```
from mpmath import *
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

C =299792458  # speed of light
C2=power(C,2) # square of speed of light
M =1.0e+9     # 1000000 tons each spaceship
A0=9.8        # desired (comfortable) proper acceleration
L0=10000.0    # length of the thread/distance considered

# mechanical thread characteristics...
# Diameter: 1 mm  Area:0.6 mm2  Break tension: 84 Kg
E =1.3e+11    # 130000 N/mm2 Young modulus
AR=6.0e-7     # 1 mm Diameter / 0.6 mm2 section Area

DIAs=24*3600
Tc=mpf(0.0)

# Earth frame A observer ct,x coordinates for constant proper
accelerating bodies
# tau: proper time  acc: proper acceleration considered

# C (trailing) spaceship... 0.5*L0 behind Center of Mass
def ct1x1(tau,acc):
    return (C2/acc)*sinh(acc*tau/C),(C2/acc)*cosh(acc*tau/C)-0.5*L0

# Center of Mass
def ctcxc(tau,acc):
    return (C2/acc)*sinh(acc*tau/C),(C2/acc)*cosh(acc*tau/C)

# B (leading) spaceship... 0.5*L0 ahead Center of Mass
def ct2x2(tau,acc):
    return (C2/acc)*sinh(acc*tau/C),(C2/acc)*cosh(acc*tau/C)+0.5*L0

# used to find intersection of simultaneity line from t1 in CM to
corresponding t2 in B spaceship
tc11 = lambda tc, t1, k: (C2/A0)*(k*sinh(A0*tc/C)-sinh(A0*t1/C))
tc12 = lambda tc, t1, k: (C2/A0)*(k*cosh(A0*tc/C)-
cosh(A0*t1/C))+(0.5*L0)
tc13 = lambda tc, t1, k: tc-Tc

# used to find intersection of simultaneity line from t1 in CM to
corresponding t2 in C spaceship
tc21 = lambda tc, t2, k: (C2/A0)*(k*sinh(A0*tc/C)-sinh(A0*t2/C))
tc22 = lambda tc, t2, k: (C2/A0)*(k*cosh(A0*tc/C)-cosh(A0*t2/C))-
(0.5*L0)
tc23 = lambda tc, t2, k: tc-Tc

# Is not possible to find intersection of simultaneity lines with
standard float precision
mp.dps = 32

def VelocityCM(tauc):
    global Tc
    Tc=mpf(tauc)
```

```
    # Given proper time tauc in CM, find corresponding simultaneous
proper time in each spaceship
    tc,t1,k=findroot([tc11,tc12,tc13],(Tc,Tc,1.0))
    tc,t2,k=findroot([tc21,tc22,tc23],(Tc,Tc,1.0))

    ctc,xc=ctcxc(tc,A0)
    ct1,x1=ct1x1(t1,A0)
    ct2,x2=ct2x2(t2,A0)

    # proper (=real) distance from CM to...
    lc2=sqrt(power(x2-xc,2)-power(ct2-ctc,2))
    lc1=sqrt(power(x1-xc,2)-power(ct1-ctc,2))

    # With the proper times, calculate the speeds relative to Earth
Frame
    vc=C*tanh(A0*tc/C)
    v1=C*tanh(A0*t1/C)
    v2=C*tanh(A0*t2/C)

    # Using relativistic speed addition, calculate de relative speeds
for CM observer
    v1c=(vc-v1)/(1-(vc*v1/C2))
    v2c=(v2-vc)/(1-(vc*v2/C2))

    return v1c,v2c,t1,t2,lc1,lc2


def DifSystem(bt):
    def Eq(u,t):
        if (u[0]>0):
            return (u[1],A0*(1.0-(C2/(C2+A0*(0.5*L0+u[0])))))-
(E*AR*(2*u[0])/(M*L0)))
        else:
            # no tension, thread is loose!
            return (u[1],A0*(1.0-(C2/(C2+A0*(0.5*L0+u[0])))))

    v1c,v2c,t1,t2,lc1,lc2 = VelocityCM(bt)

    print('Lc_c=%f  Lc_b=%f'%(lc1,lc2))

    ts = np.linspace(0.0, 7*DIAs, 100001)
    us = odeint(Eq, [0.0,v2c], ts)

    ys = us[:,0]

    plt.plot(ts,ys,'-')

    plt.xlabel('Time (s)')
    plt.ylabel('Length (m)')
    plt.title('Brake at %4.2f s    vB-CM=%4.2f
mm/hour'%(t1,v2c*1000*3600))
    plt.grid()

    plt.show()

DifSystem(1*DIAs)
```