```
#bum
import numpy as np
fis = open("Ndaten.txt", "r")
dat = []
for i in range(4):
    dat.append(fis.readline())
fis.close()
for i in range(4):
    dat[i] = dat[i][0:len(dat[i])-1]
    dat[i] = tuple([float(x) for x in dat[i].split(",")])
ep,r = dat[0]
rmh,ra,c = dat[1]
rt,et,zet,a,b = dat[2]
w1,w2,tet,phi = dat[3]
ws = np.array([w1,w2,0])
w = np.array([1.57,tet,phi,0,0,-rt*et/r])
x = np.array([0,0,0,0,et,zet])
print("rotatii/s"+str(w[5]/(2*np.pi)))
print(c)
ans = input("model? (E/P)")
lm = 0
if ans == "P" :
    cf = input("aerodyn.coef.")
    cf = float(cf)
ap = input("nutation inf.lim.")
ap = float(ap)
te = np.array([[np.cos(ep),0,-np.sin(ep)],
               [0,1,0],
               [np.sin(ep),0,np.cos(ep)]])
if ans == "P" :
    te = np.eye(3)
ta = [np.eye(3)]
tp = np.array([[-0.5,-0.5*np.sqrt(3),0],
               [0.5*np.sqrt(3),-0.5,0],
               [0,0,1]])
ta.append(tp)
ta.append(tp@tp)
tm = np.array([[a*a+3*r*r,0,0],
               [0,b*b,0],
               [0,0,a*a+b*b+3*r*r]])
tn = te@tm@(te.transpose())
tn = tn + ta[1]@tn@(ta[1].transpose()) + ta[2]@tn@(ta[2].transpose())
def m(z):
    p,t,f = tuple(z.tolist())
    m1 = np.array([[np.cos(f),np.sin(f),0],
                   [-np.sin(f),np.cos(f),0],
                   [0,0,1]])
    m2 = np.array([[1,0,0],
                   [0,np.cos(t),np.sin(t)],
                   [0,-np.sin(t),np.cos(t)]])
    m3 = np.array([[np.cos(p),np.sin(p),0],
                   [-np.sin(p),np.cos(p),0],
                   [0,0,1]])
    rez = m1@m2@m3
    return rez
def ha(z):
    p,t,f = tuple(z.tolist())
    rez = np.array([[np.sin(t)*np.sin(f),np.cos(f),0],
                    [np.sin(t)*np.cos(f),-np.sin(f),0],
                    [np.cos(t),0,1]])
    return rez
def av(z):
    rez = z[3:6]@(ha(z[0:3]).transpose())
    return rez
```

```
def hi(z):
    global i
    if np.sin(z[1]) != 0 :
        rez = np.linalg.inv(ha(z))
    else :
        i = nu
        print("zero nutation error")
    return rez
def v(y,z):
    j = np.array([0,1,0])
    re = -(y-ws)@(m(z[0:3]).transpose())
    rez = re-r*np.cross(av(z),j)
    rez = np.vstack((rez,re-r*np.cross(av(z),j@(ta[1].transpose()))))
    rez = np.vstack((rez,re-r*np.cross(av(z),j@(ta[2].transpose()))))
    return rez
def ac(y,z):
    global i
    global lm
    global em
    global lp
    rez = []
    for k in range(3):
        vi = v(y,z)[k,:]
        if ans == "P" :
            ut = np.array([0,0,1])
            vt = vi-np.inner(vi,ut)*ut
            vh = vt/np.linalg.norm(vt)
            if np.inner(vi,np.array([1,0,0])@(ta[k].transpose())) > 0 :
                re = 0.5*cf*ra*np.inner(vt,vt)*ut
                re = re + np.linalg.norm(re)*np.tan(ep)*vh
                re = re/(np.sqrt(1-np.inner(vt,vt)/(c*c))*rmh)
                rez.append(re)
            else :
                rez.append(np.array([0,0,0]))
                if lm == 0 :
                    print("annulation")
                    lm = 1
                    print(i)
    return rez
wo = w
xo = x
l = 0
i = 0
nu = input("iteration number nu")
nu = int(nu)
sp = input("angular spacing nr. sp")
sp = float(sp)
tm = 0
la = 0
le = 0
pa = 0
pe = 0
sa = 0
su = 0
lo = 0
em = 0
lp = 0
xc = np.empty((0,6))
wc = np.empty((0,6))
while (l == 0 or x[2] > 0) and i < nu :
    vu = np.amax(np.linalg.norm(v(x[3:6],w),axis = 1))
    if vu >= c :
        print("over Mach")
        print(vu)
```

```
    print(i)
    i = nu
am = np.zeros(3)
bm = np.zeros(3)
ai = ac(x[3:6],w)
ko = w[3:6]@(ha(w[0:3]).transpose())@tn
for q in range(3):
    if i < nu :
        am = am+3*r*np.cross(np.array([0,1,0])@(ta[q].transpose()),ai[q])
        bm = bm+(1/3)*ai[q]
if i == 0 :
    print(ac(x[3:6],w))
    print(bm)
bm = bm@m(w[0:3])-9.8*np.array([0,0,1])
if i == 0 :
    print("bm"+str(bm[0]))
    print(bm)
s = abs(2*np.pi/sp)
s = s/w[5]
tu = np.linalg.inv(tn)
kon = ko + s*am
ko = (np.linalg.norm(ko)/np.linalg.norm(kon))*kon
if ep != 0 :
    ko = (np.sqrt(np.inner(ko,kon+s*am))/np.linalg.norm(kon))*kon
tm = tm + s
if i == 100 or i == 0 :
    print("step "+ str(s))
xid = x[3:6] + s*bm
wi = w[0:3] + s*w[3:6]
kot = ko@tu
if abs(np.sin(wi[1])) >= ap :
    wid = kot@(hi(wi).transpose())
else :
    hd = np.array([[0,0,1/np.cos(w[1])],
                   [np.cos(w[2]),-np.sin(w[2]),0],
                   [0,0,0]])
    wid = kot@(hd.transpose())+np.array([-w[5],0,w[5]])
xi = x[0:3] + s*x[3:6]
w = np.append(wi,wid)
x = np.append(xi,xid)
wo = np.vstack((wo,w))
xo = np.vstack((xo,x))
if w[5] <= 0 :
    print("rotation revers")
    print(i)
    i = nu
if i > 10 :
    l = 1
ui = abs(np.inner(x[3:6],np.array([0,1,0])))/np.linalg.norm(x[3:6])
if ui < 0.01 and (la != 1 or le != 1) :
    if np.inner(x[3:6],np.array([1,0,0])) < 0 and la != 1 :
        print("paralel1"+str(i))
        print(x)
        print(w)
        la = 1
        le = 0
        if sa == 2 or sa == 3 :
            if x[1] < 0 :
                lo = 1
            sa = sa+1
            xc = np.vstack((xc,x))
            wc = np.vstack((wc,w))
    if np.inner(x[3:6],np.array([1,0,0])) > 0 and le != 1 :
        print("paralel2"+str(i))
```

```
            print(x)
            print(w)
            le = 1
            la = 0
            if sa == 1 or sa == 3 :
                sa = sa+1
                xc = np.vstack((xc,x))
                wc = np.vstack((wc,w))
            if sa == 4 :
                lo = 1
            if sa == 2 :
                il = i
                ib = np.linalg.norm(xo[i,0:2])
    if abs(x[1]) < 0.1 and (pa != 1 or pe != 1) :
        if np.inner(x[3:6],np.array([0,1,0])) < 0 and pa != 1 :
            print("zero1"+str(i))
            print(x)
            print(w)
            pa = 1
            pe = 0
            if sa == 0 or sa == 4 :
                sa = sa+1
                xc = np.vstack((xc,x))
                wc = np.vstack((wc,w))
        if np.inner(x[3:6],np.array([0,1,0])) > 0 and pe != 1 :
            print("zero2"+str(i))
            print(x)
            print(w)
            pe = 1
            pa = 0
            if sa == 2 or sa == 3 :
                if sa == 3 and np.inner(x[3:5],np.array([0,x[1]])) > 0 :
                    lo = 1
                sa = sa+1
                xc = np.vstack((xc,x))
                wc = np.vstack((wc,w))
    i = i+1
    ik = i
    if (sa == 5 or lo == 1) and i <= nu and su == 1 :
        ik = i
        i = nu
    if (x[2] <= 0 or i >= nu) and su == 0 :
        anp = input("red.loop?(Y/N)")
        if anp == "Y" :
            print("start red. loop")
            i = 0
            x = xc[0,:]
            w = wc[0,:]
            su = 1
            l = 0
            lo = 0
            la = 0
            le = 0
            pa = 0
            pe = 0
            tm = 0
            sa = 0
            xo = x
            wo = w
            print("init.cond.reduced loop")
            print(x)
            print(w)
            uw = np.floor(w[0:3]/(2*np.pi))
            uk = w[0:3]-2*np.pi*uw
```

```
            print(uk*180/np.pi)
            print("rot/s"+str(w[5]/(2*np.pi)))
    if x[2] <= 0 or i > nu :
        ik = i
if ik == nu :
    print("limit of iterations")
print("final")
print(xo[len(xo)-1])
print(wo[len(wo)-1])
print("iterations")
print(ik)
xo = np.float16(xo)
wo = np.float16(wo)
print(np.amax(np.linalg.norm(xo[0:100,0:3],axis = 1)))
if su == 1 :
    mo = np.linalg.norm(xo[il:,0:2], axis = 1)
    mn = np.amin(mo)
    mi = np.where(mo == mn)
    mx = mi[0][0]
    print("dist. min "+str(mn)+ "index"+ str(mx))
    print(xo[mx+il])
    print("dist.max"+str(np.amax(np.linalg.norm(xo[:,0:2], axis = 1))))
    print("return ind."+str(il))
    print(xo[il])
print(vu)
print("maxx"+ str(np.amax(xo[:,0])))
print("minx"+ str(np.amin(xo[:,0])))
print("maxy"+ str(np.amax(xo[:,1])))
print("miny"+ str(np.amin(xo[:,1])))
print([np.sin(w[1])*np.sin(w[2]),np.sin(w[1])*np.cos(w[2]),np.cos(w[1])])
print("time"+str(tm))
print(xo[len(xo)-2])

# x - position and velocity of mass center in the fix reference frame
# w - Euler angles and angular velocities of the body tied reference frame
# m - transformation matrix from fix reference frame to body tied reference
frame
# av - angular velöcity vector coordinates in the body tied reference frame
# v - velocities coordinates vectors in the body tied reference frame for each
wing
# am - resultant momentum of forces coordinates in the body tied reference frame
after simplification with 8*a*b*rmh/3
# bm - resultant force after simplification with 24*a*b*rmh
# a,b,rmh - dimensions and density x thickness of the wings
# ta - rotations matrixes to position of each wing in the body tied reference
frame
# tn - moment of inertia tensor after simplification with 8*a*b*rmh/3
# We put in Ndaten.txt the following values : ep = 0.17 ; r = 0.2 ; rmh = 1.31 ;
ra = 1.225 ; c = 334 ;
# a = b = 0.06 ; rt = 1 or rt = 1.1 ; et = -30 ; zet = 2.5 ; w1 = 0 ; w2 = 0 ;
phi = 1.55 ; tet = -0.034
# Also we input cf = 0.2 ; ap = 0.01 ; nu = 40000 ; sp = 40
# The measurement units considered for these parameters are the SI units and the
angles are given in radians
# Note that the mass of the boomerang which results from the referential
dimensions of the wings is
# approximative 86 g
```