欧拉角速度推导与不建议使用欧拉角的原因

原创 找不到服务器zhn

摘要 欧拉角速度指欧拉角对时间的微分与角速度的关系，用于刚体姿态运动学建模，本文从一种复杂但严谨（应该）的角度推导这一关系，建议使用欧拉角来描述姿态。

欧拉角 微分方程

$$\vec{w}=\begin{bmatrix}1 & 0 & -\sin\theta\\ 0 & \cos\phi & \sin\phi\cos\theta\\ 0 & -\sin\phi & \cos\phi\cos\theta\end{bmatrix}\begin{bmatrix}\dot\phi\\ \dot\theta\\ \dot\psi\end{bmatrix}=\begin{bmatrix}\dot\phi-\sin\theta\dot\psi\\ \cos\phi\dot\theta+\sin\phi\cos\theta\dot\psi\\ -\sin\phi\dot\theta+\cos\phi\cos\theta\dot\psi\end{bmatrix}$$
$$\begin{bmatrix}\dot\phi\\ \dot\theta\\ \dot\psi\end{bmatrix}=\begin{bmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta}\end{bmatrix}\vec{w}$$

其中 ϕ,θ,ψ 分别为横滚角、俯仰角、偏航角。

欧拉角微分方程推导

推导欧拉角微分方程有3种方法。第一种方法比较简单，好多教材里也都这么解释，可以参考
欧拉角导数和角速度之间的转换关系推导 -CSDN博客
另外两种方法用的都是根据旋转矩阵微分方程和旋转矩阵与欧拉角的关系来推导，可以参考
Angular Velocity expressed via Euler Angles -Physics Stack Exchange
【刚体运动】欧拉角时间导数与角速度 -知乎

下面写一下我对第一种方法的理解。把世界系和本体系分别记作 I,B，世界系I绕Z轴旋转 ψ 角后的坐标系记作Z，Z系绕Y轴旋转 θ 角后。从一个坐标系A到另一个坐标系B的旋转矩阵记作 R_{BA}，即 $v^B=R_{BA}v^A, v^A=R_{AB}v^B$，$A$系与$B$系之间的角速度向量记作 w_{AB}，并系下的坐标是一样的（有人不理解为什么一样的话我再补充）。首先可以直观地得到

$$\begin{aligned}w_{ZI}^Z&=w_{ZI}^I=[0\quad 0\quad \dot\psi]^\mathrm{T}\\ w_{YZ}^Y&=w_{YZ}^Z=[0\quad \dot\theta\quad 0]^\mathrm{T}\\ w_{BY}^B&=w_{BY}^Y=[\dot\phi\quad 0\quad 0]^\mathrm{T}\end{aligned}$$

三次旋转的旋转矩阵分别为

$$R_{ZI}=R_z(-\psi),\ R_{YZ}=R_y(-\theta),\ R_{BY}=R_x(-\phi)$$

于是

$$
\begin{aligned}
w_{ZI}^{B} &= R_{BY}\, R_{YZ}\, w_{ZI}^{Z} \\
w_{YZ}^{B} &= R_{BY}\, w_{YZ}^{Y} \\
w_{BI}^{B} &= w_{BY}^{B} + w_{YZ}^{B} + w_{ZI}^{B} \\
&= \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_x(-\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_x(-\phi) R_y(-\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \left(\begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \right) \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} - \sin\theta\dot{\psi} \\ \dot{\theta} \\ \cos\theta\dot{\psi} \end{bmatrix} \\
&= \begin{bmatrix} \dot{\phi} - \sin\theta\dot{\psi} \\ \cos\phi\dot{\theta} + \sin\phi\cos\theta\dot{\psi} \\ -\sin\phi\dot{\theta} + \cos\phi\cos\theta\dot{\psi} \end{bmatrix} \\
&= R_x(-\phi) \left(\begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_y(-\theta) \left(\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_z(-\psi) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \right) \right)
\end{aligned}
$$

下面写一下我想的另一种方法。由旋转矩阵微分方程可以得到角速度关于旋转矩阵的表达式
（旋转矩阵微分方程的推导见 角速度变化时四元数和旋转矩阵微分方程的证明）

$$
\begin{aligned}
\dot{R}^{\mathrm{T}} &= -\vec{w} \times R^{\mathrm{T}} \\
\vec{w}^{\times} &= -\dot{R}^{\mathrm{T}} R
\end{aligned}
$$

或者

$$
\begin{aligned}
R^{\mathrm{T}} R &= I \\
\dot{R}^{\mathrm{T}} R + R^{\mathrm{T}} \dot{R} &= 0 \\
\vec{w}^{\times} &= R^{\mathrm{T}} \dot{R}
\end{aligned}
$$

因为 R 可以看作是多元复合函数 $R(\phi(t), \theta(t), \psi(t))$，所以

$$
\frac{\mathrm{d}R}{\mathrm{d}t} = \frac{\partial R}{\partial \phi}\dot{\phi} + \frac{\partial R}{\partial \theta}\dot{\theta} + \frac{\partial R}{\partial \psi}\dot{\psi}
$$

于是

$$
\vec{w}^{\times} = R^{\mathrm{T}}\dot{R} = R^{\mathrm{T}}\frac{\partial R}{\partial \phi}\dot{\phi} + R^{\mathrm{T}}\frac{\partial R}{\partial \theta}\dot{\theta} + R^{\mathrm{T}}\frac{\partial R}{\partial \psi}\dot{\psi}
$$

其中 $R^{\mathrm{T}}\frac{\partial R}{\partial \phi}, R^{\mathrm{T}}\frac{\partial R}{\partial \theta}, R^{\mathrm{T}}\frac{\partial R}{\partial \psi}$ 是3个反对称矩阵，简单想想可以理解，因为 $w^{\times}$ 是反对称矩阵，而 $\dot{\phi}, \dot{\theta}, \dot{\psi}$ 都是标量并且可以是任意值，那么
时，$R^{\mathrm{T}}\frac{\partial R}{\partial \phi}$ 一定是反对称矩阵，同理，3个矩阵都是反对称矩阵，并且分别对应角速度和欧拉角时间导数之间转换矩阵 T 的3列元素

$$\vec{w} = T \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

$$\vec{w}^{\times} = \begin{bmatrix} R^{\mathrm{T}} \frac{\partial R}{\partial \phi} & R^{\mathrm{T}} \frac{\partial R}{\partial \theta} & R^{\mathrm{T}} \frac{\partial R}{\partial \psi} \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

$$\begin{aligned} T &= \begin{bmatrix} \left(R^{\mathrm{T}} \frac{\partial R}{\partial \phi}\right)_{\times} & \left(R^{\mathrm{T}} \frac{\partial R}{\partial \theta}\right)_{\times} & \left(R^{\mathrm{T}} \frac{\partial R}{\partial \psi}\right)_{\times} \end{bmatrix} \\ &= \begin{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}_{\times} & \begin{bmatrix} 0 & \sin(\phi) & \cos(\phi) \\ -\sin(\phi) & 0 & 0 \\ -\cos(\phi) & 0 & 0 \end{bmatrix}_{\times} & \begin{bmatrix} 0 & -\cos(\phi)\cos(\theta) & \sin(\phi)\cos(\\ \cos(\phi)\cos(\theta) & 0 & \sin(\theta) \\ -\sin(\phi)\cos(\theta) & -\sin(\theta) & 0 \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & -\sin\phi\cos\theta \\ 0 & \sin\phi & \cos\phi\cos\theta \end{bmatrix} \end{aligned}$$

推导代码

下面的代码使用 sympy 推导转换矩阵 T

```
1  import sympy as sp
2  phi, theta, psi = sp.symbols('\\phi, \\theta, \\psi')
3  Rx = sp.Matrix([
4      [1,           0,               0],
5      [0, sp.cos(phi),    -sp.sin(phi)],
6      [0, sp.sin(phi),     sp.cos(phi)],
7  ])
8  Ry = sp.Matrix([
9      [sp.cos(theta) , 0, sp.sin(theta)],
10     [0             , 1,             0],
11     [-sp.sin(theta), 0, sp.cos(theta)],
12 ])
13 Rz = sp.Matrix([
14     [sp.cos(psi), -sp.sin(psi), 0],
15     [sp.sin(psi),  sp.cos(psi), 0],
```

另外使用这一代码时，只需要改代码第18行处3次旋转的顺序就可以方便地计算出气动角微分方程（见 攻角、侧滑角、倾侧角与欧拉角的关系

$$\vec{w} = \begin{bmatrix} \cos\beta\cos\alpha & 0 & -\sin\alpha \\ -\sin\beta & 1 & 0 \\ \sin\alpha\cos\beta & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} \dot{\sigma} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix}$$

$$\begin{bmatrix} \dot{\sigma} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} \frac{\cos\alpha}{\cos\beta} & 0 & \frac{\sin\alpha}{\cos\beta} \\ \cos\alpha\tan\beta & 1 & \sin\alpha\tan\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \vec{w}$$

仿真

仿真一下来验证sympy推导出的结果正确，将欧拉角微分方程的仿真结果与四元数微分方程的仿真结果比较。
使用欧拉角微分方程建模的被控对象代码如下

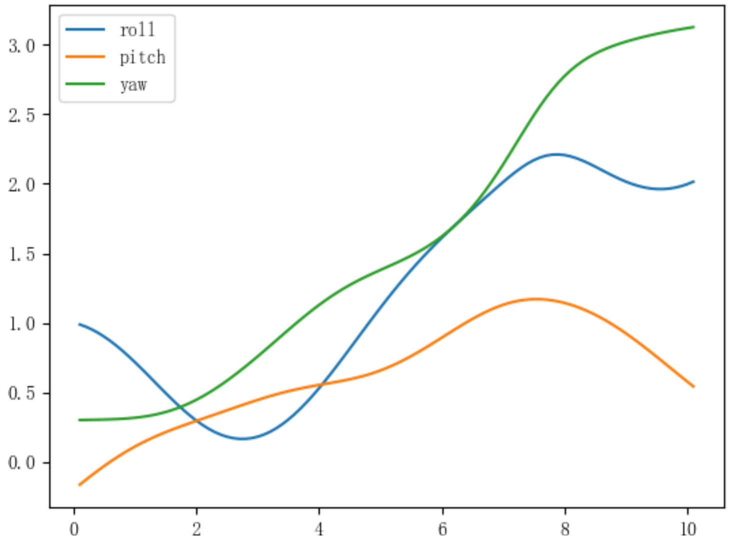
```
# rigidbodyeuler.py
# 四阶龙格库塔法根据欧拉角微分方程建立刚体的运动学和动力学模型
import numpy as np
# from quaternions import *

# 刚体
# J: 刚体的转动惯量矩阵
# initEuler: 刚体的初始欧拉角
# initOmega: 刚体的初始角速度向量
class RigidBody:
    def __init__(self, J, initEuler
        self._J = J
```

```python
        e1 = initEuler
        w1 = initOmega
        self._t = 0
        self._Jinv = np.linalg.inv(self._J)
        self._states = np.concatenate((e1, w1), dtype=float)
        self.ctrlLaw = lambda x: 0  # 控制律

    def Simulate_OneStep(self):
        h = 0.01
        K1 = self._ODE4Function(self._t, self._states)
        K2 = self._ODE4Function(self._t+h/2, self._states + h/2*K1)
        K3 = self._ODE4Function(self._t+h/2, self._states + h/2*K2)
        K4 = self._ODE4Function(self._t+h, self._states + h*K3)
        dx = h/6*(K1 + 2*K2 + 2*K3 + K4)
        self._states += dx

    def Get_State(self):
        return self._states
    def Get_EulerVec(self):
        # q1 = Euler_To_Quaternion(self._states[0:3])
        # return Quaternion_to_Euler(q1)
        return self._states[0:3]

    def _ODE4Function(self, t, x):
        E, W = x[0:3], x[3:6]
        matTR = np.matrix([
            [1, 0, -np.sin(E[1])],
            [0,  np.cos(E[0]), np.sin(E[0])*np.cos(E[1])],
            [0, -np.sin(E[0]), np.cos(E[0])*np.cos(E[1])],
        ])
        # matTR = np.matrix([
        #     [1, np.sin(E[0])*np.tan(E[1]), np.cos(E[0])*np.tan(E[1])],
        #     [0, np.cos(E[0]), - np.sin(E[0])],
        #     [0, np.sin(E[0]) / np.cos(E[1]), np.cos(E[0]) / np.cos(E[1])],
        # ])
        dE = np.linalg.inv(matTR) @ W
        # dE = matTR @ W
        torque = self.ctrlLaw(x)
        dW = self._Jinv @ (torque - np.cross(W, self._J @ W))[0]
        return np.concatenate((np.array(dE)[0], np.array(dW)[0]))
```

四元数微分方程代码见 刚体四元数姿态控制，
主程序如下

```python
# main.py
import numpy as np
import matplotlib.pyplot as plt
# from rigidbodyquaternion import RigidBody
from rigidbodyeuler import RigidBody
J = np.matrix([
            [1, 0, 0],
            [0, 10, 0],
            [0, 0, 3],
])
usv1 = RigidBody(J, np.array([1, -0.2, 0.3]), np.array([-0.1, 0.2, -0.3]))
t = 0
plottime, plotdata = [], []
while t < 10:
    for n in range(10):
```

四元数和欧拉角微分方程的仿真结果相同，如图所示。

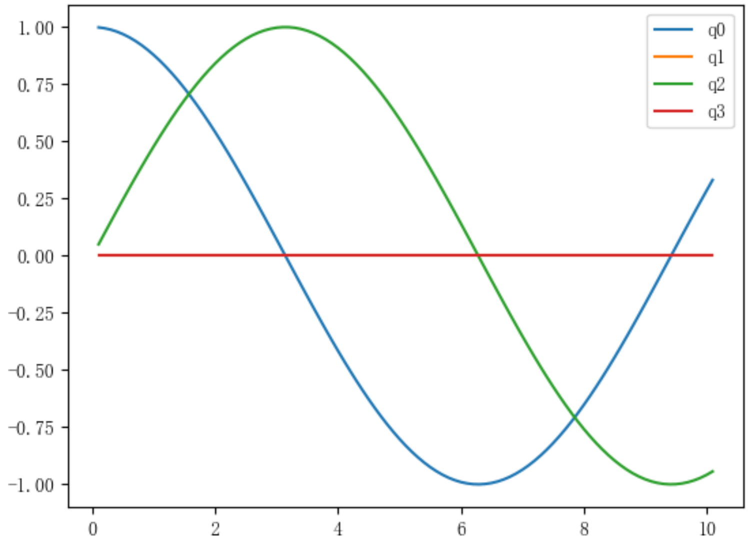


欧拉角的缺点

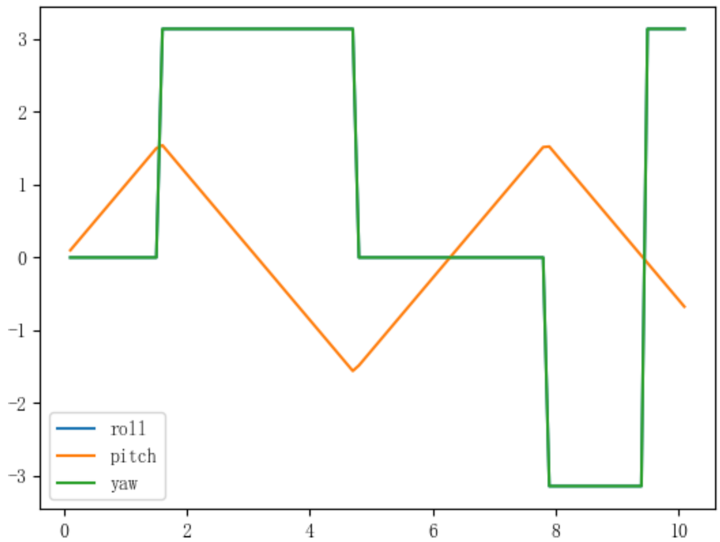
可以说欧拉角除了看上去更直观以外几乎没有任何优点了。本文从欧拉角微分方程的角度谈谈欧拉角的其中一个缺点：在限定的范围以外知道的是，四元数、欧拉角、旋转矩阵这三种描述姿态的方法之间可以互相换算，也几乎可以一一对应，之所以说“几乎”就是因为，如果旋转应关系就出了一些问题。

先看看当绕俯仰轴(Y轴)旋转一圈时，3种姿态表示方法的表现。

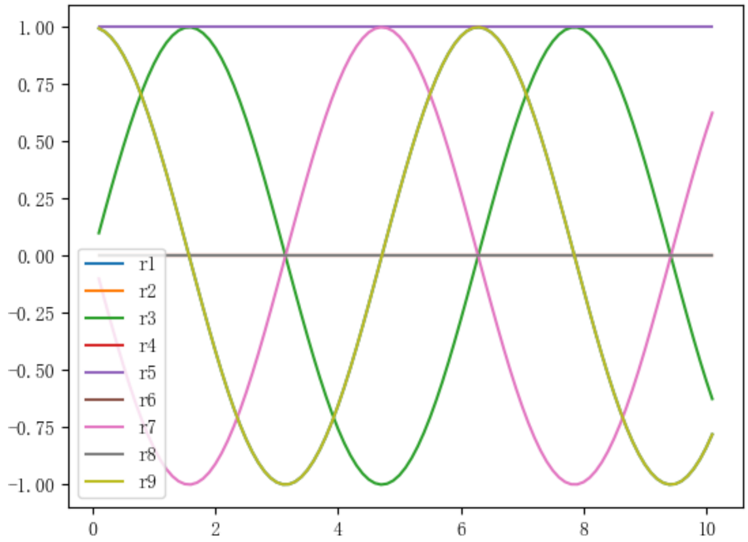
四元数：



欧拉角：



旋转矩阵：



由此可见，在大角度机动时欧拉角存在严重的不连续问题，应尽量避免使用。
画这3幅图使用四元数微分方程的代码，把控制律设成0，初始欧拉角为全0，初始角速度为 $[0,1,0]$。如果使用欧拉角微分方程，则需要把俯仰以内。也可以直接简单地使用欧拉角转一圈，同时转四元数和旋转矩阵出图，但此时我就发现了欧拉角转四元数的问题。

下面的代码是最常见的欧拉角转四元数的代码

```python
def Euler_To_Quaternion(euler):
    from numpy import sin, cos
    cr = cos(euler[0] * 0.5)
    sr = sin(euler[0] * 0.5)
    cp = cos(euler[1] * 0.5)
    sp = sin(euler[1] * 0.5)
    cy = cos(euler[2] * 0.5)
    sy = sin(euler[2] * 0.5)
    q0 = cy * cp * cr + sy * sp * sr
    q1 = cy * cp * sr - sy * sp * cr
    q2 = sy * cp * sr + cy * sp * cr
    q3 = sy * cp * cr - cy * sp * sr
    return np.array([q0, q1, q2, q3])
```

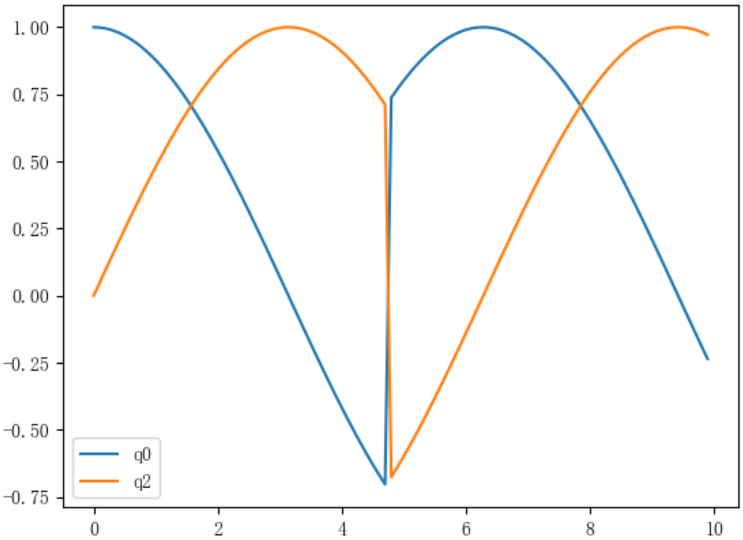
这一代码出自下面这个公式

$$
\begin{aligned}
Q &= \begin{bmatrix} \cos(\psi/2) \\ 0 \\ 0 \\ \sin(\psi/2) \end{bmatrix} \begin{bmatrix} \cos(\theta/2) \\ 0 \\ \sin(\theta/2) \\ 0 \end{bmatrix} \begin{bmatrix} \cos(\phi/2) \\ \sin(\phi/2) \\ 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix}
\end{aligned}
$$

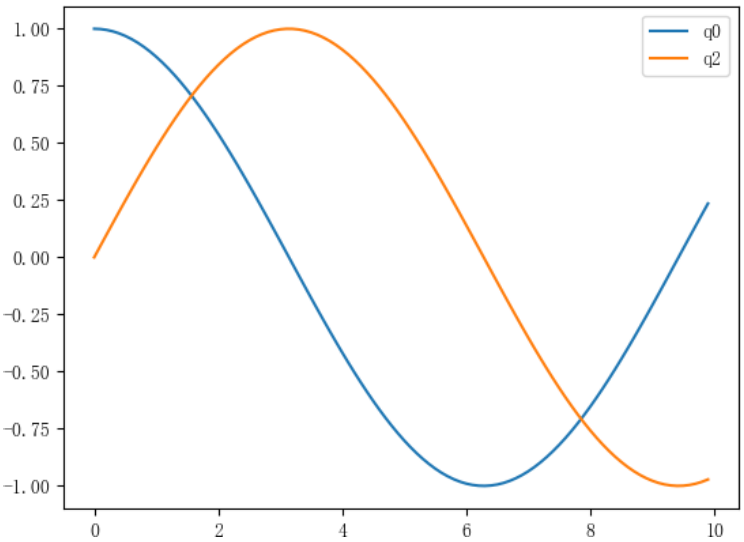
同样地绕y轴转一圈看看欧拉角转四元数式(1)的表现。首先可以求出四元数随时间变化的解析解。设角速度为 $\vec{w}=[0,1,0]^{\mathrm{T}}$，四元数初值为 $[1,0,0,0]^{\mathrm{T}}$，代入四元数微分方程

$$
\begin{aligned}
\begin{bmatrix}\dot{q}_0\\ \dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{bmatrix} &= \frac{1}{2}\begin{bmatrix}0 & -w_x & -w_y & -w_z\\ w_x & 0 & w_z & -w_y\\ w_y & -w_z & 0 & w_x\\ w_z & w_y & -w_x & 0\end{bmatrix}\begin{bmatrix}q_0\\ q_1\\ q_2\\ q_3\end{bmatrix}\\
&= \frac{1}{2}\begin{bmatrix}0 & 0 & -1 & 0\\ 0 & 0 & 0 & -1\\ 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}\begin{bmatrix}q_0\\ q_1\\ q_2\\ q_3\end{bmatrix}\\
\dot{q}_1 &= -0.5q_3 = 0\\
\dot{q}_3 &= 0.5q_1 = 0\\
\dot{q}_2 &= 0.5q_0\\
\ddot{q}_0 &= -0.5\dot{q}_2 = -0.25q_0\\
q_0(t) &= C_1\cos\frac{t}{2} + C_2\sin\frac{t}{2} = \cos\frac{t}{2}\\
q_2(t) &= \sin\frac{t}{2}
\end{aligned}
$$

这个解也很直观地表现出四元数的变化过程，但是这个结果跟式(1)的不同之处在于少了横滚角 ϕ 和偏航角 ψ，因此在大角度机动时根据式(1)
连续了，这不是四元数的问题，而是式(1)的问题。
根据式(1)计算出的错误的四元数变化曲线如下图所示。



直接使用四元数定义计算出正确的四元数变化曲线如下图所示。



出图代码如下，其中 `quaternions.py` 的代码见 自用的四元数、欧拉角、旋转矩阵转换代码。

```
import numpy as np
import matplotlib.pyplot as plt
from quaternions import *
```

```python
plottime, plotq0, plotq2 = [], [], []
for n1 in range(100):
    t = n1 * 0.1  # 角度从0到10弧度(4.7124弧度是-y轴)
    vecx = np.array([np.cos(t), 0, -np.sin(t)])  # 组成旋转矩阵的x轴坐标
    vecy = np.array([0, 1, 0])  # 组成旋转矩阵的y轴坐标
    vecz = np.array([np.sin(t), 0, np.cos(t)])  # 组成旋转矩阵的z轴坐标
    R = np.vstack([vecx, vecy, vecz]).T  # 旋转矩阵
    E = Rotation_to_Euler(R)  # 由旋转矩阵计算欧拉角
    # Q = Euler_To_Quaternion(E)  # 由欧拉角计算四元数（有问题的转换公式）
    Q = [np.cos(t/2), 0, np.sin(t/2), 0]  # 直接使用四元数定义（正确的结果）
    plottime.append(t)
```

目录