只是一小步，離實作層面還有非常大一段距離。此式中 $\exp(\int \omega dt)$ 為 path order exponential，關於此，以及要如何證明 $\exp(\int \omega dt)$ 是我們的 CK 矩陣，可以在 [2, Page 49] 找到，並注意連結與 $\sin$，$\cos$ 的泰勒展開的關係。

以上為轉動坐標系中的向量變化量與恆定坐標系中的向量變化量與轉動坐標系的角速度的關係推導，不過我們發現我們必須要知道所有時刻的貼體角速度 $\vec{\omega}_s(t_0 \cdots t_i)$。若是在慣性感測器的應用，由於 strap-down（捷聯式 or 綁附式）慣性感測器（IMU, inertial measurement unit）測量得的就是貼體角速度，因此我們可以直接將測量值帶入我們的公式，這就是捷聯式慣性感測器姿態估測演算法的一種演算法，也就是我們程式的 C 方法，我們會在第四部分說明。另一種方法是，我們也可以利用尤拉方程以遞推的方式數值求出 $\vec{\omega}_s(t_0 \cdots t_i)$，並且每遞推一次就以當前的 $\vec{\omega}_s$ 更新陀螺的姿態，然後再以尤拉方程去遞推下一個 $\vec{\omega}_s$，再更新陀螺的姿態，反覆遞推就會得到所有時刻的運動，此法為 A 方法，接下來我們將詳細說明此 A 方法。

將8式應用上一段 s 到 b frame，t 到 t+dt 的微小轉動，並且考慮 $\vec{A}$ 為剛體角動量 $\vec{L}$，則我們得到

$$(\Gamma)_s = \left(\frac{d\vec{L}}{dt}\right)_s = \left(\frac{d\vec{L}}{dt}\right)_b + (\vec{\omega})_s \times \left(\vec{L}\right)_s \tag{17}$$

這裡第一等號也用上牛頓定律。現在我們從7式知道 $\left(d\vec{\Omega}\right)_s$ 是 s 到 b frame 的角位移，而經由我們之前剛體特徵軸的設定，s 到 b frame 正是我們剛體特徵軸從 t 到 t+dt 的角位移，因此 $\left(\frac{d\vec{\Omega}}{dt}\right)_s = (\vec{\omega})_s$ 正是剛體的瞬時角速度（沿著 t 時間 s frame 取分量），接著，因為 s, b frame 都是沿著 body principle axes 而取，因此沿 s frame 的角動量 $\left(\vec{L}\right)_s$ 可以寫成

$$\left(\vec{L}\right)_s = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \times (\vec{\omega})_s$$

並且沿著 b frame 的 $\left(\frac{d\vec{L}}{dt}\right)_b$ 項也可寫成

$$\left(\frac{d\vec{L}}{dt}\right)_b = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \times \frac{d(\vec{\omega})_b}{dt}$$

其中 I_{xx}、I_{yy}、I_{zz} 都不隨時間變動。再次注意 $(\Gamma)_s$ 與 $(\vec{\omega})_s$ 與 $\left(\vec{L}\right)_s$ 都是沿著 t 時刻的剛體特徵軸 (也就是 s frame) 取的投影，並不是 Lab frame 的投影，這點要特別注意，基本上這代表，$(\Gamma)_s$ 是貼體的力矩，這裡也給出了文章開頭問題的解答！這裡大部分的書上都沒有給出恰當的原因，就連 Goldstein 在書中也只有說向量沿著任意座標取分量都可以，不過這並無法合理解釋，事實上是，我們假設了 t 時間 s 與 b frame 重和（Goldsten 的證明也是採用此假設 [1]，我這裡只是將 Goldstein 上面的證明用淺顯的線性代數重新表達。），見2式，因此 s frame 是隨時間一直變動。如果用 lab frame 的 Γ 那麼就無法成功的數值化喔[3]。這邊我們證明了17式最後那一項

[3]注意因為 s frame 會持續的改變所以 $\left(\vec{\Gamma}\right)_s$ 不可取 $\left(\vec{\Gamma}\right)_{lab}$ 的值，同理 $(\vec{\omega})_s$ 也不是 $(\vec{\omega})_{lab}$，兩者都必須經過轉換從 lab 轉到 t 時刻 s frame。

中的兩個 $\vec{\omega}$ 是相同的[4]。$\frac{d(\vec{\omega})_b}{dt}$ 即是 $\frac{d\vec{\omega}_b(t+dt)-d\vec{\omega}_s(t)}{dt}$，由於 s 與 b frame 都是剛體特徵軸，都是貼體座標軸，只是在不同時間，以此展開17式，我們就得到所謂的牛頓尤拉公式 (Newton-Euler's equation)，或簡稱尤拉公式。

$$\begin{aligned}\Gamma_x(t) &= I_x\dot{\omega}_x + (I_z - I_y)\omega_y(t)\,\omega_z(t)\\ \Gamma_y(t) &= I_y\dot{\omega}_y + (I_x - I_z)\omega_x\omega_z\\ \Gamma_z(t) &= I_z\dot{\omega}_z + (I_x - I_y)\omega_x\omega_y\end{aligned} \tag{18}$$

其中 $\vec{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}$。注意 $\vec{\Gamma}$ 及 $\vec{\omega}$ 的 x,y,z 分量都是沿著 t 時刻的剛體特徵軸 s frame 取的分量，這點必須要再次強調。之後數值模擬的時候這點是非常重要的。基本上可以說，尤拉方程是我們假設2式後所作的一種線性代數變換近似。

接下來應用上陀螺，若考慮陀螺的條件 $I_x = I_y \neq I_z$，18式可寫成

$$\frac{d}{dt}\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} = \begin{bmatrix} 0 & -\frac{I_z-I_y}{I_x}\omega_z & 0\\ -\frac{I_x-I_z}{I_y}\omega_z & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} + \begin{bmatrix}\frac{\Gamma_x}{I_x}\\ \frac{\Gamma_y}{I_y}\\ \frac{\Gamma_z}{I_z}\end{bmatrix} \tag{19}$$

如之前所強調，右邊所有項都是在時間為 t 時刻的 s frame 取得值，也因此以上的微分方程組可以用普通數值由拉法或四階 Ruge Kutta 求出左側 $\vec{\omega}_b(t+dt)$，也就是從 $\vec{\omega}_s(t)$ 求得 $\vec{\omega}_b(t+dt)$，而 $\vec{\omega}_b(t+dt)$ 就是下一時間的 $\vec{\omega}_s(t)$。不過對於任意的剛體轉動系統，只要能從18式右側 $\vec{\omega}_s(t)$ 求得左側 $\vec{\omega}_b(t+dt)$，都還是能夠適用接下來的模擬方法，也因此這裡描述的方法是具有任意一般性的，可以應用在任何的剛體轉動，包括 asymmetric rotor 的情況。有不少的數值方法可以解一般的非線性一階 ODE 尤拉方程 [3]。這裡我以 Ruge Kutta 四階法求解上式，來得到 $\vec{\omega}_s\,(t_0 \cdots t_i)$，並且寫成 python 程式，程式將在下一章介紹。這裡所使用的方法與原子分子運動模擬中的方法雷同，可以參考 [9]。

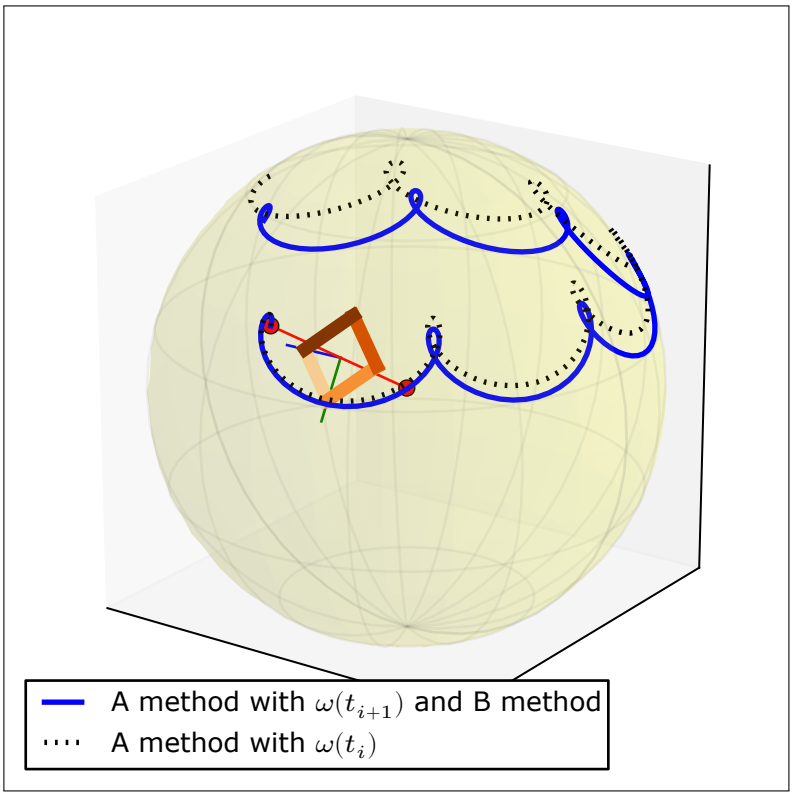
1 以 t_{i+1} 時間的轉動向量來近似 t_i 到 t_{i+1} 的轉動

以上11式中以 $CK\,(\vec{\omega}_s(t_i)dt)$ 來近似 t_i 到 t_{i+1} 的轉動事實上還不夠好，圖6顯示，以 $\omega(t_i)$ 轉動向量近似的結果與 Hasbun 的正確結果偏差不少。這邊我提出以 $CK\,(\vec{\omega}_s(t_{i+1})dt)$ 來近似 t_i 到 t_{i+1} 的轉動，圖6顯示模擬結果幾乎與 Hasbun 的正確解重合，至少此圖表上無法看出任何差別。為什麼有如此大的影響，以下我也嘗試提供物理解釋。這裡我們暫時假設 $\vec{\omega}_s(t_i)dt = \vec{\Omega}_s(t_i)$，我們知道轉動向量在 t_{i+1} 跟 t_i 時刻在 body frame 中的向量值一般不會一樣，也就是 $\vec{\Omega}_{i+1}(t_{i+1}) \neq \vec{\Omega}_i(t_i)$，這代表從 t_i 到 t_{i+1} 時，轉動向量在 body 座標上有變化，也因此我們不能夠單只考慮陀螺轉了 $\vec{\Omega}_s(t_i)$ 而已，此額外轉動向量的變化在 t_i 時 s frame 的向量值為 $\Omega_{i+1}(t_{i+1}) - \Omega_i(t_i) = \Omega_i(t_i) + d\Omega_i(dt) - \Omega_i(t_i) = d\Omega_s(dt)$，也是一個轉動向量，所以 space 空間中總共的轉動可以考慮成兩步，第一步轉 $\Omega_s(t_i)$，第二步轉 $d\Omega_s(dt)$，寫成轉動矩陣

$$CK(\Omega_s(t_i)) \times CK(d\Omega_s(dt)) = CK(\Omega_s(t_i) + d\Omega_s(dt)) = CK(\Omega_{i+1}(t_{i+1})) \tag{20}$$

[4]但我們必須強調，任意情況下，角速度 ($\vec{\omega}$) 在 body 轉動座標下的投影並不是 body 座標上觀察到的角速度！這是很常見的錯誤，這裡我們是有條件的考慮 t 到 t+dt 時刻的 t 時刻 s,b 座標重和。

Figure 6: 陀螺的對稱軸的單位向量的模擬軌跡圖。以 $\omega(t_i)$ 或 $\omega(t_{i+1})$ 轉動向量來近似 t_i 到 t_{i+1} 時間的轉動的模擬結果，並與 Hasbun 教授的數值解 B 法比較。$\omega(t_{i+1})$ 的結果與 Hasbun 教授的結果在此圖形中幾乎重疊，這代表以 $\omega(t_{i+1})$ 來近似比 $\omega(t_i)$ 好很多。



這代表我們只要考慮陀螺從 t 到 t+dt 的時候是轉了 $\Omega_s(t+dt)$ 而不只是 $\Omega_s(t)$，因此考慮 $\Omega_s(t+dt)$ 我們就更準確的近似了這個轉動，以下的 Python 程式模擬會證明，考慮了 $\Omega_s(t+dt)$ 給出的結果比 $\Omega_s(t)$ 好非常多。若如此考慮則13式變成

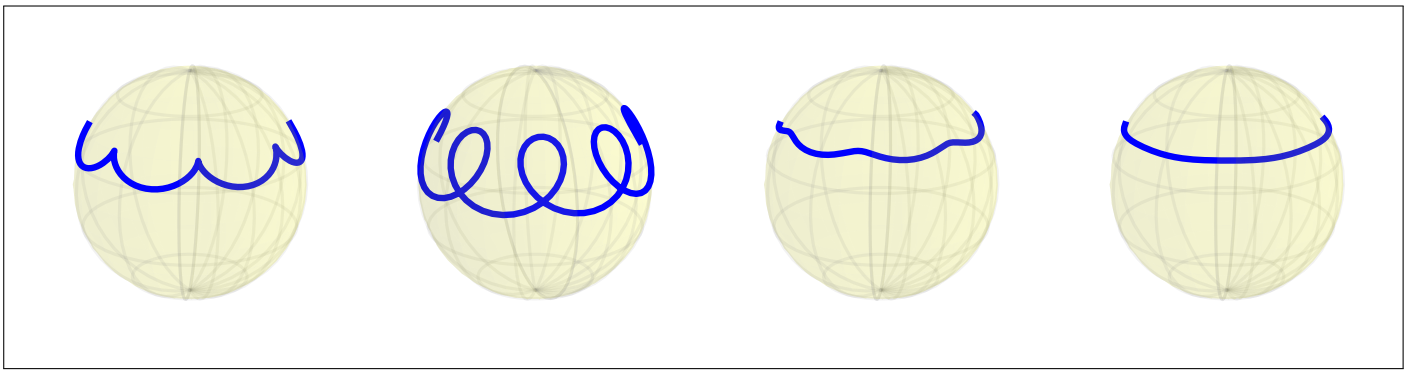
$$\hat{z}_{lab}\left(t_{i+1}\right) = \left[CK\left(\vec{\Omega}_0\right)\cdot CK\left(\vec{\omega}_s\left(t_1\right)dt\right)\cdot CK\left(\vec{\omega}_s\left(t_2\right)dt\right)\cdot\cdots\cdot CK\left(\vec{\omega}_s\left(t_{i+1}\right)dt\right)\right]\times \begin{bmatrix}0 & 0 & 1\end{bmatrix} \tag{21}$$

此公式即為我 Python 程式中使用的 DCM 遞推的公式。相同方法可求得另外兩軸 x,y 的運動。

使用 $\omega(t+dt)$ 來做 t 時間的轉動近似讓我們的程式碼變得非常簡單，比書中 [6, Page 301, Equation 10.24] 給出的轉動向量近似公式簡單太多了。簡單的程式碼代表之後在做速度優化，或轉寫成速度快的編譯式語言上，會簡單非常多。若需要更精確的近似，書 [6, Page 301, Equation 10.24] 中也給出轉動向量的近似公式，不過此公式頗為複雜，因此這邊先嘗試比較以我們 $\vec{\omega}_s(t_{i+1})$ 方法與解析解的不同。下面就將上述方法寫成 python 程式，以尤拉方程數值解出貼體角速度，接著用遞推公式21畫圖模擬其 xyz 軸運動。這邊劃出四種陀螺的經典運動，其模擬參數請參考下一章節的參數預設值。程式產生的 3D 動畫的演示請參考以下連結。

3D animation: http://tinypic.com/r/wk20ch/8

Figure 7: 左到右: (a) 尖點 (b) 環狀 (c) 波紋 (d) 正圓運動。



Part III
方向餘弦演算法 Python 程式碼之說明

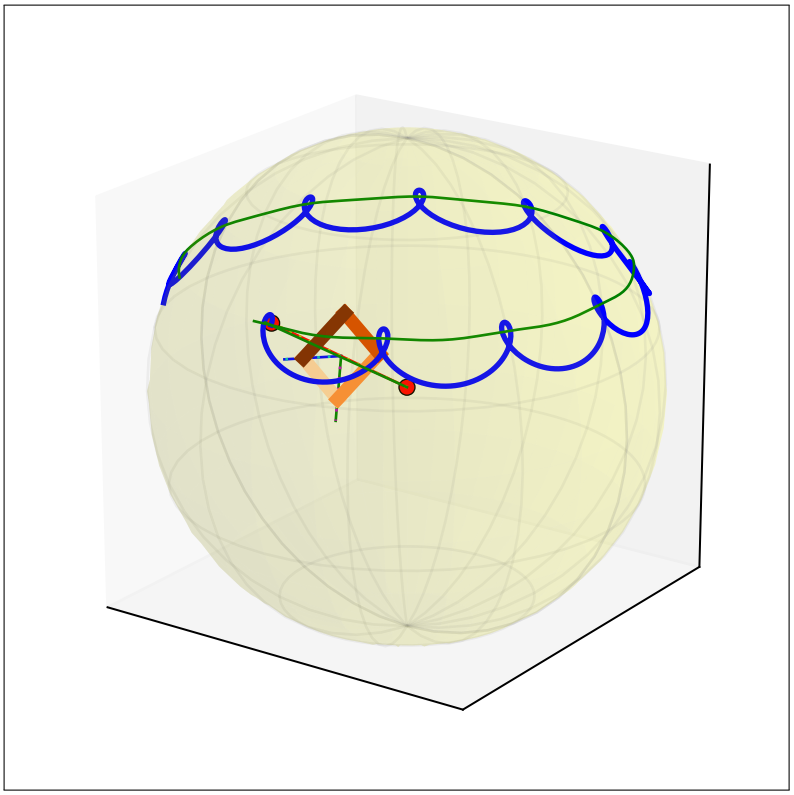
程式經過幾次升級修改後，目前已經以物件導向方式編寫完成。主要是環繞著一個類別〔class〕名為 RigidBodyObject 來發展。此類別的宣告及其方法〔演算法的主程式〕被包在一個套件 module 裡，此套件為 RGCordTransVXX.py 檔，使用前先 import RGCordTransVXX，XX 代表目前的版本。動畫及圖表產生的程式是整合在 RBPlotFuncVXX.py 此 module 檔中，因此也要 import 此 module。我使用的 Python 版本為 Python 2.7.5 - Anaconda 1.8.0 (32-bit)，NumPy 1.7.1，SciPy 0.13.0，Matplotlib 1.3.1。不同版本的 Python〔如 Python 3.x〕會有一些指令的不同可能要做小修改，可以聯絡我我可以幫忙做修正。使用者可可調整的參數列在文件最後的程式使用說明書中。說明書名稱是 GyroSoft Simulation Documentation。以下為如何使用程式的一個最簡單範例，以及其輸出的陀螺動畫，我們將此範例存成一個 Gyroscope-Demo.py 檔，使用的時候只要執行此檔就可以，裡面的內容如下:

```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as p3
import matplotlib.animation as animation
import RBPlotFunctionV3 as RBPlot
import RGCordTransV13 as RG

b = RG.RigidBodyObject()
#set para here
b.freq= 20
b.GenerateDependentVariables()
#set case here(after dep var is created)
b.setcase(1)
b.EulerDCMiter()
b.DrawOption['A_axes'] = True
b.DrawOption['A_square'] = True
b.DrawOption['A_z_axis_trace'] = True
b.DrawOption['A_Angular Momentum Trace'] = True
b.DrawOption['A_Angular Momentum Vec'] = True
b.DrawOption['A_Angular Velocity Vec'] = True
b.framerate = 20

fig2 = plt.figure(2,figsize=(7,7))
ax4 = p3.Axes3D(fig2)
RBPlot.AnimationFrameSetUp(b,ax4,plt)

line_ani = animation.FuncAnimation(fig2, RBPlot.update_line_new,
 list(range(1,b.N,b.framerate)), fargs=b.linesarg,interval=100,
 blit=False,repeat=True)

plt.show()
```

模擬程式畫出剛體特徵軸的 x, y, z 軸單位向量，分別為藍紅綠線段，我們以一個正方形來代表剛體，正方形的中心點定在 z 軸的二分之一處，由於 Python 的 Matplotlib 三維繪圖程式庫

mplot3d 並沒有太多的 3D 繪圖支援，因此這邊只是以一個簡單的示意的正方形來代表陀螺，之後我也考慮將模擬結果輸出至專門做 3D rendering 的繪圖程式語言，如 openGL，來做更精美的陀螺外觀。z 軸單位向量的頂點對時間的軌跡圖，也就是 locus，是以藍色的曲線表示。Hasbun 教授的尤拉角法給出的 z 軸單位向量的 locus，是以黑色的曲線表示。另外剛體的總角動量的單位向量以黑色線段表示，剛體的角速度除以角速度初始值量值的 normalized 向量以綠色線段表示。

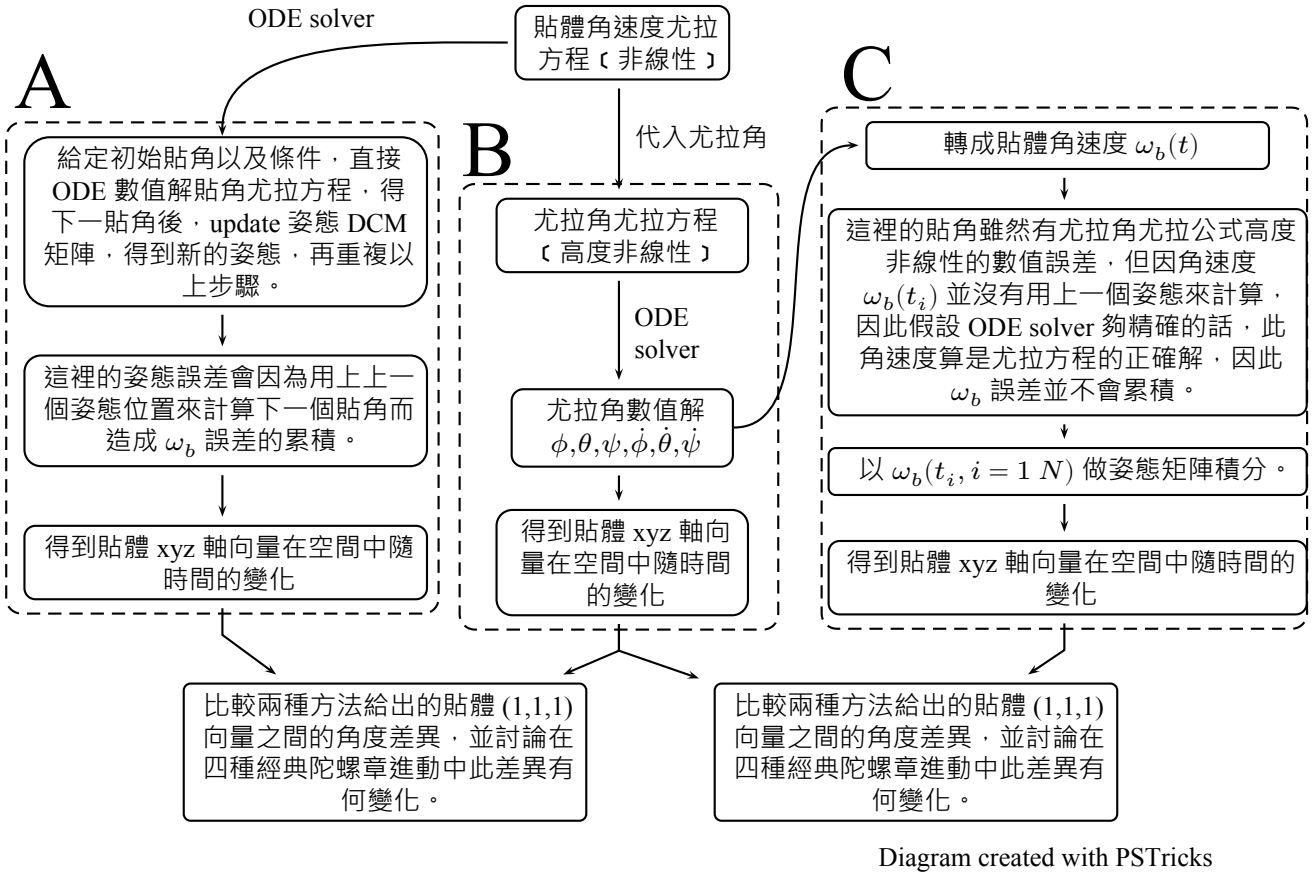
Figure 8: 模擬條件 (SI units): I=0.002; Is=0.0008; g=9.8; M=1 ; arm=.04; spin freq= 20 Hz; Initial angle from vertical 54.57 degree; Sampling rate 2000 Hz.



如之前所提到，程式有三種不同的模擬陀螺運動的方法，可以互相比較互相驗證。這三種方法各自有其應用領域，我們會詳細解釋。這三種方法的流程圖我們整理在圖9中，這樣可以幫助我們清楚的了解其差異性。

首先 A 方法為，數值解轉動座標的貼體角速度的尤拉方程（牛頓尤拉方程），從初始值得到下一個貼體角速度，並以此貼體角速度依照第一章節推導的方向餘弦遞推公式21求得下一個姿態，再以此姿態回到貼體尤拉方程解出下一個貼角，也就是重複以上步驟。這樣可以得到剛體 xyz 軸的運動（這邊我們是討論應用於陀螺的運動）。此方法我們姑且稱為力矩已知的貼體尤拉方程的姿態遞推，因為每一步遞推解貼體角速度時都需要知道貼體尤拉方程中的力矩項（這裡也呼應第一章我們所強調' 貼體' 力矩的證明是有其重要性），而陀螺的力矩是我們已知的重力而來。由於此 A 方法是一種轉動向量的堆疊近似，因此會有誤差，而我們想要知道這誤差有多大，因此我們會跟以下的 B 方法 -也就是傳統的數值解尤拉角尤拉方程（代入尤拉角後的牛頓尤拉方程）做比較。A 方法以四階 Runge Kutta 直接數值解陀螺尤拉方程（topEOM(wi,torquei)）

Figure 9: Python 程式所包含的 ABC 三種姿態解算法示意圖。



得到貼體角速度，接著用遞推公式13來得到陀螺特徵軸 xyz 軸的單位向量 cordvec 對時間的變化。cordvec 陣列的維度 shape=(N,3,7)。其中維度為 7 的方向的首要三個陣列就是 xyz 軸。DCM 遞推的程式在 DCMiter 方程裡。

B 方法，數值解尤拉角尤拉方程，就是將牛頓尤拉方程中的貼體角速度代換成尤拉角，然後進行 ODE 數值解。由於牛頓尤拉方程本身就呈現非線性，代入尤拉角後得到的微分方程組更是呈現高度非線性，因此取樣時間需要縮短來防止數值解發散。並且，解尤拉角尤拉方程組在 $\theta = 0^\circ$ 時因為會除以 $\sin(\theta)$，因此在此角度附近會發散產生 NaN 的值。因此使用 B 方法時必須要遠離這個限制。不過，若 ODE 的取樣時間夠小（當計算量大，計算時間長並不是問題時）或誤差有作控制，此方法算是一種較精確的數值法，也因此我們將比較 AB 兩種方法給出的結果。

值得一提的是，B 方法與使用 Lagrange 力學所得到的方程是可以 reduce 成一樣的微分方程組，所以基本上也可以說我們的 B 方法也是 Lagrange 法，關於如何證明兩個方法是一樣的，只要查詢 Goldstein 書上 [1, Page 216] 的 Equation 5-62 與 Eberly 書上 [12, Page 158] 的 Equation 3.50 最終得到的是一樣的微分方程組即可得知。

B 方法的程式碼來自 Hasbun 教授的著作 [5]，不過他的程式碼是 Matlab 寫成，因此為了能跟我的 python 程式碼比較，我將他的程式重寫成 python 程式碼，此方程叫做 HasbunEulerEquationODEsolve，列在最後的程式說明書中。補充一提，Hasbun 教授所採用的尤拉角定義與我的

相同，都是依照 z-x'-z' 軸依右手定則轉 $\phi - \theta - \varphi$ 角，也與 Goldstein 書上相同，因此比較上會比較方便，不需要做轉換。Hasbun 教授的程式碼是尤拉角所寫出來的，所以會有奇異點問題，θ 在零度時會產生 NaN。

A 方法可以適用於任意的陀螺狀態，因此不會遇到以尤拉角表達的方法所遇到在九十度角所遭遇的奇異點問題。不過，B 方法只要 ODE 數值解收斂，結果會是比較正確的，因為 A 方法的姿態方向餘弦遞推是一種空間轉動向量的堆疊近似，因此會有誤差，而 B 方法不會有此問題。當然，A 方法適用任意的剛體轉動方程，可直接套用，而 B 的尤拉角法還得要人為的將力矩項以尤拉角表達，之後才能進行數值解，因此要看力矩項的複雜程度，因此各有其適用場合。

第三種 C 方法是，在我們從 B 方法得到尤拉角隨時間的數值解後，我們可以將尤拉角數值解轉到 body frame 計算出貼體角速度的數值解，這樣我們會得到一組 $\omega_b(t_i)$，$i = 1 \sim N$，接著我們可以用方向餘弦公式13一個一個把 $t_0 \sim t_N$ 的姿態求出來（或者說將 $\omega_b(t_i)dt$ 一個一個堆疊起來）。此方法適用的情形是，若只知道貼體角速度，力矩項未知，如飛機的姿態感測器，此時此方法是唯一可以估測物體姿態的方法。此法我們也說是以貼體角速度直接做姿態方向餘弦矩陣遞推法。這裡要注意與 A 方法不同的是，A 方法有利用貼體尤拉方程以上一個姿態來計算下一個貼角，在以下一個貼角去求得新的姿態。因此 A 方法只要知道初始角速度，與外力力矩項的型式。C 方法是只需要貼體角速度，不管是從量測得來，或者是從理論得來，C 方法只負責姿態矩陣遞推。若加上噪音過濾，C 方法即為角速度感測器解算姿態的公式，因此適用在感測器的姿態估測上面。因此我們這邊與 B 方法的比較也提供了一個平台，來驗證姿態估測演算的正確性與準確性。我們在下一章還會繼續討論姿態估測演算法。

我們將在範例與分析中詳細討論三種方法在四種經典運動中的表現，以及相互比較。

2 A、C 方法的理論驗證與第三方程式碼的比較

與我們 A 方法非常接近的是 Rapaport 書上 [9, Page 232] 的方向餘弦法，讀者只要留心比對即可觀察到兩者皆是微小轉動向量的應用，書上也有介紹另一種四元數法，我將會把我們的 A 方法的程式模擬結果（陀螺章動進動）與另一位 Christian 教授以四元數法寫成的完整的程式 easy javascript simulation（同樣是陀螺的章進動模擬）作直接的系統性比較。(不過還有一個問題待解決：還不知道如何將 Christian 教授的 Javascript 運動結果作輸出。)

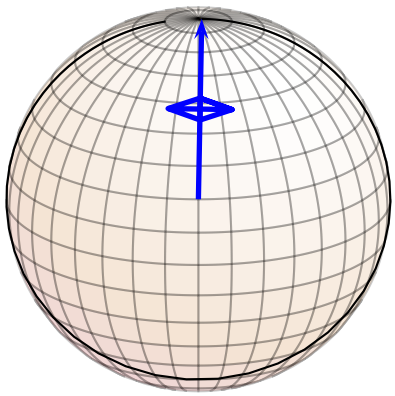
Part IV
範例與分析

以下的範例討論若是遇到我們沒有提到的參數，代表都是以預設值代入，使用者不須特別去改變，程式都有內建參數預設值。我們將參數的預設值列在 table 中。

parameter	description
M = 0.3	mass of top in kg
R = 0.025	top is a disk with radius R in meters
L = 0.005	width of disk in m
arm = 0.01	location of center of mass of top from origin in meters
counter weight = 0.05	mass doesn't spin along symmetry axis
counter weight location from origin = 0.1	location from origin
Ix,Iy,Iz	moment of inertia, calculated from above parameters, see end docs
g = 9.8	gravity constant m/s^2
freq = 20	top revolution speed in hertz, along symmetric axis
tn=1.3	end of simulation time
t0=0.0	start of time zero
samplerate = 2000	rate of iteration in Hz
classical case = 1	selection of four typical nutation and precession motions: 1,2,3,4
orien = np.array([-np.pi/3,0,0])	starting orientation vector of top from lab xyz

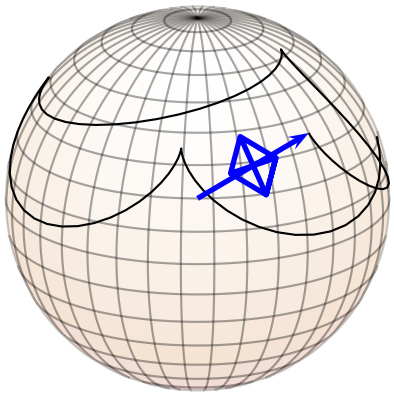
Case 1 方法 *A* 在尖點運動情況下，轉動頻率為 *1 Hz*，$\theta = 1^\circ$ 情況下的陀螺動畫。

此範例是一個簡單的驗證若陀螺在幾乎不旋轉的情況（轉動速度只有 1 Hz），此情形相當於陀螺只受重力影響而有倒下的運動，因為我們的運動是只有假設陀螺與圓點的接觸點是固定不動的，我們並沒有地面來限制陀螺的運動，因此陀螺會如同單擺一樣，擺動到最低點後，再回復到原本的高度。



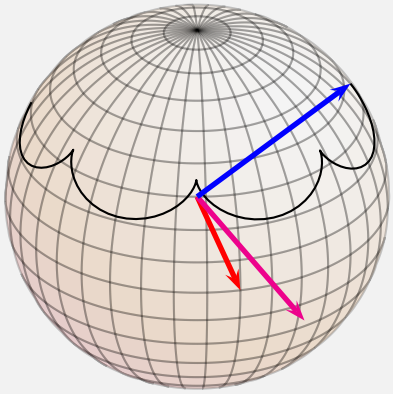
Case 2 方法 *A* 在尖點運動下，$\theta = 55^\circ$ 情況，重力常數改為 g=*20* m/s^2，的陀螺運動。

另一個簡單的範例，示範如何任意更改重力常數，同法可應用於更改其他常數，可更改的常數請參考文件最後的說明書中 Rigid Body Class 的 user adjustable parameters。重力增加後，明顯的尖點跟尖點中間的弧線運動區段變大變長了，這是很合理的。



Case 3 這裡與實體陀螺儀做定性定量的觀察比較。，可以點選連結觀察我們用 *gyroscope.com* 公司所賣的陀螺儀所製作的影片http://tinypic.com/r/2clcee/8，觀察四種經典章動進動運動。然後再用以下的模擬 *code*，模擬在相同的條件下，在電腦上觀察同樣四種章進動運動，並比較討論。我們可以觀察到，實體陀螺儀因有軸承摩擦存在而導致能量耗損很快，因此運動的 *damping* 很大，章進動的震幅只能持續幾秒就因阻尼而消失，電腦模擬就可以觀察到完整的運動。

```
# Simulation code of a real gyroscope

import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as p3
import matplotlib.animation as animation
import RBPlotFunctionV3 as RBPlot
import RGCordTransV13 as RG

b = RG.RigidBodyObject()
b.freq=-20
b.orien = np.array([-np.radians(55),0,0])
b.counter_weight = 0.05
b.counter_weight_location_from_origin = 0.1
b.GenerateDependentVariables()
b.setcase(1)
b.EulerDCMiter()
b.DrawOption['A_axes'] = True
b.DrawOption['A_square'] = True
b.DrawOption['A_z_axis_trace'] = True
b.DrawOption['A_Angular Momentum Trace'] = True
b.DrawOption['A_Angular Momentum Vec'] = True
b.DrawOption['A_Angular Velocity Vec'] = True
b.framerate = 20
b.view_azim_angle=-5
fig2 = plt.figure(2,figsize=(4,4))
ax4 = p3.Axes3D(fig2)
RBPlot.AnimationFrameSetUp(b,ax4,plt)
line_ani = animation.FuncAnimation(fig2, RBPlot.update_line_new,
 list(range(1,b.N,b.framerate)),
 fargs=b.linesarg,interval=100, blit=False,repeat=False)
plt.show()
```

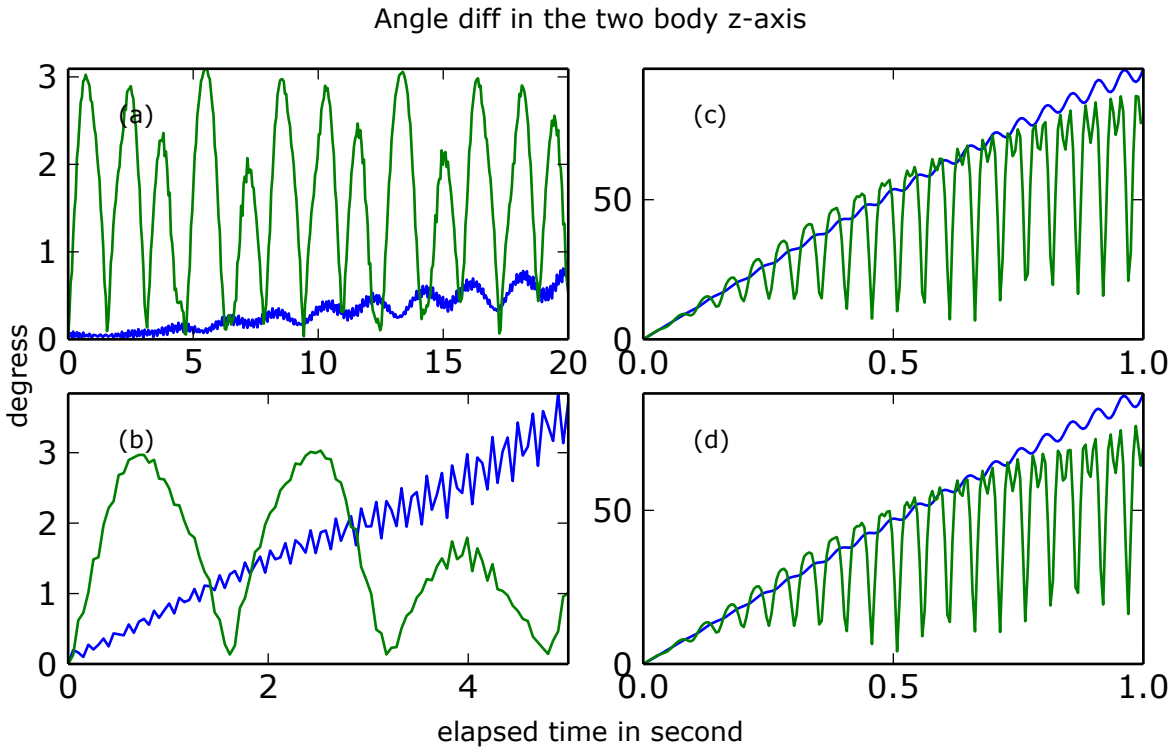


若要切換四種不同的經典運動，更改 `b.setcase(1)` 中的數字，1: 尖點，2: 環狀，3: 波紋，4: 正圓運動。若要更改陀螺初始的與 space z 軸的傾斜角，`b.orien =np.array([-`

`np.radians(55),0,0])` 中的數字即為角度 in degree。由於陀螺儀有部分的質量是不繞行陀螺對稱軸轉動，如陀螺儀的鋁製外框，但這部分的質量還是會進行章動進動，因此跟理想上的理論假設有些微的差異。我們必須加入一個 counter weight 來描述這部分的質量的運動，這樣才能正確描述真實的運動。此 counter weight 質量是可以調整其量值大小，其質心到原點的距離，這樣一來我們可以做更真實且條件範圍更廣泛的測試陀螺運動。

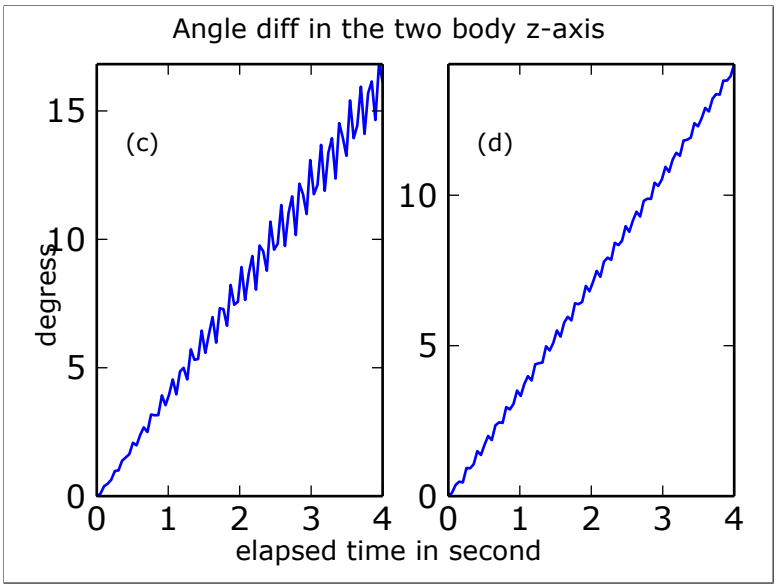
Case 4 我們比較 *ABC* 法給出的陀螺 $(1,1,1)$ 向量隨時間的變化，*B* 方法由於是採用 *python* 的 *ode solver*，此 *solver* 有確保其解有收斂 *(*我們也可以調整 `samplerate` 來觀察 *B* 法結果確實不會有變化*)*，因此這邊我們將 *B* 法視為正確解，我們將觀察 *A* 與 *C* 法給出的 $(1,1,1)$ 向量跟 *B* 法的差別，我們將畫出 *A* 法與 *B* 法給出的 $(1,1,1)$ 向量間的夾角隨時間的變化 *(*藍實線*)*，與 *C* 法與 *B* 法給出的 $(1,1,1)$ 向量間的夾角隨時間的變化 *(*紅虛線*)*。我們分別針對四種經典陀螺章動進動狀況作比較，此圖在圖*10*中。

Figure 10: AB 與 BC 法的 $(1,1,1)$ 向量比較圖。



我們發現在第一種尖點運動 a 狀況中 A 法表現很好，在 20 秒 A 方法都可以與 B 法保持在 0.5 度左右的偏移誤差，這在轉動向量積分法中表現算是非常好的，有實際應用的價值。b 經典運動下也還可以，不過在 cd 兩種運動中就越來越差了，在 cd 這兩種波紋與正圓兩種經典運動下，A 與 C 法 (1,1,1) 向量都偏移發散的非常快，五秒就已經到了 100 degree，這邊在實際應用上就會有困難度。不過，若是我們仔細觀察動畫結果，看 A 方法給出的 xyz 軸是如何隨時間發散的，我們發現事實上 z 軸對稱軸 A 與 B 方法給出的差異是非常小的，差比較多的是 x 與 y 軸，這一點是有趣的，物理上的原因還在理解當中，但，在實際的應用上我們或許可以利用這一點，當我們知道我們是在 cd 兩種運動狀況下，我們會知道，雖然 x 與 y 軸不準，但 z 軸對稱

Figure 11: cd 兩種經典運動下 AB 兩法給出的兩個 z 軸間的角度差異。



軸會是比較準確的，因此當我們喪失 xy 軸準度，我們還可以相信 z 軸。因此我們也針對 cd 兩種經典運動畫出 AB 兩法給出的 z 軸間角度差異隨時間的變化。

C 方法在 cd 兩種運動狀況表現的情況會在以下例子中討論。有趣的是，當 $\theta = 90^\circ$ 時，在 d 經典運動 (正圓運動) 情況下，C 方法的結果非常準確，這將在以下例子中呈現。目前還在了解其不同章進動及不同角度下為何會給出不同的差異結果。

Case 5 *BC* 方法在正圓運動下的比較，我們考慮 *C* 方法在兩個初始角度 $(\theta = 55^\circ and \theta = 90^\circ)$ 的準確性。圖*12*分別劃出在兩個角度下 *BC* 方法解出的陀螺 *z* 軸的軌跡圖 *locus plot*，與 *BC* 方法解出的兩組陀螺在最後時間的 *xyz* 軸 *(*紅黃藍*)*，*B* 組結果以實線劃出，*C* 組結果以虛線畫出。我們會發現在 *55* 度的時候，*C* 方法表現得並沒有很好，甚至可以說很差，但在 *90* 度的時候，*C* 方法幾乎與 *B* 方法重合，表現得極好。這是非常有趣的一個現象，我還在嘗試理解為何在不同角度下 *C* 方法的準確度會有不同，若是能找出物理原因，我們說不定可以運用此現象於姿態控制當中，運用姿態解算法在某些特殊運動下會特別準確的優勢來做姿態控制。

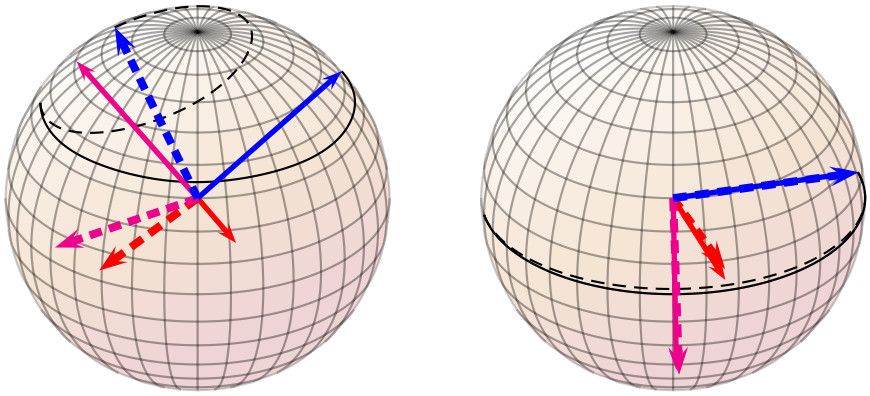
C 方法在 a 運動狀況下在 20 秒內雖然誤差比 A 方法大 (圖10)，但是由於 A 方法的誤差會持續累積，因此在四十秒後 A 方法就會有較大的誤差了。同樣的在 b 運動狀況下也有類似的狀況，五秒內 A 方法比較好，五秒後 C 方法比較好。那麼 C 方法的誤差會不會累積呢？那我們就要加長模擬的時間。

Case 6 加長模擬時間來觀察 *C* 方法的發散度。

Case 7 比較 *C* 方法使用不同轉動向量來近似所產生的不同，可以改進 *cd* 兩種運動的估測準確度。

以下也列出 ABC 方法的優劣比較與適用場合

Figure 12: Comparing BC method in circular precession motion at two initial angles.



	優點	缺點	適合用途
B 法	模擬時間可以很長 最準確	需知道外力形式 尤拉角轉換繁瑣 需解高度非線性 ODE 需要電腦來解	準確度要求高的場合
A 法	計算量小，快 程式容易	須知道外力形式 誤差較大 需要電腦來解	電腦剛體運動模擬，積分器 電腦圖學，遊戲引擎 原子分子運動模擬
C 法	不須知道外力形式 計算時間最快 不需要 PC，只消 IC 晶片 可攜式感測裝置，體積小	誤差最大	陀螺儀感測器的姿態演算解算

Part V
在綁縛式慣性感測器姿態演算的應用

我們知道在姿態感測的應用上，若是轉動軸不隨時間而變化，則姿態估測上會容易且準確非常多。如汽車的航向陀螺儀姿態感測器，由於其大部分是在平面上轉彎運動，其轉動軸多在垂直方向，因此姿態的估測上是容易且準確的。但若是轉動軸會隨時間而變化，如同人體、飛機、或陀螺的姿態估測，其姿態的準確度隨時間而發散速度是很快的。一般在作轉動軸變化的姿態估測上都比較困難，一般都會融合其他感測方法，如 GPS、地磁、重力來輔助姿態的判定。而此篇討論一個重點就是是否能夠在無其他融合感測方法時，對轉動軸變化的姿態估測的發散度有所歸納，我們以轉動速度相當快（轉動軸變化也快且大）的陀螺為測試平台，與陀螺的尤拉角尤拉方程數值解作系統性比較，來看看姿態估測演算法（C 方法）在極端的轉動條件下，準確程度如何。並且討論為何四種經典章動進動會有不同的發散程度，以及是否我們可以利用發散程度較小的情況並作實際的應用。因此此物件導向化的軟體平台適合作為驗證，比較姿態估測演算法的平台。

上面的 DCM 的程式可以獨立出來應用在感測器的姿態估測上。由於 DCM 是考慮貼體角速度 $\vec{\omega}$，因此主要應用在綁縛式慣性感測元件（strap-down IMU）上，此類包含範圍涵蓋陀螺儀角速度感測器，微機電角速度感測器，等諸如此類的元件。只要把元件讀到的貼體角速度資料輸出，然後正確的輸入到我們這邊的 DCM 遞推程式，就可以得到物件轉動的姿態。這邊大部分的書籍都無法從原理說起，都只有提供公式，而且也很少詳細的說明公式如何使用。不過要補充的是，我這邊只有轉動部分，並沒有平移部分的演算法，不過轉動姿態是姿態估測的基礎，平移的演算是相對容易很多的。並且，這邊也沒有噪音過濾的功能，因為任何的量測儀器都會有噪音，因此在輸入 DCM 遞推前，必須先經過過濾。一般常見的過濾方法是卡爾曼濾波，噪音濾波的討論是我下一步的目標。

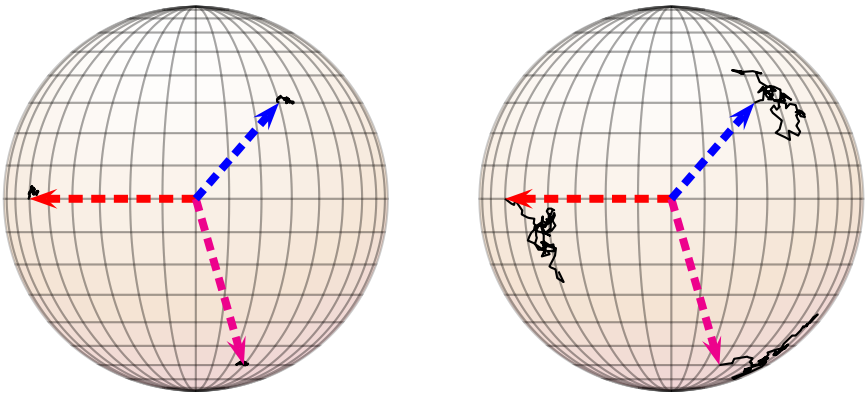
Case 8 *C* 方法加上噪音後的靜止測試。

我們使用 C 方法來測試在此狀況下 C 方法給出的姿態隨時間的運動是否也是靜止的。假設貼體角速度是零加上一個隨機的噪音 $\vec{\omega}(t) = 0 + Noise(t)$，$Noise(t)$ 為一固定 Amplitude 乘上 -1 到 1 的隨機數，固定的 Amplitude 單位為 radian per second，量值大小可從 `b.IncludeNoiseInOmega(5)` 中的數字 5 更改。我們比較兩個 Amplitude，產生此動畫的程式如下:

```
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as p3
import matplotlib.animation as animation
import RBPlotFunctionV3 as RBPlot
import RGCordTransV13 as RG

b = RG.RigidBodyObject()
b.tn=4
b.samplerate=500
b.GenerateDependentVariables()
b.setcase(1)
b.HasbunEulerEquationODEsolve()
b.eulerW2bodyW()
b.w_body_hasbun = b.w_body_hasbun*0
b.IncludeNoiseInOmega(5)
```

```
b.EulerDCMiter()
b.directDCMiter()
b.DrawOption['C_axes'] = True
b.DrawOption['C_z_axis_trace'] = True
b.framerate = 200
b.view_azim_angle=70
fig2 = plt.figure(2,figsize=(2.5,2.5))
ax4 = p3.Axes3D(fig2)
RBPlot.AnimationFrameSetUp(b,ax4,plt)

line_ani = animation.FuncAnimation(fig2, RBPlot.update_line_new,
 list(range(1,b.N,b.framerate)),
 fargs=b.linesarg,interval=100, blit=False,repeat=True)
plt.show()
```

我們畫出 z 軸與 Z 軸隨時間的軌跡圖，我們這邊將兩個 Amplitude 的結果劃出，左邊是 1 radian per sec，右邊是 5 radian per sec。這樣我們就可以跟市面上賣的陀螺儀及 angular rate sensor 做比較。57.32 degree = 1 radian，所以我們這邊是考慮噪音很大的陀螺儀，或是陀螺儀在環境溫度變化或機械震動很大的極端的例子。



Case 9 *C* 方法加上噪音後，經過 *running average* 的濾波方式，來測試尖點運動情況下的估測能力。

**need thinking 即時解尤拉方程然後作 DCM 姿態遞推的好處是，不用事先知道力矩項，因此會比尤拉角尤拉方程方法更適用於力矩會隨時間變動或力矩為未知的情況，舉例來說，我們可以用在機器人的恣態控制上面，又或者，可以用在有用到剛體轉動的遊戲物理引擎上面。這些狀況都是尤拉角尤拉方程中的力矩項無法事先預知，因此無法事先解其尤拉角尤拉方程。又或者，若是力矩項太過複雜，如無人機的空氣動力，也適合使用我們這邊所介紹的即時解尤拉方程並作姿態遞推，只要有角加速度計可以測量角加速度就可以。我們這邊所介紹的陀螺，是一個應用的例子，由於陀螺的轉動速度是每秒 20 轉，因此轉動速度很高，對於姿態遞推的精度要求算很高，因此陀螺適合當作驗證姿態遞推演算的非常好的例子，若陀螺能算的準，其他轉動速度不高的情況將會非常準。陀螺的情況我們可以比喻為，把機器人或無人機丟入龍捲風裡面，並且要作姿態控制。

Part VI
補充

陀螺等周速運動 Figure 7(d) 的初始值條件如何計算呢? 等周速的條件在 Goldstein 第二版 5-77 式給出

$$Mgl = \dot{\phi}\left(I_3\omega_3 - I_1\dot{\phi}\cos\theta_0\right) \tag{22}$$

，不過此式是由尤拉角 (euler angles) 給出，但我們需要的是 anguler velocity along body 的初始值，因此我們必須轉換尤拉角到 anguler velocity along body，方法如下。上式中 ω_3 即為我們的 $(\omega_z)_b$，比如說是 20 Hz，θ_0 即為我們之前的 orien 向量所定，由上式可求出兩組 $\dot{\phi}(t_0)$。另外尤拉角跟 anugler velocity along body 的關係式在書 [1, Chapter 4, Equation 125] 給出

$$(\omega_x)_b = \dot{\phi}\sin\theta\sin\psi + \dot{\theta}\cos\psi \tag{23}$$
$$(\omega_y)_b = \dot{\phi}\sin\theta\cos\psi - \dot{\theta}\sin\psi \tag{24}$$
$$(\omega_z)_b = \dot{\phi}\cos\theta + \dot{\psi} \tag{25}$$

知道 $\dot{\phi}(t_0)$、θ_0、$\dot{\theta}_0 = 0$ 及 ψ_0，用一二條後就可得到 $\omega_{xb}(t_0)$ 與 $\omega_{yb}(t_0)$，這樣我們就得到 anguler velocity along body 的初始值。因為 $\dot{\phi}(t_0)$ 有兩組，因此解出的貼體角速度也會有兩組，兩組的物理意義分別如下，一種情況是 fast top，這個狀況相當於重力的影響遠小於總角動量 L，因此這個特別的例子基本上相當於忽略重力，而陀螺基本上會像一個 free top 一樣進行 precession。另一種狀況是 slow top，也就是上面模擬結果中第四種的狀況，這裡提供的 python 程式所有情況都可以模擬。另外一個特殊的情況是在 fast top 的情形下，如果初始值 $\theta_0 = 0$，也就是陀螺 z 軸的起始狀態是垂直於水平面的，這樣的話陀螺幾乎會像靜止不動一樣，我們也叫這情況做 sleeping top。這些計算經典狀況的程式方程被包含在 setcase() 方程裡。

Remark 10 要陀螺具有 *Precession and Nutation* 的動作，$L/\Delta L$ 必須要大，如果 L 小於 ΔL，則只會有陀螺質量受重力影響往下倒下的運動 *(不過這對檢查程式有沒有錯誤很有幫助!)*，理想上 L 至少要大於 ΔL，最好 L 大大於 ΔL。化成數值上的比較：這代表

$$L \gg \Delta L \Rightarrow I\cdot 2\pi f \gg \vec{\Gamma}\Delta t \Rightarrow I\cdot 2\pi f \gg \vec{r}\times\vec{F}\cdot 1/f \Rightarrow f \gg \sqrt{\frac{arm\cdot Mg\cdot\sin(\theta)}{2\pi I\cdot G}} \tag{26}$$

where θ is gyro's tilt angle and G is moment of inertial geometry factor. 考慮 Δt 的量級大約是陀螺轉幾圈的時間 *(characteristic time)*，量級上約是 $\sim 1/f$，若假設 *arm* 是 *10 cm, M = 1kg, g=10 m/s^2, I = 0.5M(0.05)^2,* 則 *f* 最少要 *10 Hertz* 以上。因此我們將以這些參數比較 *f = 1, 10, 50 Hertz* 所給出的陀螺運動。

References

[1] Herbert Goldstein, *Classical Mechanics*. Addison Wesley, Massachusetts, 2nd Edition, 1980

[2] David Tong, *Classical Dynamics University of Cambridge Part II Mathematical Tripos*. Cambridge UK, 2004-2005, (Course note, available on the web)

[3] Matlab online documentation - Ordinary differential equations., Matlab R2014a

[4] 徐小明钟万勰，刚体动力学的四元数表示及保辛积分，《应用数学和力学》2014，35（1）：111

[5] Javier E. Hasbun, *Classical Mechanics with Matlab Appications.* Jones and Bartlett Publishers, London UK, 2009

[6] D.H. Titterton and J.L. Weston, *Strapdown Inertial Navigation Technology*, Peter Peregrinus Ltd., London UK, 1997

[7] David Baraff, Physically Based Modeling - Rigid Body Simulation, Pixar Animation Studios notes

[8] Harvey Gould and Jan Tobochnik and Wolfgang Christian, *An Introduction to Computer Simulation Methods Third Edition*, Addison-Wesley, 2006, draft available on comPADRE

[9] Dennis Rapaport, *The Art of Molecular Dynamics Simulation 2nd Edition*, Cambridge University Press, 2004

[10] Eugene Butikov, Precession and nutation of a gyroscope, St. Petersburg, Russia, http://butikov.faculty.ifmo.ru/Applets/Gyroscope.pdf

[11] Paul Savage, *Strapdown Inertial Navigation Integration Algorithm Design Part 1: Attitude Algorithms*, Journal of Guidance, Control, and Dynamics, Vol. 21, No. 1, January - February 1998

[12] David Eberly, *Game Physics,* 2nd Edition, CRC Press, 2010

This document is prepared with Scientific Workplace 5.0 and typeset with Tex Live 2013 (Xelatex).
Blogpost: http://ppt.cc/yS2-
Python code: https://drive.google.com/file/d/0B96HmLH-SQVmM1dvYlFiQm9ESGM/edit?usp=sharing

GyroSoft Simulation Documentation

Release 0.0.1

TL

March 20, 2015

Contents

Contents:

This document shows all the functions used in the python program to calculate the equations of motion of the top, the parameters users can change, and the 3D figure and animation generating functions.

1 Funtional Descriptions of equation of motion of top

Module of the Rigid Body class and functions for spinning top's equation of motions.

Created on Tue Nov 11 16:05:05 2014

@author: whymranderson.blogspot.tw

There are two global functions CK and euler2space.

`RGCordTransV13.`**`CK`**(*rotvec*)
Cayley-Klein parameters. Important: the built rotation matrix is with its couterclockwise active sense. This is basically the Rodriguez rotation formula in a matric form.

`RGCordTransV13.`**`euler2space`**(*eulerW*)
Return Euler angles transformation matrix from lab frame to time t frame.

Matrix takes on passive right-hand sense. The list of input argument w is w(0):phi, w(1):phi_dot, w(2):theta, w(3):theta_dot, w(4):psi

---

Functions of RigidBodyOjbect class followed by parameters of top that can be used by the user in the program:

class `RGCordTransV13.`**`RigidBodyObject`**
Rigid body class

CalculateRhoInJCycle(*w1*, *w2*, *w3*, *J_dt*)
Return rotation vector using J cycle calculation.

Rho should have the same dimension -1 as the sampling rate.

EulerDCMiter()
Integration of body angular velocity with a DCM method and advances with Newton-Euler equation.

This is the A method. Also this method use w(t_{i+1})dt in rotation vector approximation by default. One can also choose to use the J-cycle rotation approximation with the use_Jcycle=1 attribute.

EulerDCMiter_wt_i()
'This is A method with omega(t_{i}) approximation.

GenerateDependentVariables()
Build and initialize needed arrays and parameters

HasbunEulerEquationODEsolve()
Numerically solve Newton-Euler equations with Euler angles using Python ode library.

This is rewritten from Prof Hasbun's matlab code in his book **Classical Mechanics with MATLAB Applications** This method is essentially the same with the Lagrange method. See text. The list of Euler_angles_hasbun = w(0):phi, w(1):phi_dot, w(2):theta, w(3):theta_dot, w(4):psi

IncludeNoiseInOmega(*NoiseAmplitudeFactor*)
Include noise in body angular velocity in C method.

Return a noisy w_body_hasbun. Take one argument as noise amplitude w.r.t original velocity amplitude.

directDCMiter()
direct DCM iteration from body anguler velocity. This is method C.

w_lab are calculated from method B

eulerW2bodyW(*euler_angles_hasbun*)
Convert Hasbun's euler angles solution to body angular velocity

The euler_angles_hasbun array has size N+1-1

setcase(*case*, **arg*)
Choose from four classical nutation and precession motions or set customary initial condition

topEOM(*wi*, *tlist*, *torquei*)
EOM of a symmetric top

topRK(*wi*, *torquei*)
4th order Rugge Kutta numeric solver

uniformprecesscal()
return the initial condition of body angular velocity required to generate a uniform precession top.

Calculation base on the parameters and the setup of the top. The formula is in Goldstein's book. The formula has a singular point at $ heta$ = 90 degree, but a unique way is developed to deal with this. So the program will work at any angle.

Ix = 0.0005768750000000001
x moment of inertia 0.5*M*R**2

Iy = 0.0005768750000000001
y moment of inertia 0.25*M*R**2 + 1/12*M*L**2 +M*arm**2 + counter_weight*counter_weight_location_from_origin**2

Iz = 9.375000000000002e-05
z moment of inertia 0.5*M*R**2

L = 0.005
width of disk in m

M = 0.3
mass of top in kg

R = 0.025
top is a disk with radius R in meters

arm = 0.01
location of center of mass of top from origin in meters

classical_case = 1
selection of four typical gyroscopic motions: 1,2,3,4

counter_weight = 0.05
mass doesn't spin along symmetry axis

counter_weight_location_from_origin = 0.1
location from origin

freq = 20.0
top revolution speed in hertz, along symmetric axis

g = 9.8
gravity m/s^2

orien = array([-1.04719755, 0. , 0.])
starting orientation vector of z' of top from lab xyz

samplerate = 2000
rate of iteration in Hz

t0 = 0.0
start of time zero

tn = 1.3
end of simulation time, take longer time if tn > 10 seconds

view_azim_angle = 185
3D view azimuthal angle

The following parameters are used in command like `Drawoption['A_axes'] = True` to make the belonging physical observable visible in a 3D animation.

class `RGCordTransV13.`**RigidBodyObject**

GenerateDependentVariables()

DrawOption
'A_axes': False,

'B_axes': False,

'C_axes': False,

'A_z_axis_trace': False,

'B_z_axis_trace': False,

'A_Angular Momentum Trace' : False,

'B_Angular Momentum Trace' : False,

'Angular Velocity Trace' : False,

'A_Angular Momentum Vec' : False,

'B_Angular Momentum Vec' : False,

'A_Angular Velocity Vec' : False,

'A_square': False

2 3D plotting and animation functions

Plot and animation functions/environment setup for Rigid Body class

Created on Tue Nov 11 16:05:05 2014

@author: whymranderson.blogspot.tw

`RBPlotFunctionV3.`**`AnimationFrameSetUp`**(*RBO*, *ax*, *plt*, *istep*)
plot the first frame of animation

`RBPlotFunctionV3.`**`CK`**(*rotvec*)
Cayley-Klein parameters. Important: the built rotation matrix is with its couterclockwise active sense. This is basically the Rodriguez rotation formula in a matric form.

`RBPlotFunctionV3.`**`CalDiffInDeg`**(*cordvec*)
Calculate and return angle difference between A and B method's 111 unit vectors

`RBPlotFunctionV3.`**`CalDiffInDegBC`**(*cordvec*)
Calculate and return angle difference between B and C method's 111 unit vectors

`RBPlotFunctionV3.`**`get111norm`**(*xyzaxes*)
Return array of body (1,1,1) unit vector as a function of time N in size (N,3)

`RBPlotFunctionV3.`**`getPlotAnimationLines`**(*RBO*)
Construct and return the line animation data sequence in a rigid body object for 3D animation

`RBPlotFunctionV3.`**`plot_back`**(*axes*)
plot the lucid sphere, back part

`RBPlotFunctionV3.`**`plot_body_space_cone`**(*ax*, *zn*, *omegavec*, *Lvec*)
Plot the body and space cone

`RBPlotFunctionV3.`**`plot_front`**(*axes*)
plot the lucid sphere, front part

`RBPlotFunctionV3.`**`update_line_new`**(*x*, **arg*)
line animation function, will be called by functionanimation, take multiple line data sequences as arguments

3 Updates and changelog

2015/3/10 Comparison to Christian Wolfgang's simulation of gyroscope is added. The example file to run is `Gyroscope_Christian_Wolfgang_compared.py`.

2013/3/5 Rotation vector approximation J-cycle added to `EulerDCMiter()` as an option.

2015/2/26 Space cone and body cone plotting function added. Function's name is `plot_body_space_cone()`. It is a static plot. 3D animating cone will need to be integrated in the future.

2015/2/10 Method A now has a option to use Python ODE solver instead of the RK's method I wrote. The accuracy is arguably the same. But using Python ODE solver can lower the sampling rate a lot.

2014/12/29 C method noise-included still case added.