航空機力学第二　課題 1

航空宇宙工学科 3 年
学生記番号:03-170313
飯山敬大

2017/10/1

諸元は以下の通りとする (以下の迎角の単位は rad[s])
質量 $m = 7500[kg]$
翼面積 $S = 27.9[m^2]$
揚力係数 $C_L = C_{L\alpha}\alpha$ ($C_{L\alpha} = 4.30$)
抗力係数 $C_D = C_{D0} + K\alpha^2$ ($C_{D0} = 0.0548, K = 3.02$)
また各種記号については、プリントにならうものとする。

1

この航空機が高度 5000m(空気密度 $\rho = 0.74[kg/m^3]$), 速度 U=180[m/s] で水平定常直線飛行を行うときの迎角 $\alpha[deg]$ と推力 T[N] を求める
水平定常直線飛行を行うためには、

$$\dot{z_e} = -U\sin\gamma = 0 \tag{1}$$

$$\dot{U} = \frac{-D + T\cos\alpha}{m} - g\sin\gamma = 0 \tag{2}$$

$$\dot{\gamma} = \frac{1}{U}(\frac{L + T\sin\alpha}{m}\cos\Phi - g\cos\gamma) = 0 \tag{3}$$

の 3 式を満たす必要がある。ここで (1) より $\gamma = 0$ であり、また α が微小であると仮定すれば、$\cos\alpha = 1 - \frac{1}{2}\alpha^2$, $\sin\alpha = \alpha$ と置けるので、(2),(3) より

$$\dot{U} = -q_\infty S(C_{D0} + K\alpha^2) + T(1 - \frac{1}{2}\alpha^2) = 0 \tag{4}$$

$$\dot{\gamma} = -q_\infty SC_{L\alpha}\alpha + T\alpha - mg = 0 \tag{5}$$

ただし、$q_\infty = \frac{1}{2}\rho U^2$ である。
これを解くと、

$$\begin{cases} \alpha = 2.8879\,[deg] \\ T = 20921\,[N] \end{cases} \tag{6}$$

となる。

2

航空機が 1 の状態で 0-20[s] の間飛行し、以下の式の様なロール角制御を行うときの航空機の挙動を調べる

$$\Phi(t) = \begin{cases} 0 & (0 < t < 20) \\ \Phi_0 \sin\omega(t - 20[s]) & (20 \quad t \quad 60) \\ 0 & (60 < t < 80) \end{cases} \tag{7}$$

航空機についての 3 自由度方程式の 6 つの微分方程式についてルンゲクッタ法を用いて、航空機の飛行経路、速度、姿勢を計算する。
計算に用いたソースコードは以下の通りである

```
%matplotlib inline
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym

#parameter
m = 7500 #[kg]
s = 27.9 #[m^2]
cl_alpha = 4.30
cd_0 = 0.0548
k = 3.02
rho = 0.74 #[kg/m^3]
g = 9.806 #[m_s^2]
alpha = 0 #radian
thrust = 0

#array for results
t_array = []
u_array = []
gamma_array = []
psi_array = []
phi_array = []
xe_array = []
ye_array = []
ze_array = []

pi = 3.141592
rad = pi/180

def cal_alpha_thrust():
    global m,s,rho,g,cl_alpha,cd_0,k,alpha,thrust,pi,rad
    rad = pi / 180

    (alpha_s,thrust_s) = sym.symbols("alpha_s thrust_s")
    u = 180
    q = 0.5 * rho * u**2

    f1 = - q*s*(cd_0 + k*alpha_s**2) + thrust_s * (1 - alpha_s**2/2)
    f2 = q * s * cl_alpha * alpha_s + thrust_s*alpha_s - m*g
    res = sym.solve([f1,f2],[alpha_s,thrust_s])
    alpha = res[0][0]
    thrust = res[0][1]
    print("alpha=",alpha/rad,"[deg]")
    print("Thrust=",thrust,"[N]")

#roll
def phi(time):
```

```
    global m,s,rho,g,cl_alpha,cd_0,k,alpha,thrust,pi,rad
    if time < 20:
        return 0
    elif time < 60:
        omega = 2 * pi / 40
        rad = pi/180
        return 30 * rad * sym.sin(omega*(time-20))
    else:
        return 0

#velocity
def diff_u(t,u,gamma,psi):
    global m,s,rho,g,cl_alpha,cd_0,k,alpha,thrust,pi,rad
    q = 0.5 * rho * u**2
    drag = q * s * (cd_0 + k*alpha**2)
    diff_u = (-drag + thrust*sym.cos(alpha))/m - g * sym.sin(gamma)
    return diff_u

#route angle
def diff_gamma(t,u,gamma,psi):
    global m,s,rho,g,cl_alpha,cd_0,k,alpha,thrust,pi,rad
    if t < 20:
        return 0
    global cl_alpha,alpha
    q = 0.5 * rho * u**2
    lift = q * s * cl_alpha * alpha
    diff_gamma = ((lift + thrust*sym.sin(alpha))* sym.cos(phi(t))/m - g*sym.cos(gamma)) / u
    return diff_gamma

#yaw
def diff_psi(t,u,gamma,psi):
    global m,s,rho,g,cl_alpha,cd_0,k,alpha,thrust,pi,rad
    q = 0.5 * rho * u**2
    lift = q * s * cl_alpha * alpha
    diff_psi = ((lift + thrust*sym.sin(alpha))/m) * (sym.sin(phi(t))/sym.cos(gamma)) / u
    return diff_psi

def diff_xe(t,u,gamma,psi):
    diff_xe = u * sym.cos(gamma) * sym.cos(psi)
    return diff_xe

def diff_ye(t,u,gamma,psi):
    diff_ye = u * sym.cos(gamma) * sym.sin(psi)
    return diff_ye

def diff_ze(t,u,gamma,psi):
    diff_ze = -u * sym.sin(gamma)
    return diff_ze

def cal_flightroute():
    global m,s,u,rho,g,cl_alpha,cd_0,k,alpha,thrust,pi,rad
    global xe_array,ye_array,ze_array,t_array,u_array,gamma_array,psi_array,phi_array

    #initialize
    gamma = 0
    psi = 0 #yaw
    phi_data=0 #roll
    t=0
    dt = 0.5
    endt = 80.0
    xe = 0
    ye = 0
    ze = -5000

    #array for results
    t_array = [t]
```

```
u_array = [u]
gamma_array = [gamma]
psi_array = [psi]
phi_array = [phi_data]
xe_array = [xe]
ye_array = [ye]
ze_array = [-ze]

#replace names
x = u
y = gamma
z = psi

#array for rungekutta method
k0 = [0,0,0,0,0,0]
k1 = [0,0,0,0,0,0]
k2 = [0,0,0,0,0,0]
k3 = [0,0,0,0,0,0]

while t < endt:
    k0[0]=dt*diff_u(t,x,y,z)
    k0[1]=dt*diff_gamma(t,x,y,z)
    k0[2]=dt*diff_psi(t,x,y,z)
    k0[3]=dt*diff_xe(t,x,y,z)
    k0[4]=dt*diff_ye(t,x,y,z)
    k0[5]=dt*diff_ze(t,x,y,z)

    k1[0]=dt*diff_u(t+dt/2.0,x+k0[0]/2.0,y+k0[1]/2.0,z+k0[2]/2.0)
    k1[1]=dt*diff_gamma(t+dt/2.0,x+k0[0]/2.0,y+k0[1]/2.0,z+k0[2]/2.0)
    k1[2]=dt*diff_psi(t+dt/2.0,x+k0[0]/2.0,y+k0[1]/2.0,z+k0[2]/2.0)
    k1[3]=dt*diff_xe(t+dt/2.0,x+k0[0]/2.0,y+k0[1]/2.0,z+k0[2]/2.0)
    k1[4]=dt*diff_ye(t+dt/2.0,x+k0[0]/2.0,y+k0[1]/2.0,z+k0[2]/2.0)
    k1[5]=dt*diff_ze(t+dt/2.0,x+k0[0]/2.0,y+k0[1]/2.0,z+k0[2]/2.0)

    k2[0]=dt*diff_u(t+dt/2.0,x+k1[0]/2.0,y+k1[1]/2.0,z+k1[2]/2.0)
    k2[1]=dt*diff_gamma(t+dt/2.0,x+k1[0]/2.0,y+k1[1]/2.0,z+k1[2]/2.0)
    k2[2]=dt*diff_psi(t+dt/2.0,x+k1[0]/2.0,y+k1[1]/2.0,z+k1[2]/2.0)
    k2[3]=dt*diff_xe(t+dt/2.0,x+k1[0]/2.0,y+k1[1]/2.0,z+k1[2]/2.0)
    k2[4]=dt*diff_ye(t+dt/2.0,x+k1[0]/2.0,y+k1[1]/2.0,z+k1[2]/2.0)
    k2[5]=dt*diff_ze(t+dt/2.0,x+k1[0]/2.0,y+k1[1]/2.0,z+k1[2]/2.0)

    k3[0]=dt*diff_u(t+dt,x+k2[0],y+k2[1],z+k2[2])
    k3[1]=dt*diff_gamma(t+dt,x+k2[0],y+k2[1],z+k2[2])
    k3[2]=dt*diff_psi(t+dt,x+k2[0],y+k2[1],z+k2[2])
    k3[3]=dt*diff_xe(t+dt,x+k2[0],y+k2[1],z+k2[2])
    k3[4]=dt*diff_ye(t+dt,x+k2[0],y+k2[1],z+k2[2])
    k3[5]=dt*diff_ze(t+dt,x+k2[0],y+k2[1],z+k2[2])

    x=x+(k0[0]+2.0*k1[0]+2.0*k2[0]+k3[0])/6.0
    y=y+(k0[1]+2.0*k1[1]+2.0*k2[1]+k3[1])/6.0
    z=z+(k0[2]+2.0*k1[2]+2.0*k2[2]+k3[2])/6.0
    xe=xe+(k0[3]+2.0*k1[3]+2.0*k2[3]+k3[3])/6.0
    ye=ye+(k0[4]+2.0*k1[4]+2.0*k2[4]+k3[4])/6.0
    ze=ze+(k0[5]+2.0*k1[5]+2.0*k2[5]+k3[5])/6.0

    t_array.append(t)
    u_array.append(x)
    gamma_array.append(y/rad)
    psi_array.append(z/rad)
    phi_data = phi(t)
    phi_array.append(phi_data/rad)
    xe_array.append(xe)
    ye_array.append(ye)
    ze_array.append(-ze)
    t += dt
```

結果のグラフは以下の様になる

機体のルート及び速度は以下の様になる

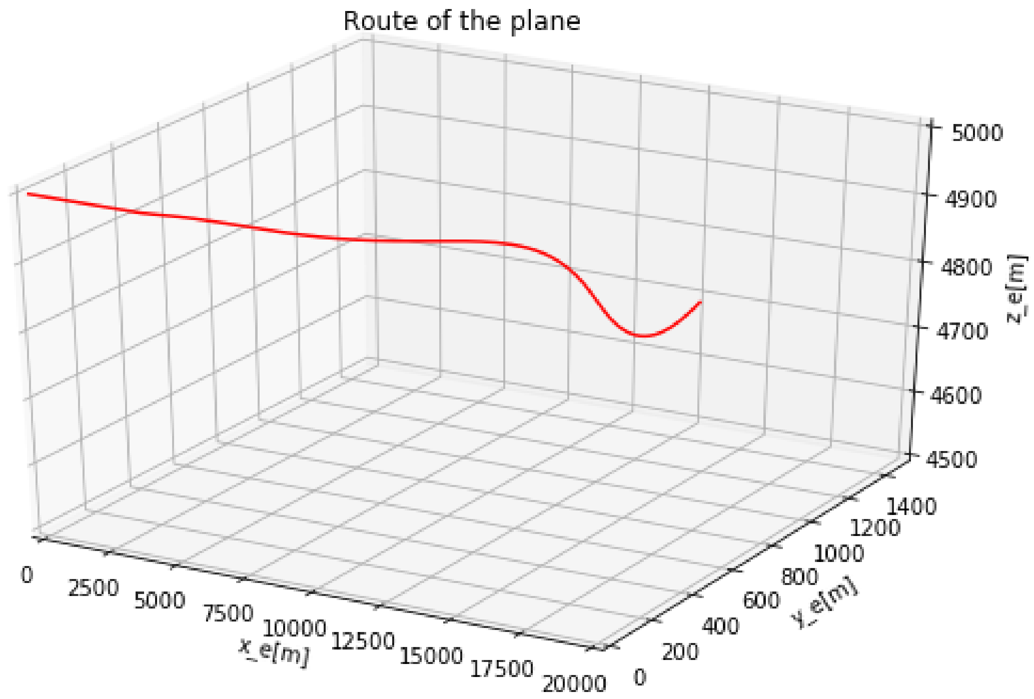

図 1　飛行経路

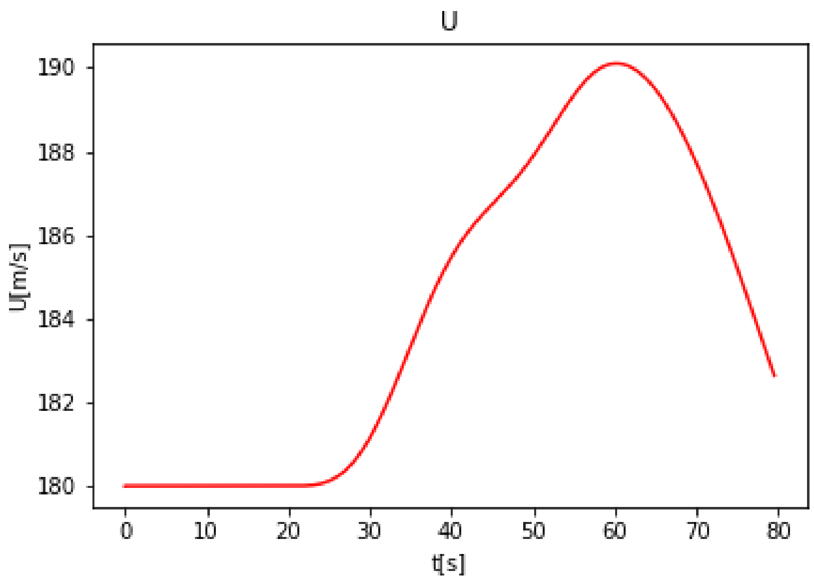

図 2　飛行速度 U(時間変化)

機体の位置の時間変化は以下のグラフの様になる

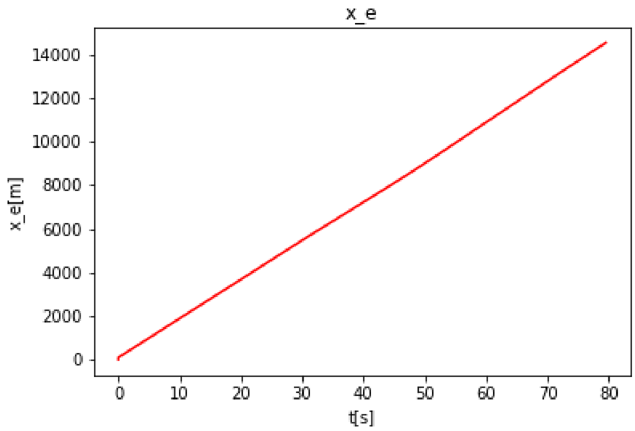

図 3　x_e(時間変化)

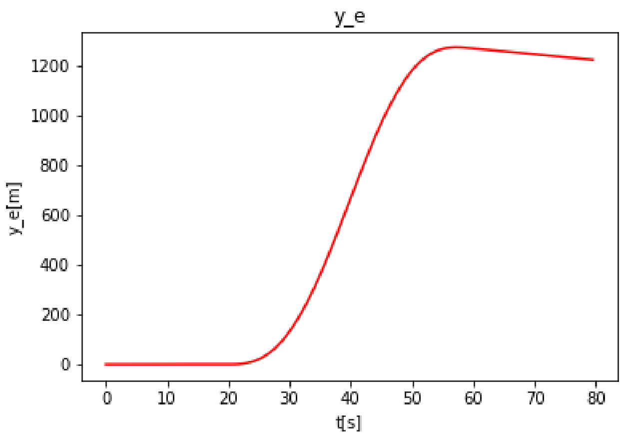

図 4　y_e(時間変化)

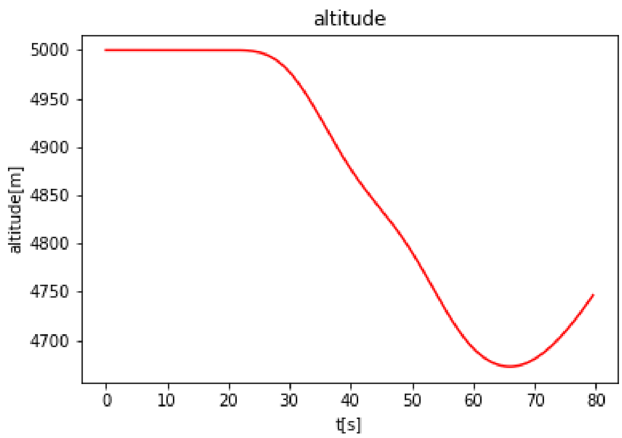

図 5　飛行高度

機体の姿勢は以下のグラフの様になる

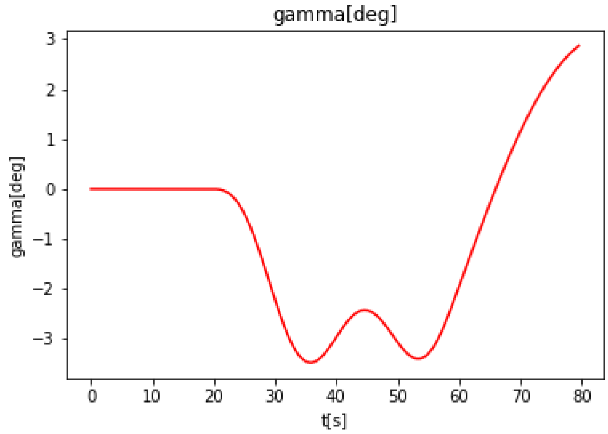

図 6　経路角 γ

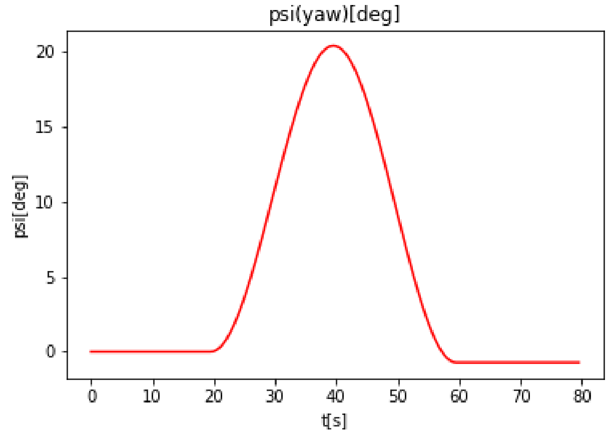

図 7　ヨー Ψ

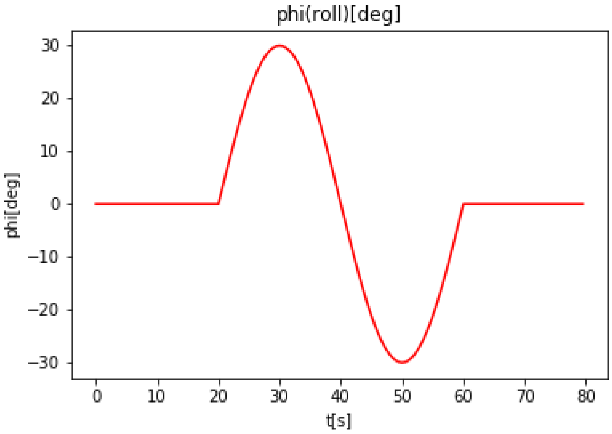

図 8　ロール Φ

3

最後に、上記の様なロール角制御を行った場合に、速度と高度を一定に保つために行うべき迎角と推力の制御について考える
高度を一定に保つためには、

$$\dot{z_e} = -U \sin\gamma = 0 \tag{8}$$

とならなくてはならないから、常に $\gamma = 0$ とならなくてはならない。
よって、速度と高度を一定に保つためには、

$$\dot{U} = \frac{-D + T\cos\alpha}{m} - g\sin\gamma = 0 \tag{9}$$

$$\dot{\gamma} = \frac{1}{U}(\frac{L + T\sin\alpha}{m}\cos\Phi - g\cos\gamma) = 0 \tag{10}$$

の 2 式が任意の時刻 t において成り立つ様に α と T を制御する必要がある。ここで (1) より $\gamma = 0$ であり、また α が微小であると仮定すれば、$\cos\alpha = 1 - \frac{1}{2}\alpha^2$, $\sin\alpha = \alpha$ と置けるので、(2),(3) より各時刻 t について、以下の 2 式を満たす α と T を求めて行けば良い。

$$\dot{U} = -q_\infty S(C_{D0} + K\alpha^2) + T(1 - \frac{1}{2}\alpha^2) = 0 \tag{11}$$

$$\dot{\gamma} = (-q_\infty SC_{L\alpha}\alpha + T\alpha)\cos\Phi(t) - mg = 0 \tag{12}$$

ただし、$q_\infty = \frac{1}{2}\rho U^2$ である。
計算には以下のソースコードを用いた

```
def cal_stable():
    global m,s,u,rho,g,cl_alpha,cd_0,k,pi
    stable_alpha = []
    stable_thrust = []
    stable_t = []
    t = 0
    dt = 1
    q = 0.5 * rho * u**2
    (alpha_s,thrust_s) = sym.symbols("alpha_s thrust_s")
    while t < 80:
        f1 = - q*s*(cd_0 + k*alpha_s**2) + thrust_s * (1 - alpha_s**2/2)
        f2 = (q * s * cl_alpha * alpha_s + thrust_s*alpha_s)* sym.cos(phi(t)) - m*g
        res = sym.solve([f1,f2],[alpha_s,thrust_s])
        alpha = res[0][0]
        thrust = res[0][1]
        stable_t.append(t)
        stable_alpha.append(alpha/rad)
        stable_thrust.append(thrust)
        t += dt
```

結果は以下の図の様になる

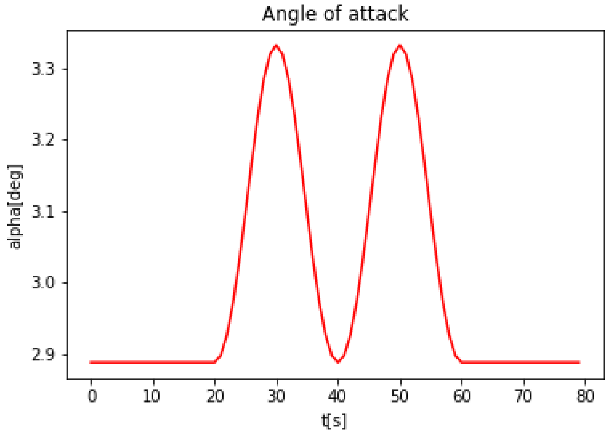

図 9　迎角 α[deg]

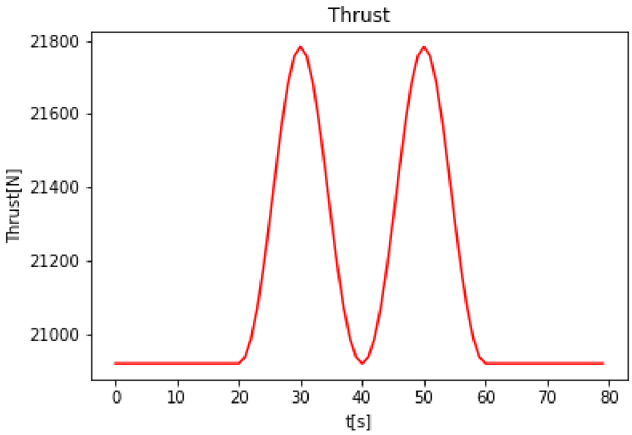

図 10　推力 T[N]