UNIVERSIDAD
POLITÉCNICA
DE MADRID

Dinámica de reentrada

Aerodinámica de Altas Velocidades y Fenómenos de Reentrada
Máster Universitario en Sistemas Espaciales (IDR-UPM)

Autor: Andrés Pedraza Rodríguez

Profesor: Sebastián Nicolás Franchini Longhi

Madrid, 28 de mayo, 2021

Resumen

El objetivo del presente trabajo es la implementación de las ecuaciones de la dinámica de reentrada de una cápsula mediante un esquema numérico: Runge-Kutta 4 embebido. Mediante este código se simularán posibles reentradas de la cápsula de la misión Vostok-1 para cumplir con los requisitos de carga aerodinámica necesaria para asegurar la supervivencia del cosmonauta Yuri Gagarin. Finalmente se seleccionará un ángulo de entrada de 5° y una eficiencia aerodinámica de 0.1, se estudiará la trayectoria final y se compararán los resultados con las soluciones analíticas.

Índice

Índice de figuras

Índice de tablas

1. Introducción

Para que un vehículo sea puesto en órbita ha de alcanzar velocidades suficientemente altas como para vencer el campo gravitacional del planeta que tenga debajo. Esto significa que, a la hora de realizar una reentrada, el vehículo tendrá que perder la energía cinética anteriormente conferida y esta energía ha de perderse en forma de calor. Además, la reentrada supone una serie de aceleraciones que ha de soportar la carga de pago, que en la mayoría de casos suelen ser humanos. También ha de considerarse la trayectoria que ha de seguir el vehículo para no aterrizar en un lugar inadecuado o para poder luego ser rescatado.

Es por ello que es necesario un estudio detallado de la dinámica de la reentrada, estudio, que suele llevarse acabo mediante la integración numérica de las ecuaciones de la dinámica. En este caso se realizará un estudio sencillo en el que se empleará un esquema de integración Runge-Kutta 4 embebido. Así mismo se comparará la solución obtenida con la predicha por los modelos analíticos de reentrada balística y en planeo. Este estudio tomará de base las características de la cápsula Vostok-1, la primera en llevar un ser humano al espacio y traerlo de vuelta. Huelga decir que el código desarrollado es capaz de realizar muchos más análisis (como por ejemplo reentradas en otros planetas con eficiencias aerodinámicas y coeficientes balísticos dependientes del ángulo de vuelo) y la iteración de sus parámetros es sencilla, por lo que se puede usar en conjunto con un algoritmo de optimización. Sin embargo, se ha seleccionado este caso a modo de ejemplo. En [1] existen otros ejemplos.

2. Metodología

Para la realización de este trabajo se ha hecho uso de la información facilitada a través del aula virtual y de los apuntes de la asignatura *Aerodinámica de Altas Velocidades y Fenómenos de Reentrada*, impartida en el Máster Universitario en Sistemas Espaciales (IDR-UPM).

El algoritmo de integración numérica se ha desarrollado en Python haciendo uso de librerías externas como NUMPY, para tratar con los vectores de estado y como MATPLOTLIB para obtener los gráficos deseados.

Las caracteríasticas de la Vostok-1 se han obtenido de [2] y se han hecho una serie de simplificaciones para un estudio más sencillo.

3. Vostok-1

Vostok 1 fue el primer cohete espacial del Programa Vostok y la primera misión espacial tripulada del programa espacial soviético. En ella el cosmonauta Yuri Gagarin habría de realizar un vuelo de 108 minutos a una altitud media de 315 km y con una velocidad entorno a los 7823.2 m/s (véase [3] y [4])

Al estar tripulada, a la hora de la reentrada, las máximas aceleraciones posibles a las que podía verse sometida la cápsula eran de entre 8 y 9 g, por lo que la cápsula habría de contar con una eficiencia aerodinámica mayor que cero para aliviar las aceleraciones típicas de una entrada balística. En este trabajo se analizarán diferentes valores de eficiencia aerodinámica para cumplir con este requisito.

En cuanto al coeficiente balístico la nave contaba con un peso al aterrizaje de 2400 kg lo que junto con sus 2.3 m de diámetro y un coeficiente de resistencia aerodinámica que, en vista de la geometría de la nave, que se muestra en la Figura 1, no habrá de ser muy distinto al de una esfera en régimen hipersónico. Aplicando el método de inclinación local de Newton modificado se tiene que la esfera habrá de presentar un coeficiente de resistencia entorno a 0.9197.

Por último, la simulación será llevada a cabo hasta una altitud de 7 km ya que es cuando Gagarin fue eyectado de la nave.

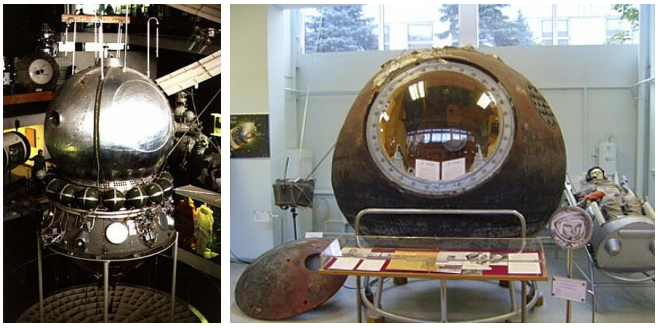

Figura 1: Cápsula de la misión Vostok-1 antes del lanzamiento (izquierda) y después del aterrizaje (derecha).

En la Tabla 1 se recogen los datos que serán empleados para la simulación.

Tabla 1: Parámetros empleados en la simulación extraídos a partir de las características de la cápsula de la misión Vostok-1.

Coeficiente balístico			
Coeficiente de resistencia aerodinámica	Diámetro [m]	Masa [kg]	Coeficiente balístico
0.9197	2.3	2,400	628.0851
Eficiencia aerodinámica			
Entre 0 y 0.5 (para no superar una carga de 9 g)			

Condiciones iniciales	
U_0	7823.2 m/s
γ_e	Entre 3.15° y 7.5°
z_0	315 km
θ_0	0 °

4. Planteamineto matemático

4.1. Dinámica de la reentrada

Las ecuaciones necesarias para describir el comportamiento dinámico de un vehículo en la reentrada están dados por el siguiente conjunto de ecuaciones diferenciales con una posición planteada en ejes polares y una trayectoria referenciada a ejes cuerpo (véase Figura 2):

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -g(z)\cdot\sin\gamma - D(z,u), \tag{1}$$

$$\frac{\mathrm{d}\gamma}{\mathrm{d}t} = \frac{1}{u}\left(L(z,u) - g(z)\cdot\cos\gamma + \frac{u^2}{r}\cdot\cos\gamma\right), \tag{2}$$

$$\frac{\mathrm{d}r}{\mathrm{d}t} = u\cdot\sin\gamma, \tag{3}$$

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = \frac{1}{r}u\cdot\cos\gamma. \tag{4}$$

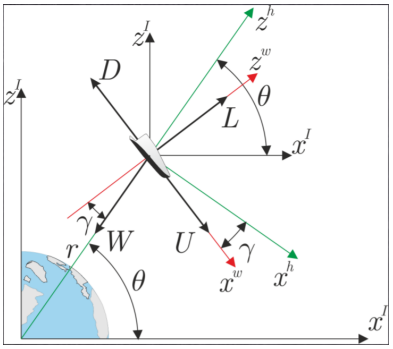

Figura 2: Representación de las variables y sistemas de referencia que intervienen en las ecuaciones.

donde u es la velocidad de vuelo, *gamma* es el ángulo de trayectoria (*flying path angle*), r es la coordenada radial desde el centro de la Tierra, z es la altitud y *theta* es la coordenada angular (véase Figura 2. Las funciones de densidad $\rho(z)$ y gravedad $g(z)$ se analizarán con más detalle en adelante. Por último, los parámetros L y D vienen dados por:

$$L(z,u) = \frac{\rho(z)u^2E}{2\beta},$$

$$D(z,u) = \frac{\rho(z)u^2}{2\beta},$$

donde a su vez, los coeficientes de eficiencia aerodinámica E y balístico β vienen dados por:

$$\beta = \frac{m}{Sc_D}\ ,$$

$$E = \frac{c_L}{c_D}\ .$$

Además se estudiarán otras variables de interés como el factor de carga:

$$n = -\frac{\mathrm{d}U}{\mathrm{d}t} * \frac{1}{g_0} \tag{5}$$

Para la comprobación del código se hará uso de las soluciones analíticas de reentrada balística:

$$
\begin{aligned}
B &= \frac{z_s \rho_0}{2\beta \sin\gamma_e} \,, \\
C &= \frac{u_e^2 \rho_0}{2\beta g_0} \,, \\
\frac{u}{u_e} &= \exp\left(B \cdot \exp\left(-\frac{z}{z_e}\right)\right) \,, \\
n &= C \cdot \exp\left(2B \cdot \exp\left(-\frac{z}{z_e}\right) - \frac{z}{z_s}\right) \,.
\end{aligned}
$$

y de reentrada con sustentación:

$$
\begin{aligned}
\frac{u}{u_e} &= \left(\sqrt{1 + \frac{E R_T \rho(z)}{2\beta}}\right)^{-1} \,, \\
n &= \frac{1}{E}\left[1 - \left(\frac{u}{u_e}\right)^2\right] = \left(\sqrt{E + \frac{2\beta \cdot \exp\left(-\frac{z}{z_e}\right)}{\rho_0 R_T}}\right)^{-1} \,.
\end{aligned}
$$

4.2. Esquema de integración

A partir de las ecuaciones 2 a 4 se ha propuesto un esquema de integración numérica Runge-Kutta 4 embebido. Este esquema, además de proporcionar unos resultados por lo general muy precisos y con rapidez, varía el paso temporal de manera que el error cometido por el mallado se sitúe por debajo de un error establecido. El esquema Runge-Kutta 4 consiste en:

```
    def RK4(self,Dt,Y):

        F = self.F

        K1 = F(Y)
        K2 = F(Y+Dt/2 * K1)
        K3 = F(Y+Dt/2 * K2)
        K4 = F(Y+Dt   * K3)

        return Y + Dt/6 * (K1+2*K2+2*K3+K4)
```

donde la función F viene determinada por las ecuaciones de la reentrada:

```
    def F(self,Y):

        u       = Y[0]
        gamma   = Y[1]
        r       = Y[2]
        theta   = Y[3]

        z = r-self.RT

        g           = self.g_fun(z)
        rho         = self.rho_fun(z)
        beta_val    = self.beta(gamma)
        E_val       = self.E(gamma)

        eqns  = [-g*sin(gamma)-rho/(2*beta_val) * u**2,
        1/u * (u**2 * (rho*E_val/(2*beta_val) + cos(gamma)/r) - g*cos(gamma)),
        u*sin(gamma),
        1/r * u*cos(gamma)]

        return np.array(eqns)
```

El salto temporal necesario se calcula a partir del error de truncamiento local y el orden del esquema:

```
# Cálculo del siguiente estado
Y1 = RK4(Dt,Y)
Y21 = RK4(Dt/2,Y)
Y2 = RK4(Dt/2,Y21)

# Cálculo del error
err = Y1-Y2
err[0] = err[0]
err[1] = tan(err[1]/2)*1e3 # Los errores angulares se penalizan más
err[2] = err[2]
err[3] = tan(err[3]/2)*1e3
mag = np.sqrt(err.dot(err))
if mag < 1e-12:
    Dt_new = 1e12
else:
    Dt_new = Dt*(self.epsilon/mag)**(1/(4+1))
```

El error se obtiene de la diferencia entre la solución y la solución obtenida con un mallado el doble de fino. Como se puede observar, los errores asociados al ángulo están penalizados con mayor severidad ya que resultan mucho más críticos.

Si el salto temporal necesario calculado es menor al del paso dado se repite el cálculo con un salto más pequeño. Si por el contrario el salto temporal ha sido demasiado grande se aumenta en 0.01 s para el siguiente paso de integración.

```
if Dt <= Dt_new:
    Y   = Y1
    dY  = F(Y)
    t   = t+Dt
    k   = k+1
    self.y  = np.vstack([self.y,Y])
    self.dy = np.vstack([self.dy,dY])
    self.t  = np.append(self.t,t)

    Dt = Dt+0.01
elif Dt > Dt_new:
    Dt = Dt_new
else:
    Dt = 0.1
```

Así se tiene un esquema rápido en los tramos de menor complejidad numérica y preciso en aquellas zonas donde se necesita. El error máximo admisible se puede especificar mediante:

```
Vostok = mision()
Vostok.epsilon = 0.1
```

La condición de parada por defecto es altitud nula pero se puede especificar mediante la coordenada radial o la temporal. Si la cápsula se sitúa por encima de 100 km de su altitud inicial el programa determina que ha rebotado y se para.

```
Vostok = mision()
u, gamma, r, theta = Vostok.reentrada([Ue,gamma_e,z0,theta0],rfin=Vostok.RT+7e3)
u1, gamma1, r1, theta1 = Vostok.reentrada([Ue,gamma_e,z0,theta0],tfin=1200)
```

En caso de que las iteraciones conduzcan a un número demasiado grande de pasos la simulación se detiene. Este número de pasos máximos se puede modificar

```
Vostok = mision()
Vostok.k_lim = 1e5
```

4.3. Densidad y gravedad

La atmósfera juega un papel crucial a la hora de determinar el comportamiento de la cápsula en la reentrada y además esta puede ser muy cambiante. Es por ello que el código desarrollado permite incluir cualquier función de la densidad en función de la altura mediante la modificación de los atributos "g_fun" y "rho_fun":

```
Vostok = mision()
from isa import isa_rho
Vostok.rho_fun = isa_rho
Vostok.g_fun = lambda z: 9.8 * (6.371e6/(z+6.371e6))**2
```

Las funciones predeterminadas del código son:

```
    def G(self,z):

        g0 = self.g0
        RT = self.RT
        g = g0 * (RT/(z+RT))**2

        return g

```

```
9     def RHO(self,z):
10
11        RT = self.RT
12        if z <= 150e3: #m
13            rho0 = 1.225 #kg/m^3
14            z0 = 7.524e3 #m
15        elif z> 150e3: #m
16            rho0 = 3.875e-9 #kg/m^3
17            z0 = 59.06e3 #m
18
19        try:
20            rho = rho0 * exp(-z/z0)
21        except:
22            rho = 1000 #kg/m^3
23        return rho
```

En A.3 se muestra el código empleado para la modelización de la atmósfera. Sin embargo, estudios más detallados necesitarán de modelos dinámicos atmosféricos los cuales no pueden ser implementados en el código actual a no ser que se conozca la velocidad de la cápsula en función de la altura y se use la altitud como parámetro de entrada o se modifique ligeramente el código A.2 allá donde se haga la llamada a "self.rho_fun" y "self.g_fun" especificando la entrada de las variables deseadas, por ejemplo:

```
1 # rho = self.rho_fun(z)
2 rho = self.rho_fun(z,u)
```

5. Resultados

5.1. Ángulo de entrada

A la hora de determinar la eficiencia aerodinámica solo ciertos valores de γ_0 conducen a una reentrada satisfactoria. Si el valor de γ_0 es demasiado alto, la nave se precipitará con mucha velocidad sobre la Tierra y probablemente se desintegrará en la atmósfera. Si por el contrario el ángulo es demasiado pequeño la cápsula rebotará en la atmósfera.

En la Figura 3 se muestra la influencia que tiene el ángulo γ sobre la evolución de la altitud de distintas cápsulas con distintas eficiencias aerodinámicas. Como se puede observar para un ángulo de unos 3.15° la cápsula es muy susceptible de rebotar por lo que se debe escoger un ángulo algo mayor a este valor.

En la Figura 4 se muestra el factor de carga máximo que alcanzan dichas cápsulas en función de la altitud para distintos ángulos de entrada. Como se puede observar, para valores de γ mayores a 5° resulta especialmente difícil acotar las aceleraciones máximas por debajo de lo que el cosmonauta es capaz de resistir.

En vista de los resultados obtenidos se ha seleccionado un ángulo de reentrada de 5° ya que proporciona aceleraciones acotadas dentro de lo establecido para casi todas las eficiencias (de forma que un fallo en el control de actitud no se convierte en un fallo crítico) y se aleja de el ángulo de reentrada crítico lo suficiente como para asumir cierta incertidumbre respecto a esta medida.

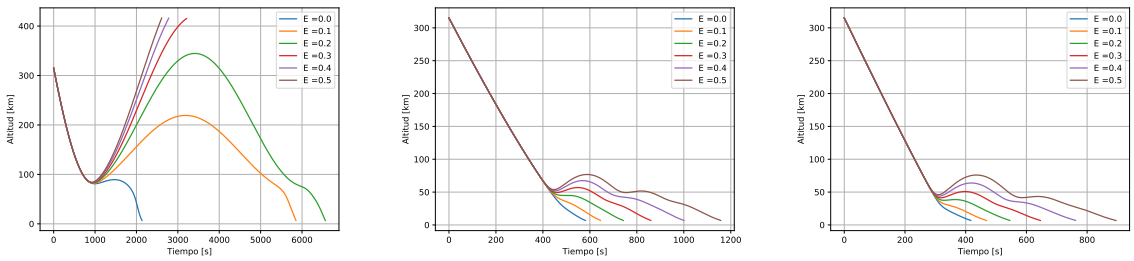

Figura 3: Evolución de la altitud con el tiempo para tres ángulos de entrada: 3.15°(izquierda), 5°(centro) y 7.5°(derecha)

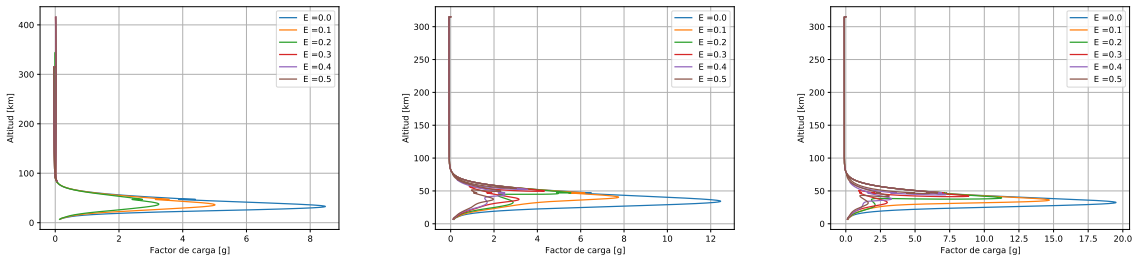

Figura 4: Evolución del factor de carga según la altitud para tres ángulos de entrada: 3.15°(izquierda), 5°(centro) y 7.5°(derecha)

5.2. Eficiencia aerodinámica

Como se puede observar en la Figura 5 eficiencias aerodinámicas más altas conducen a unas aceleraciones menores. Sin embargo, hay que tener en cuenta otros factores como la geometría de la cápsula o el tiempo y distancia necesarios para el aterrizaje por lo que se habrá de realizar la selección de la eficiencia teniendo en consideración dichos factores.

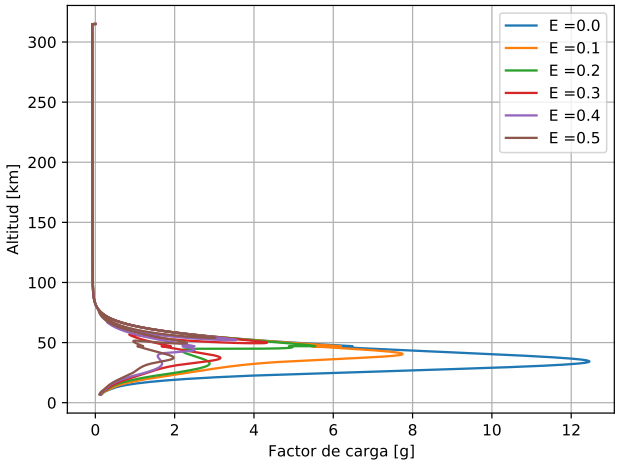

Figura 5: Evolución del factor de carga en función de la altitud para diferentes eficiencias aerodinámicas.

En primer lugar se han determinado la evolución en función del tiempo de las variables de estado, los resultados obtenidos se representan en la Figura 6. Como se puede observar mayores eficiencias conducen también a mayores tiempos de reentrada lo cual podría ser perjudicial para el piloto. Es por ello que se escoge la menor eficiencia aerodinámica que asegure la supervivencia el tripulante. En este caso se selecciona $E = 0.1$ ya que aunque se podría tratar de buscar un valor más ajustado, al tratarse de una misión tripulada se prefiere disponer de cierto margen de seguridad.

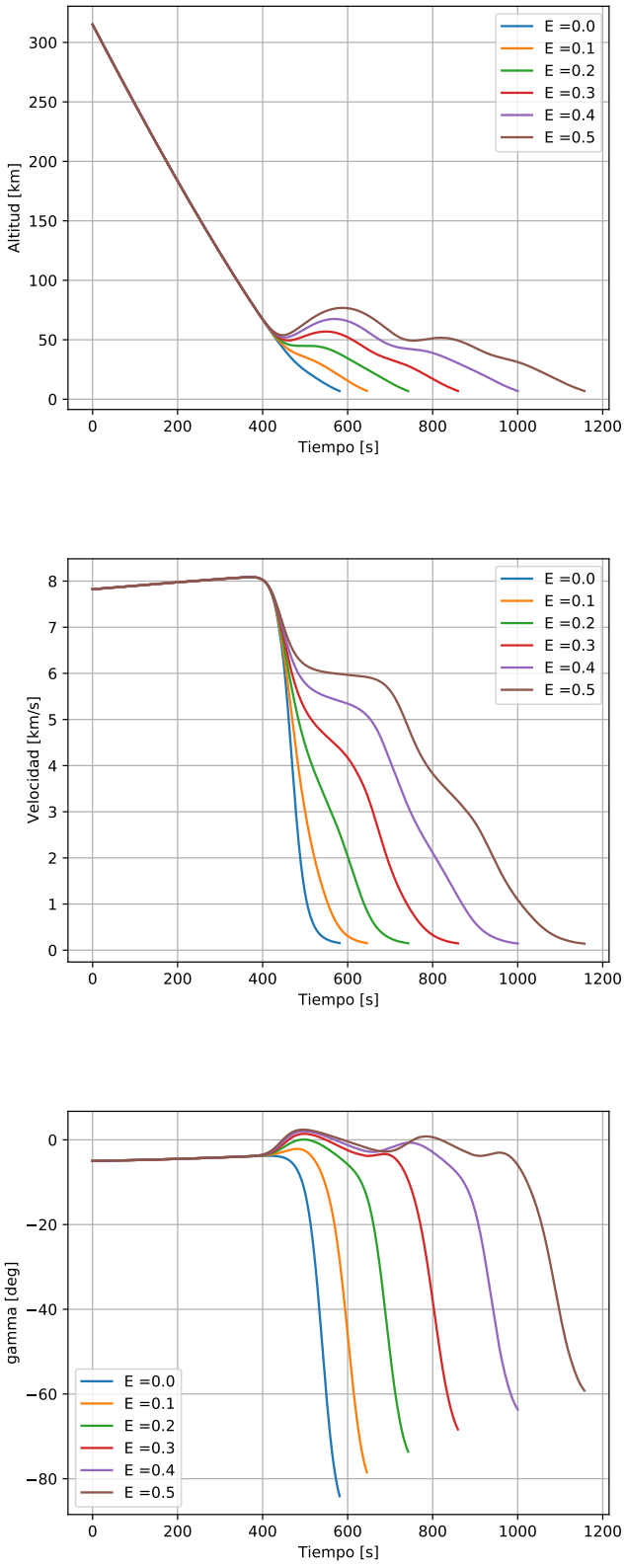

Figura 6: Evolución de la altitud, velocidad y ángulo de trayectoria con el tiempo para distintas eficiencias aerodinámicas.

5.3. Trayectoria

Una vez escogidos el ángulo y la eficiencia se ha de determinar con la mayor exactitud posible el lugar de aterrizaje puesto que puede comprometer a la población y el piloto ha de ser rescatado (ambas consideraciones no se tuvieron en cuenta o los cálculos fallaron durante el desarrollo de la misión ya que Gagarin aterrizó en mitad de una granja colectiva y la cápsula a unos 26 km del cosmonauta [**?**]). En la Figura 7 se analiza la ruta que la cápsula habría de seguir según las ecuaciones de la dinámica de reentrada (nótese que dichas ecuaciones son válidas para flujo hipersónico y que durante las últimas etapas de vuelo la cápsula ha decelerado suficiente como para abandonar este régimen).

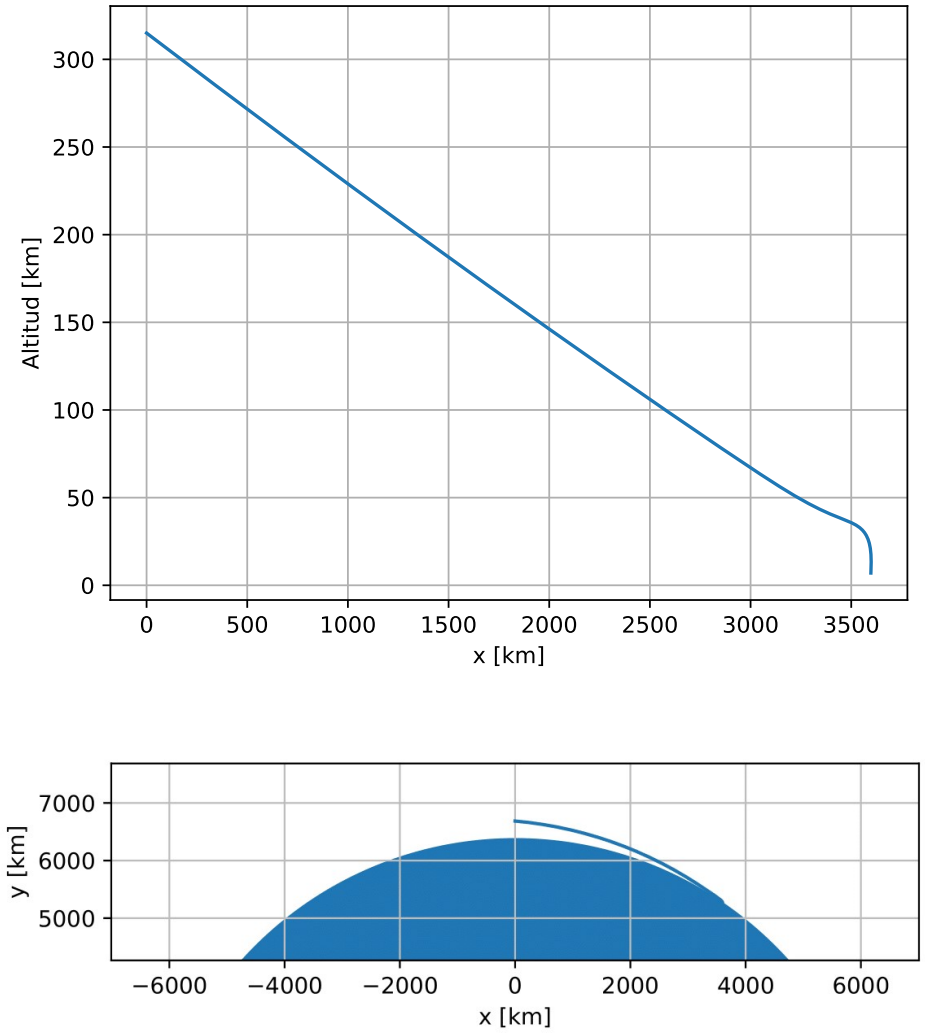

Figura 7: Trayectoria de la cápsula respecto al horizonte (arriba) y respecto a la Tierra (abajo).

5.4. Correlación con modelos analíticos

Por último los resultados obtenidos se comparan con las soluciones analíticas de reentrada balística y en planeo. Si el estudio ha sido correcto es de esperar que la solución se encuentre cercana a ambas soluciones. En la Figura 8 se representan gráficamente las soluciones analíticas y la numérica. Como se puede ver las tres soluciones son bastante similares pero en el caso del factor de carga se hace patente el comportamiento de fugoide de la cápsula, es decir, que a la hora de atravesar la atmósfera la cápsula puede acumular cierta energía potencial que la impulse de nuevo hacia arriba antes de volver a bajar. Este comportamiento es similar al del rebote pero con una amplitud mucho menor. En la evolución de la Figura 6 se puede ver que cápsulas con mayores eficiencias aerodinámicas presentan este comportamiento oscilatorio de un modo mucho más acusado.

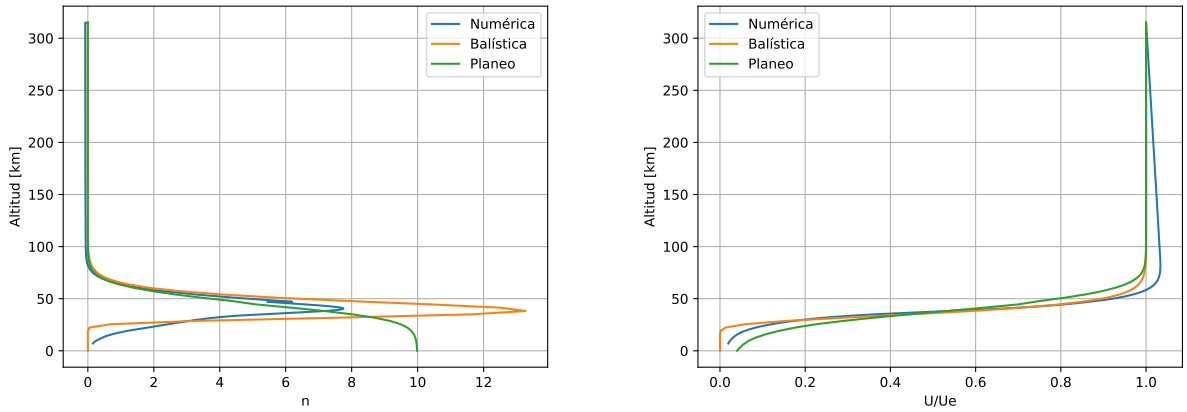

Figura 8: Comparación de la solución numérica con las soluciones analíticas de planeo y entrada balística según factor de carga (izquierda) y evolución de velocidades (derecha).

6. Conclusiones

Del trabajo realizado se extraen las siguientes conclusiones:

- Es necesario un estudio detallado de las condiciones de la reentrada para asegurar el correcto desarrollo de la misión. Como se puede ver en las Figuras 3 y 4 la ventana de valores posibles de ángulo de reentrada es harto estrecha y puede conducir a situaciones que comprometan la misión. Finalmente se ha elegido para este estudio un ángulo de reentrada de 5°.
- La eficiencia aerodinámica juega un papel clave a la hora de determinar la trayectoria de la reentrada y las cargas máximas, en este caso se han ensayado valores entre 0 y 0.5 pues en el desarrollo de la misión Vostok-1 ([2]) se contemplaron valores en este rango ya que la cápsula tiene forma esférica.
- Del estudio de la eficiencia aerodinámica se ha decidido escoger un valor de 0.1 ya que presenta aceleraciones dentro de lo admisible sin suponer un tiempo de vuelo demasiado grande. Esta última conclusión se extrae de la Figura 6
- La simulación numérica ha sido contrastada con las soluciones analíticas de entrada en planeo y entrada balística con un resultado satisfactorio como es patente en la Figura 8

En posteriores estudios habría de tenerse en cuenta también la evolución de la eficiencia y el coeficiente balístico a medida que se realiza el descenso, pues el ángulo de ataque cambia si no se ve corregido y la nave se va liberando de peso a medida que baja. En las últimas etapas habrían de aplicarse otras ecuaciones acordes con el régimen aerodinámico en curso y para el aterrizaje se tendrían que incluir el efecto de los paracaídas y las trayectorias de la nave y del cosmonauta el cual fue eyectado a 7 km de altura. Este tipo de modificaciones se podrían realizar de forma sencilla sobre el código introduciendo las debidas dependencias en el método “F“ de la clase “mision“ ya que el esquema puede adaptar el salto temporal de manera que en la discontinuidad no se produzcan saltos demasiado bruscos en la solución pero haría de prestarse especial cuidado a la estabilidad de la ecuación para no llegar a soluciones espurias.

A. Códigos empleados

A continuación se muestran los códigos empleados para las simulaciones descritas anteriormente. Estos códigos se encuentran disponibles también en [1].

A.1. Vostok

Se ha desarrollado un código a modo de ejemplo que usa características de la cápsula de la misión Vostok-1.

```
from mision_class import mision, save_result, read_result
from numpy import rad2deg, deg2rad, cos, sin
import numpy as np

# Representaciones gráficas
import matplotlib.pyplot as plt
import os

# Directorio de imágenes
figures_dir = './Figures/Vostok/'
if not os.path.exists(figures_dir):
    os.makedirs(figures_dir)

# Reentrada de la cápsula Vostok 1
Vostok = mision()

# Coeficiente balístico y eficiencia según el ángulo de vuelo
Vostok.beta = lambda gamma: 628.0851

# Densidad ISA
from isa import isa_rho
Vostok.rho_fun = isa_rho

# Reentrada

Ue      = 7.8232e3
gamma_e = deg2rad(-5)
z0      = 315e3
theta0  = 0
```

```
30
31 Vostok.verbose = 100  # Cada cuantas iteraciones hace display
32 Vostok.epsilon = 0.01 # Error admisible
33
34 # Estudio de la eficiencia aerodinámica necesaria
35
36 Eficiencias = np.linspace(0,0.5,6)
37 #Eficiencias = [0,0.2,0.4,0.6,0.8,0.9,1]
38
39 nmax_list = []
40 for Eficiencia in Eficiencias:
41     Vostok.E = lambda gamma: Eficiencia
42     Vostok.reentrada([Ue,gamma_e,z0,theta0],rfin=Vostok.RT+7e3)
43     save_result('Vostok'+str(round(Eficiencia*100))+'t',Vostok.t)
44     save_result('Vostok'+str(round(Eficiencia*100))+'y',Vostok.y)
45     save_result('Vostok'+str(round(Eficiencia*100))+'dy',Vostok.dy)
46     nmax_list.append(Vostok.nmax)
47
48 fig = plt.figure()
49 legends = []
50 for Eficiencia in Eficiencias:
51     y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
52     dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
53     plt.plot(-dy[:,0]/Vostok.g0,(y[:,2]-Vostok.RT)/1e3)
54     legends.append('E ='+str(round(Eficiencia,3)))
55 plt.xlabel('Factor de carga [g]')
56 plt.ylabel('Altitud [km]')
57 plt.grid()
58 plt.legend(legends)
59 plt.savefig(figures_dir+'Estudio_eficiencia.pdf')
60 plt.close()
61
62 ## Estudio de las soluciones a posteriori
63
64 altura_tiempo       = 'yes'
65 velocidad_tiempo    = 'yes'
66 gamma_tiempo        = 'yes'
67 trayectorias        = 'yes'
```

```
68
69  if altura_tiempo == 'yes':
70      fig = plt.figure()
71      for Eficiencia in Eficiencias:
72          t  = read_result('Vostok'+str(round(Eficiencia*100))+'t')
73          y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
74          dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
75          plt.plot(t,(y[:,2]-Vostok.RT)/1e3)
76          legends.append('E ='+str(round(Eficiencia,3)))
77      plt.xlabel('Tiempo [s]')
78      plt.ylabel('Altitud [km]')
79      plt.grid()
80      plt.legend(legends)
81      plt.savefig(figures_dir+'Altitud.pdf')
82      plt.close()
83
84  if velocidad_tiempo == 'yes':
85      fig = plt.figure()
86      for Eficiencia in Eficiencias:
87          t  = read_result('Vostok'+str(round(Eficiencia*100))+'t')
88          y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
89          dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
90          plt.plot(t,(y[:,0])/1e3)
91          legends.append('E ='+str(round(Eficiencia,3)))
92      plt.xlabel('Tiempo [s]')
93      plt.ylabel('Velocidad [km/s]')
94      plt.grid()
95      plt.legend(legends)
96      plt.savefig(figures_dir+'Velocidad.pdf')
97      plt.close()
98
99  if gamma_tiempo == 'yes':
100     fig = plt.figure()
101     for Eficiencia in Eficiencias:
102         t  = read_result('Vostok'+str(round(Eficiencia*100))+'t')
103         y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
104         dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
105         plt.plot(t,rad2deg(y[:,1]))
```

```
106         legends.append('E ='+str(round(Eficiencia,3)))
107     plt.xlabel('Tiempo [s]')
108     plt.ylabel('gamma [deg]')
109     plt.grid()
110     plt.legend(legends)
111     plt.savefig(figures_dir+'gamma.pdf')
112     plt.close()
113 
114 if trayectorias == 'yes':
115     fig = plt.figure()
116     for Eficiencia in Eficiencias:
117         t  = read_result('Vostok'+str(round(Eficiencia*100))+'t')
118         y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
119         dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
120         x = y[:,2]*sin(y[:,3])/1e3
121         plt.plot(x,(y[:,2]-Vostok.RT)/1e3)
122         legends.append('E ='+str(round(Eficiencia,3)))
123     plt.xlabel('x [km]')
124     plt.ylabel('Altitud [km]')
125     plt.grid()
126     plt.legend(legends)
127     plt.savefig(figures_dir+'trayectorias.pdf')
128     plt.close()
129 
130 
131 ## Trayectoria de la reentrada seleccionada
132 
133 trayectoria         = 'yes'
134 
135 index = nmax_list.index(max([value for index,value in enumerate(nmax_list) if value <
        8]))
136 print('')
137 print('***')
138 print('La eficiencia seleccionada es:',Eficiencias[index])
139 print('Esto supone una aceleración de:',nmax_list[index],'g')
140 print('***')
141 print('')
142 
```

```
143 Vostok.E = lambda gamma: Eficiencias[index]
144 Vostok.reentrada([Ue,gamma_e,z0,theta0],rfin=Vostok.RT+7e3)
145
146 import matplotlib.image as image
147 import matplotlib.patches as patches
148
149 if trayectoria == 'yes':
150     x = Vostok.r*sin(Vostok.theta)/1e3
151     y = Vostok.r*cos(Vostok.theta)/1e3
152
153     fig = plt.figure()
154     plt.plot(x,Vostok.altitud/1e3)
155     plt.grid()
156     plt.xlabel('x [km]')
157     plt.ylabel('Altitud [km]')
158     plt.savefig(figures_dir+'Trayectoria.pdf')
159     plt.close()
160
161     fig, ax = plt.subplots()
162
163     Tierra = plt.Circle((0, 0), Vostok.RT/1e3, color='tab:blue')
164     ax.add_patch(Tierra)
165     plt.plot(x,y)
166     plt.grid()
167     #ax.set_xlim(min(x)-1e3,max(x)+1e3)
168     ax.set_ylim(min(y)-1e3,max(y)+1e3)
169     ax.set_aspect('equal', 'box')
170     plt.xlabel('x [km]')
171     plt.ylabel('y [km]')
172     plt.savefig(figures_dir+'Trayectoria_planeta.pdf')
173     plt.close()
174
175 ## Correlación de la reentrada seleccionada
176
177 Vostok.reentrada_analitica([Ue,gamma_e,z0,theta0])
```

A.2. mision (clase)

Este código contiene la clase “mision“

- Atributos:

  Las funciones “beta(gamma)“ y “E(gamma)“

  Las constantes relacionadas con el planeta: “RT“, “g0“, “mu“ (por defecto los de la Tierra)

  Las funciones “rho_fun(z)“ y “g_fun(z)“ que pueden ser lambda function, funciones definidas en el código o cualquier función externa, siempre y cuando devuelvan un escalar (por defecto son un modelo exponencial de densidad y la gravedad newtoniana)

  El error admisible “epsilon“ (por defecto 0.1)

- Métodos:

  reentrada([u0,gamma0,r0,theta0],rfin = 6.371e6,tfin = ”)

  Devuelve: u, gamma, r, theta

  Además guarda como atributos: u, gamma, r, altitud, theta, t y n

  reentrada_analitica([u0,gamma0,r0,theta0])

  Crea y guarda las imágenes de compartiva: U/Ue vs z y n vs z

```
from math import exp, log, sin, cos, tan, pi
import numpy as np
from numpy import deg2rad, rad2deg
from numpy import genfromtxt
import os

# Directorio de imágenes
figures_dir = './Figures/'
if not os.path.exists(figures_dir):
    os.makedirs(figures_dir)

# Para guardar los resultados en un csv
def save_result(name,result):
    results_dir = './Results/' # Ruta
    if not os.path.exists(results_dir):
        os.makedirs(results_dir)
    np.savetxt(results_dir+str(name)+'.csv', result, delimiter=',', fmt='%s')
```

```
18
19 # Para leer los resultados de un csv
20 def read_result(name):
21     results_dir = './Results/' # Ruta
22     if not os.path.exists(results_dir):
23         print('No existe la carpeta de resultados ./Results')
24     my_data = genfromtxt(results_dir+str(name)+'.csv', delimiter=',')
25     return my_data
26
27 class mision():
28
29     # Constructor
30     def __init__(self,beta= lambda gamma: 64, E = lambda gamma: 0.01):
31
32         # Características del vehículo
33         self.beta = beta    # Coeficiente balístico, por defecto es una función
               constante
34         self.E    = E       # Eficiencia aerodinámica, por defecto es una función
               constante
35
36         # Constantes
37
38         self.RT = 6.371e6 #m        # Radio del planeta
39         self.g0 = 9.81 #m/s          # Gravedad a nivel de superficie
40         self.mu = 3.986 #m^3 s^(-2)   # Constante gravitacional
41
42         # Gravedad y atmósfera
43         # Si no se modifican se usan las predeterminadas
44
45         self.rho_fun = self.RHO
46         self.g_fun   = self.G
47
48         #
49         self.epsilon  = 0.1
50         self.verbose  = 500  #Pasos con output
51         self.k_lim    = 1e5  # Máximos pasos
52         self.alt_flag = 1
53         self.plt_flag = 1
```

```
54
55      # Ecuaciones de reentrada según esquema numérico
56      def reentrada(self,Y0,rfin = 6.371e6,tfin = ''):
57
58          print('*** Simulación en curso ***')
59
60          # Si la distancia inicial es menor que el radio de la Tierra se asume que es
                un dato de altitud
61          if Y0[2] <= self.RT and self.alt_flag == 1: Y0[2] = Y0[2] + self.RT
62          print('')
63          print('Se ha asumido que la condición inicial de distancia al centro de la
                Tierra es la altitud inicial')
64          print(' Para desactivar esta opción modifique el booleano alt_flag')
65          print(' >> reentrada.alt_flag = 0')
66
67          # Criterio de parada
68          print('')
69          if rfin == 6.371e6 and tfin == '':
70              print('La simulación será llevada a cabo hasta llegar al nivel del mar
                    terrestre')
71              print(' Para establecer otra altidud de parada a ada un argumento rfin')
72              print(' >> reentrada(self,Dt,Y0,rfin = 11e6)')
73              print(' Para establecer un tiempo de parada a ada un argumento tfin')
74              print(' >> reentrada(self,Dt,Y0,tfin = 42000)')
75
76          print('')
77          self.embedded_RK4(Y0,rfin,tfin)
78          print('*** Simulación finalizada ***')
79
80          self.u          = self.y[:,0]
81          self.gamma      = self.y[:,1]
82          self.r          = self.y[:,2]
83          self.altitud    = self.y[:,2] - self.RT
84          self.theta      = self.y[:,3]
85          self.n          = -self.dy[:,0] / self.g0
86          self.nmax       = self.n.max()
87          print('Máxima carga alcanzada n=',self.nmax,'g')
88
```

```
89          return self.u, self.gamma, self.r, self.theta
90
91      # Soluciones analíticas para comparar, crea las figuras directamente
92      def reentrada_analitica(self,Y0=[7e3,-10,100e3,0]):
93
94          if Y0[2] <= self.RT and self.alt_flag == 1: Y0[2] = Y0[2] + self.RT
95          z0 = ze = Y0[2]-self.RT
96
97          rho0 = self.RHO(0)
98          zs = -1/log(self.RHO(1)/rho0)
99
100         g0 = self.g0
101         mu = self.mu
102         RT = self.RT
103         beta_val = self.beta(0)
104         E_val    = self.E(0)
105
106         # Representación gráfica
107         print('')
108         print('Se representará graficamente la solución numérica frente a las analí
                tica')
109         print(' Para desactivar esta opción modifique el booleano plt_flag')
110         print(' >> reentrada.plt_flag = 0')
111         if self.plt_flag == 1:
112             import matplotlib.pyplot as plt
113             if hasattr(self, 'u'):
114                 u       = self.u
115                 ue      = self.u[0]
116                 gamma_e = self.gamma[0]
117                 z       = np.linspace(0,z0,100)
118                 n       = self.n
119
120                 print('Ue =',ue/1e3,'km/s')
121                 print('gamma_e =',rad2deg(gamma_e),'deg')
122                 print('z0 =',z0/1e3,'km')
123                 print('rho0 =',rho0,'kg/m^3')
124                 print('zs =',zs/1e3,'km')
125             else:
```

```
126             print('** Error: Corra primero la simulación numérica')
127             exit()
128 
129         b = zs*rho0/(2*beta_val*sin(gamma_e))
130         c = ue**2*rho0/(2*beta_val*g0)
131 
132         self.u_numerica     = u/ue
133         self.u_balistica    = np.exp( b * np.exp(-z/zs) )
134         self.u_planeo       = np.empty((0,1))
135         for zz in z:
136             self.u_planeo= np.append(self.u_planeo,(1+E_val/beta_val * self.
                rho_fun(zz)*RT/2)**-0.5)
137 
138         self.n_numerica     = n
139         self.n_balistica    = c * np.exp(2*b*np.exp(-z/zs)-z/zs)
140         self.n_planeo       = 1/E_val * (1-(self.u_planeo)**2)
141 
142         # U/Ue vs Z
143         fig = plt.figure()
144         plt.plot(self.u_numerica,self.altitud/1e3)
145         plt.plot(self.u_balistica,z/1e3)
146         plt.plot(self.u_planeo,z/1e3)
147         plt.xlabel('U/Ue')
148         plt.ylabel('Altitud [km]')
149         plt.legend(['Numérica','Balística','Planeo'])
150         plt.grid()
151         plt.savefig(figures_dir+'u_analitica.pdf')
152         plt.close()
153         # n vs z
154         fig = plt.figure()
155         plt.plot(self.n_numerica,self.altitud/1e3)
156         plt.plot(self.n_balistica,z/1e3)
157         plt.plot(self.n_planeo,z/1e3)
158         plt.xlabel('n')
159         plt.ylabel('Altitud [km]')
160         plt.legend(['Numérica','Balística','Planeo'])
161         plt.grid()
162         plt.savefig(figures_dir+'n_analitica.pdf')
```

```
163             plt.close()
164
165     ## Esquemas numéricos
166
167     def embedded_RK4(self,Y0,rfin,tfin):
168
169         # Paso inicial
170         Dt = 0.1
171
172         # Condiciones iniciales
173         Y = Y0
174         t = 0
175
176         # Ecuaciones y el solver RK4
177         F = self.F
178         RK4 = self.RK4
179
180         # Vectores de estado, resultados
181         self.y  = np.array(Y)
182         self.dy = np.array([0, 0, 0, 0])
183         self.t  = np.array(t)
184
185         k = 1
186         last_print = 0
187
188         print('Iteracion:',0,'// Dt =',:.3e.format(Dt),'// Altitud:', round((Y[2]—self
                .RT)/1e3,3), 'km // Tiempo:',0,'s')
189         # Criterio de parada espacial
190         while Y[2]>=rfin:
191
192             # Criterio de parada temporal
193             if isinstance(tfin,float):
194                 if t >= tfin: break
195
196             # Cálculo del siguiente estado
197             Y1 = RK4(Dt,Y)
198             Y21 = RK4(Dt/2,Y)
199             Y2 = RK4(Dt/2,Y21)
```

```
200
201             # Cálculo del error
202             err = Y1-Y2
203             err[0] = err[0]
204             err[1] = tan(err[1]/2)*1e3 # Los errores angulares se penalizan más
205             err[2] = err[2]
206             err[3] = tan(err[3]/2)*1e3
207             mag = np.sqrt(err.dot(err))
208             if mag < 1e-12:
209                 Dt_new = 1e12
210             else:
211                 Dt_new = Dt*(self.epsilon/mag)**(1/(4+1))
212
213             Dt_old = Dt
214
215             if Dt <= Dt_new: # El paso de tiempo ha sido correcto y se usa la iteración
216                 Y   = Y1
217                 dY  = F(Y)
218                 t   = t+Dt
219                 k   = k+1
220                 self.y  = np.vstack([self.y,Y])
221                 self.dy = np.vstack([self.dy,dY])
222                 self.t  = np.append(self.t,t)
223
224                 Dt = Dt+0.01  # Para la siguiente vez se aumenta el paso
225             elif Dt > Dt_new:             # El paso de tiempo era insuficiente y se repite la iteración
226                 Dt = Dt_new
227             else:                         # Algún problema ha sucedido a la hora de calcular Dt_new
228                 Dt = 0.1
229
230             # Criterio de parada Dt extremadamente peque o
231             if Dt < 1e-28:
232                 print('Imposible dar el siguiente paso')
233                 break
234
```

```
235             # Criterio de parada rebote
236             if Y[2] > Y0[2] + 100e3:
237                 print('La cápsula ha rebotado')
238                 break
239 
240             # Criterio de parada por número de pasos
241             if k>= self.k_lim:
242                  print('Número máximo de pasos alcanzado')
243                  break
244 
245             # Report de las iteraciones
246             if k%self.verbose == 0 and k != last_print:
247                 print('Iteracion:',k,'// Dt =',:.3e.format(Dt_old),'// Altitud:',
                        round((Y[2]-self.RT)/1e3,3), 'km // Tiempo:',round(t,3),'s')
248                 last_print = k
249 
250     def RK4(self,Dt,Y):
251 
252         F = self.F
253 
254         # Runge Kutta 4
255         K1 = F(Y)
256         K2 = F(Y+Dt/2 * K1)
257         K3 = F(Y+Dt/2 * K2)
258         K4 = F(Y+Dt   * K3)
259 
260         return Y + Dt/6 * (K1+2*K2+2*K3+K4)
261 
262 
263     # Ecuaciones de la dinámica de reentrada
264     def F(self,Y):
265 
266         u       = Y[0]
267         gamma   = Y[1]
268         r       = Y[2]
269         theta   = Y[3]
270 
271         z = r-self.RT
```

```
272
273         g           = self.g_fun(z)
274         rho         = self.rho_fun(z)
275         beta_val    = self.beta(gamma)
276         E_val       = self.E(gamma)
277
278         eqns  = [-g*sin(gamma)-rho/(2*beta_val) * u**2,
279         1/u * (u**2 * (rho*E_val/(2*beta_val) + cos(gamma)/r) - g*cos(gamma)),
280         u*sin(gamma),
281         1/r * u*cos(gamma)]
282
283         return np.array(eqns)
284
285     ## Funciones de gravedad y densidad
286
287     def G(self,z):
288
289         g0 = self.g0
290         RT = self.RT
291         g = g0 * (RT/(z+RT))**2
292
293         return g
294
295     def RHO(self,z):
296
297         RT = self.RT
298         if z <= 150e3: #m
299             rho0 = 1.225 #kg/m^3
300             z0 = 7.524e3 #m
301         elif z> 150e3: #m
302             rho0 = 3.875e-9 #kg/m^3
303             z0 = 59.06e3 #m
304
305         try:
306             rho = rho0 * exp(-z/z0)
307         except:
308             rho = 1000 #kg/m^3
309         return rho
```

A.3. ISA

Uno de los modelos de atmósfera más comunes, dentro de los modelos estáticos es el que se conoce como ISA (Internetional Standart Atmosphere) y este ha sido el modelo usado para determinar la densidad en función de la altura.

```
from math import exp

def isa_altitude(h, F10 = 330, Ap = 0):

# h en metros
# P en Pascales
# rho en kg/m^3
# T en Kelvin

        g = 9.81 #m/s^2
        M = 28.9e-3 #kg/mol
        R = 287
        T0 = 288.15 #K
        P0 = 101325 #Pa
        rho0 = 1.225 #kg/m^3

        if h < 11000:
                landa = -6.5e-3
                T = T0 + landa*h
                P = P0* (T/T0)**(-g/(R*landa))
                rho = rho0 * (T/T0)**(-g/(R*landa)-1)

        elif h < 25000:
                T11 = 216.65
                P11 = 22552
                rho11 = 0.3629
                #landa=0

                T=T11
                P = P11*exp(-g*(h-11000)/(R*T))
                rho=rho11*exp(-g*(h-11000)/(R*T))

        elif h<47000:
```

```
34
35              landa=3e-3
36              T25=216.65
37              P25=2481
38              rho25=0.0399
39
40              T=T25+landa*(h-25000)
41              P=P25*(T/T25)**(-g/(R*landa))
42              rho=rho25*(T/T25)**(-g/(R*landa)-1)
43
44          elif h<180e3 :
45
46              #CIRA model
47              a0 =  7.001985e-2
48              a1 = -4.336216e-3
49              a2 = -5.009831e-3
50              a3 =  1.621827e-4
51              a4 = -2.471283e-6
52              a5 =  1.904383e-8
53              a6 = -7.189421e-11
54              a7 =  1.060067e-13
55
56              h = h*1e-3
57
58              polyh = ((((((a7*h + a6)*h + a5)*h + a4)*h + a3)*h + a2)*h + a1)*h +
                    a0
59
60              rho = 10**(polyh)
61
62              #Esto no esta bien
63              T = 900 + 2.5 * (F10 - 70) + 1.5*Ap
64              P= rho*R*T
65
66          elif h<=550e3:
67
68              R = 287
69
70              T = 900 + 2.5 * (F10 - 70) + 1.5*Ap
```

```
71                mu = 27 - 0.012 * (h - 200)
72                H = T / mu
73                try:
74                        rho = 6*10e-10*exp( - (h - 175) / H )
75                except:
76                        rho = 0
77
78                #Esto no esta bien
79                P= rho*R*T
80
81        else:
82                #print('Atmósfera no apreciable')
83
84                rho=0
85                P=0
86                T=0
87
88        return P, rho, T
89
90 def isa_P(z):
91        P, rho, T = isa_altitude(z)
92        return P
93
94 def isa_rho(z):
95        P, rho, T = isa_altitude(z)
96        return rho
97
98 def isa_T(z):
99        P, rho, T = isa_altitude(z)
100       return T
```

Bibliografía

[1] temisAP/Trabajos$_A AV_1$23.
URL https://github.com/temisAP/Trabajos_AAV_123

[2] Vostok.
URL http://www.astronautix.com/v/vostok.html

[3] Vostok 1 - Wikipedia.
URL https://en.wikipedia.org/wiki/Vostok_1

[4] 6 Surprising Facts About the First Manned Space Mission | Live Science.
URL https://www.livescience.com/33185-yuri-gagarin-vostok-1-faq-facts.html