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.

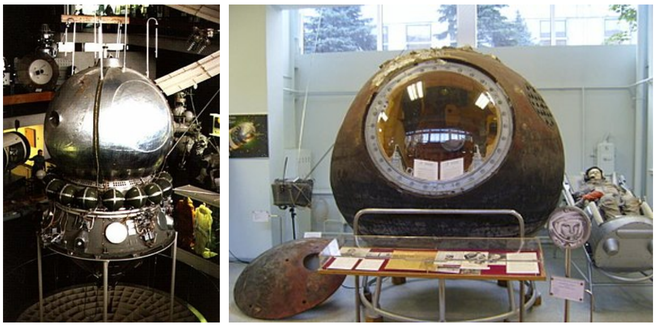


Figura 1: Cápsula de la misión Vostok-1 antes del lanzamiento (izquierda) y después del aterrizaje (derecha).

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):

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -g(z)\cdot\sin\gamma - D(z,u), \tag{1}$$
$$\frac{\mathrm{d}\gamma}{\mathrm{d}t} = \frac{1}{u}\left(L(z,u) - g(z)\cdot\cos\gamma + \frac{u^2}{r}\cdot\cos\gamma\right), \tag{2}$$
$$\frac{\mathrm{d}r}{\mathrm{d}t} = u\cdot\sin\gamma, \tag{3}$$
$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = \frac{1}{r}u\cdot\cos\gamma, \tag{4}$$

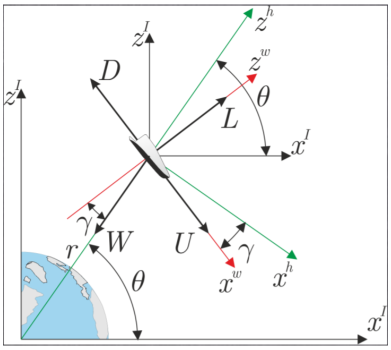


Figura 2: Representación de las variables y sistemas de referencia que intervienen en las ecuaciones.

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:

$$L(z,u) = \frac{\rho(z)u^2E}{2\beta}, \tag{5}$$
$$D(z,u) = \frac{\rho(z)u^2}{2\beta}, \tag{6}$$

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

$$\beta = \frac{m}{Sc_D}\ , \tag{7}$$

$$E = \frac{c_L}{c_D}\ . \tag{8}$$

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

$$n = -\frac{\mathrm{d}U}{\mathrm{d}t}\frac{1}{g_0} \tag{9}$$

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

$$B = \frac{z_s \rho_0}{2\beta \sin \gamma_e} \, , \tag{10}$$

$$C = \frac{u_e^2 \rho_0}{2\beta g_0} \, , \tag{11}$$

$$\frac{u}{u_e} = \exp\left(B \cdot \exp\left(-\frac{z}{z_e}\right)\right) \, , \tag{12}$$

$$n = C \cdot \exp\left(2B \cdot \exp\left(-\frac{z}{z_e}\right) - \frac{z}{z_s}\right) \, . \tag{13}$$

y de reentrada con sustentación:

$$\frac{u}{u_e} = \left(\sqrt{1 + \frac{E R_T \rho(z)}{2\beta}}\right)^{-1} \, , \tag{14}$$

$$n = \frac{1}{E}\left[1 - \left(\frac{u}{u_e}\right)^2\right] = \left(\sqrt{E + \frac{2\beta \cdot \exp\left(-\frac{z}{z_e}\right)}{\rho_0 R_T}}\right)^{-1} \, , \tag{15}$$

donde u_e, z_e y γ_e son las variables de estado a 100 km de altitud; g_0 es el valor de la aceleración gravitatoria a 0 km de altitud; y ρ_0 y z_s es el que corresponde al modelo exponencial de la forma:

$$\rho = \rho_0 \exp\left(-\frac{z}{z_s}\right) \tag{16}$$

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:

```python
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:

```python
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

```

```python
    def RHO(self,z):

        RT = self.RT
        if z <= 150e3: #m
            rho0 = 1.225 #kg/m^3
            z0 = 7.524e3 #m
        elif z> 150e3: #m
            rho0 = 3.875e-9 #kg/m^3
            z0 = 59.06e3 #m

        try:
            rho = rho0 * exp(-z/z0)
        except:
            rho = 1000 #kg/m^3
        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:

```python
# rho = self.rho_fun(z)
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 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.

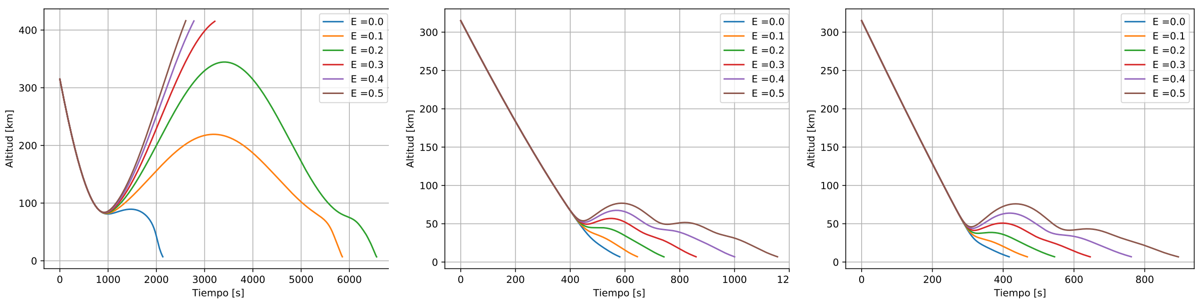


Figura 3: Evolución de la altitud con el tiempo para tres ángulos de entrada: 3.15°(izquierda), 5°(centro) y 7.5°(derecha)

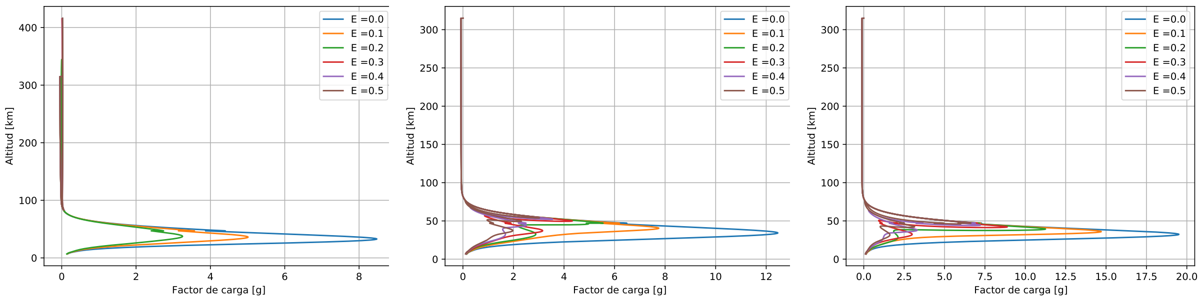


Figura 4: Evolución del factor de carga según la altitud para tres ángulos de entrada: 3.15°(izquierda), 5°(centro) y 7.5°(derecha)

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.

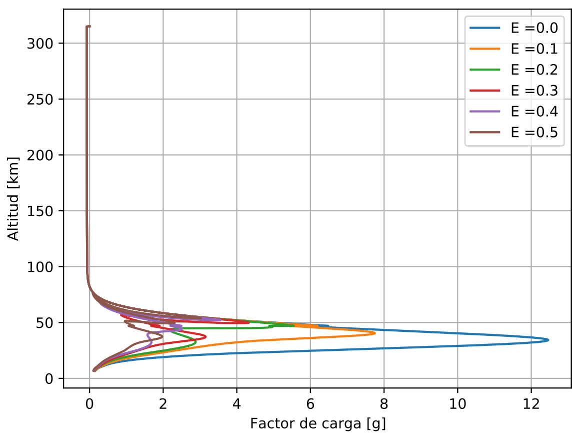
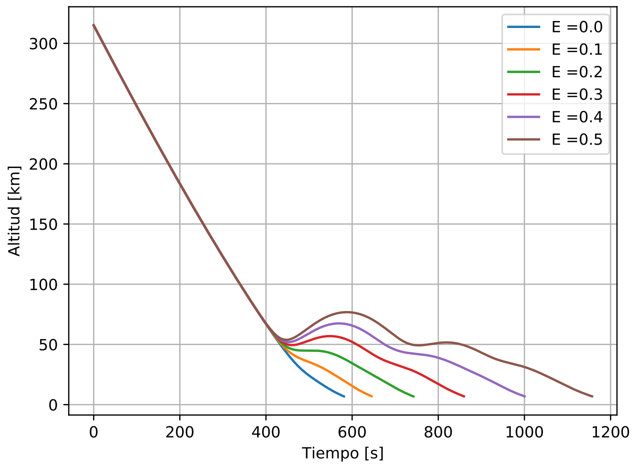
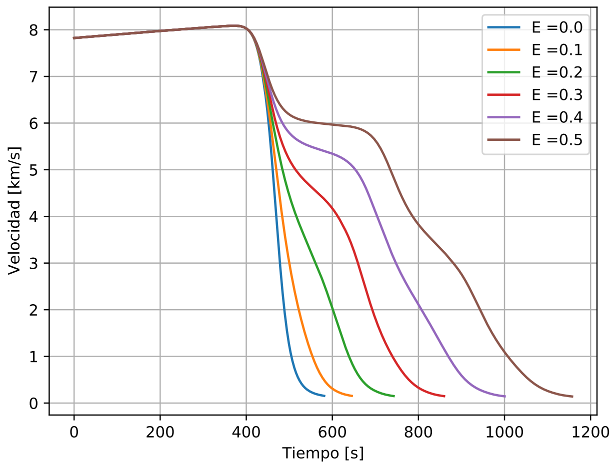


Figura 5: Evolución del factor de carga en función de la altitud para diferentes eficiencias aerodinámicas.

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.





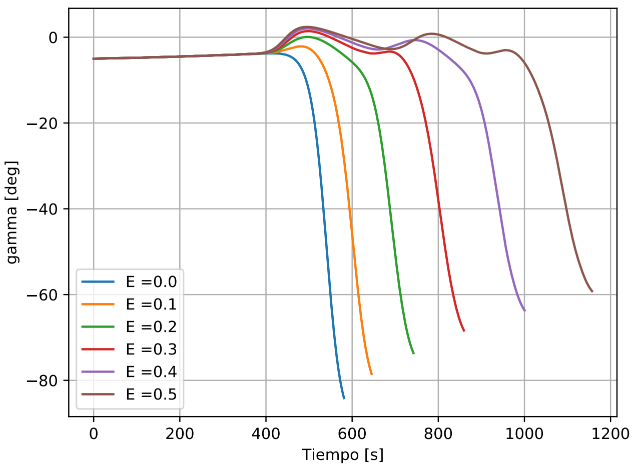


Figura 6: Evolución de la altitud, velocidad y ángulo de trayectoria con el tiempo para distintas eficiencias aerodinámicas.

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).

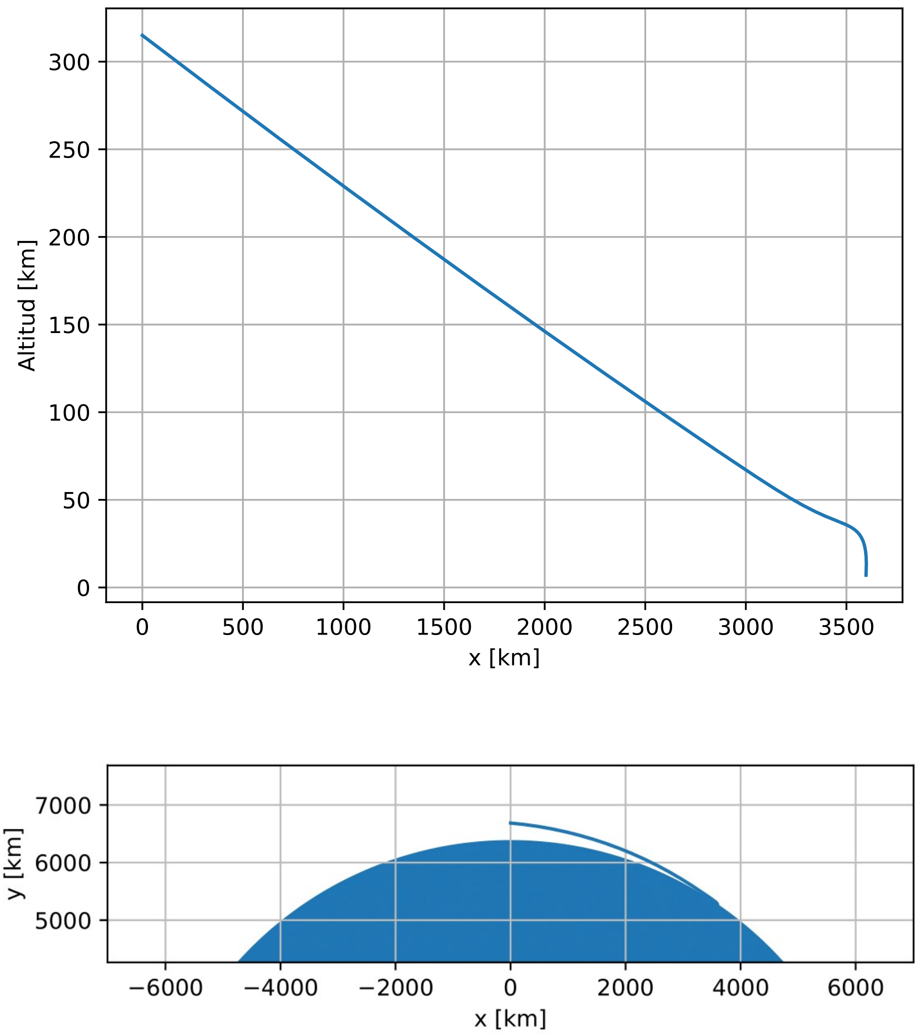


Figura 7: Trayectoria de la cápsula respecto al horizonte (arriba) y respecto a la Tierra (abajo).

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.

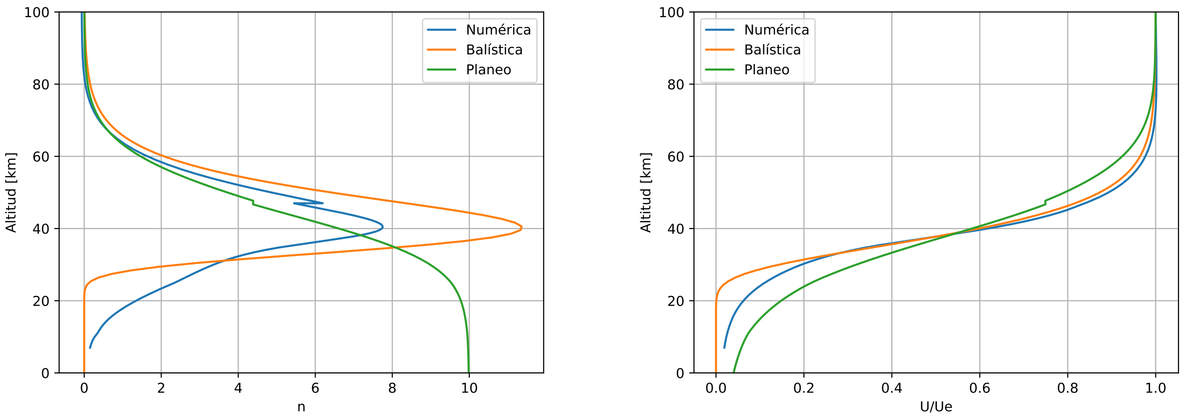


Figura 8: Comparación de la solución numérica con las soluciones analíticas de planeo y entrada balística según factor de carga (izquierda) y evolución de velocidades (derecha).

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.

```python
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
```

```python

Vostok.verbose = 100  # Cada cuantas iteraciones hace display
Vostok.epsilon = 0.01 # Error admisible

# Estudio de la eficiencia aerodinámica necesaria

Eficiencias = np.linspace(0,0.5,6)
#Eficiencias = [0,0.2,0.4,0.6,0.8,0.9,1]

nmax_list = []
for Eficiencia in Eficiencias:
    Vostok.E = lambda gamma: Eficiencia
    Vostok.reentrada([Ue,gamma_e,z0,theta0],rfin=Vostok.RT+7e3)
    save_result('Vostok'+str(round(Eficiencia*100))+'t',Vostok.t)
    save_result('Vostok'+str(round(Eficiencia*100))+'y',Vostok.y)
    save_result('Vostok'+str(round(Eficiencia*100))+'dy',Vostok.dy)
    nmax_list.append(Vostok.nmax)

fig = plt.figure()
legends = []
for Eficiencia in Eficiencias:
    y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
    dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
    plt.plot(-dy[:,0]/Vostok.g0,(y[:,2]-Vostok.RT)/1e3)
    legends.append('E ='+str(round(Eficiencia,3)))
plt.xlabel('Factor de carga [g]')
plt.ylabel('Altitud [km]')
plt.grid()
plt.legend(legends)
plt.savefig(figures_dir+'Estudio_eficiencia.pdf')
plt.close()

## Estudio de las soluciones a posteriori

altura_tiempo       = 'yes'
velocidad_tiempo    = 'yes'
gamma_tiempo        = 'yes'
trayectorias        = 'yes'
```

```python

if altura_tiempo == 'yes':
    fig = plt.figure()
    for Eficiencia in Eficiencias:
        t  = read_result('Vostok'+str(round(Eficiencia*100))+'t')
        y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
        dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
        plt.plot(t,(y[:,2]-Vostok.RT)/1e3)
        legends.append('E ='+str(round(Eficiencia,3)))
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Altitud [km]')
    plt.grid()
    plt.legend(legends)
    plt.savefig(figures_dir+'Altitud.pdf')
    plt.close()

if velocidad_tiempo == 'yes':
    fig = plt.figure()
    for Eficiencia in Eficiencias:
        t  = read_result('Vostok'+str(round(Eficiencia*100))+'t')
        y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
        dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
        plt.plot(t,(y[:,0])/1e3)
        legends.append('E ='+str(round(Eficiencia,3)))
    plt.xlabel('Tiempo [s]')
    plt.ylabel('Velocidad [km/s]')
    plt.grid()
    plt.legend(legends)
    plt.savefig(figures_dir+'Velocidad.pdf')
    plt.close()

if gamma_tiempo == 'yes':
    fig = plt.figure()
    for Eficiencia in Eficiencias:
        t  = read_result('Vostok'+str(round(Eficiencia*100))+'t')
        y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
        dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
        plt.plot(t,rad2deg(y[:,1]))
```

```python
        legends.append('E ='+str(round(Eficiencia,3)))
    plt.xlabel('Tiempo [s]')
    plt.ylabel('gamma [deg]')
    plt.grid()
    plt.legend(legends)
    plt.savefig(figures_dir+'gamma.pdf')
    plt.close()

if trayectorias == 'yes':
    fig = plt.figure()
    for Eficiencia in Eficiencias:
        t  = read_result('Vostok'+str(round(Eficiencia*100))+'t')
        y  = read_result('Vostok'+str(round(Eficiencia*100))+'y')
        dy = read_result('Vostok'+str(round(Eficiencia*100))+'dy')
        x = y[:,2]*sin(y[:,3])/1e3
        plt.plot(x,(y[:,2]-Vostok.RT)/1e3)
        legends.append('E ='+str(round(Eficiencia,3)))
    plt.xlabel('x [km]')
    plt.ylabel('Altitud [km]')
    plt.grid()
    plt.legend(legends)
    plt.savefig(figures_dir+'trayectorias.pdf')
    plt.close()


## Trayectoria de la reentrada seleccionada

trayectoria        = 'yes'

index = nmax_list.index(max([value for index,value in enumerate(nmax_list) if value <
    8]))
print('')
print('***')
print('La eficiencia seleccionada es:',Eficiencias[index])
print('Esto supone una aceleración de:',nmax_list[index],'g')
print('***')
print('')
```

```
Vostok.E = lambda gamma: Eficiencias[index]
Vostok.reentrada([Ue,gamma_e,z0,theta0],rfin=Vostok.RT+7e3)

import matplotlib.image as image
import matplotlib.patches as patches

if trayectoria == 'yes':
    x = Vostok.r*sin(Vostok.theta)/1e3
    y = Vostok.r*cos(Vostok.theta)/1e3

    fig = plt.figure()
    plt.plot(x,Vostok.altitud/1e3)
    plt.grid()
    plt.xlabel('x [km]')
    plt.ylabel('Altitud [km]')
    plt.savefig(figures_dir+'Trayectoria.pdf')
    plt.close()

    fig, ax = plt.subplots()

    Tierra = plt.Circle((0, 0), Vostok.RT/1e3, color='tab:blue')
    ax.add_patch(Tierra)
    plt.plot(x,y)
    plt.grid()
    #ax.set_xlim(min(x)-1e3,max(x)+1e3)
    ax.set_ylim(min(y)-1e3,max(y)+1e3)
    ax.set_aspect('equal', 'box')
    plt.xlabel('x [km]')
    plt.ylabel('y [km]')
    plt.savefig(figures_dir+'Trayectoria_planeta.pdf')
    plt.close()

## Correlación de la reentrada seleccionada

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

```python
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')
```

```

# Para leer los resultados de un csv
def read_result(name):
    results_dir = './Results/' # Ruta
    if not os.path.exists(results_dir):
        print('No existe la carpeta de resultados ./Results')
    my_data = genfromtxt(results_dir+str(name)+'.csv', delimiter=',')
    return my_data

class mision():

    # Constructor
    def __init__(self,beta= lambda gamma: 64, E = lambda gamma: 0.01):

        # Características del vehículo
        self.beta = beta    # Coeficiente balístico, por defecto es una función constante
        self.E    = E       # Eficiencia aerodinámica, por defecto es una función constante

        # Constantes

        self.RT = 6.371e6 #m        # Radio del planeta
        self.g0 = 9.81 #m/ s         # Gravedad a nivel de superficie
        self.mu = 3.986 #m  s^(-2)   # Constante gravitacional

        # Gravedad y atmósfera
        # Si no se modifican se usan las predeterminadas

        self.rho_fun = self.RHO
        self.g_fun   = self.G

        #
        self.epsilon  = 0.1
        self.verbose  = 500  #Pasos con output
        self.k_lim    = 1e5  # Máximos pasos
        self.alt_flag = 1
        self.plt_flag = 1
```

```python

    # Ecuaciones de reentrada según esquema numérico
    def reentrada(self,Y0,rfin = 6.371e6,tfin = ''):

        print('*** Simulación en curso ***')

        # Si la distancia inicial es menor que el radio de la Tierra se asume que es un dato de altitud
        if Y0[2] <= self.RT and self.alt_flag == 1: Y0[2] = Y0[2] + self.RT
        print('')
        print('Se ha asumido que la condición inicial de distancia al centro de la Tierra es la altitud inicial')
        print(' Para desactivar esta opción modifique el booleano alt_flag')
        print(' >> reentrada.alt_flag = 0')

        # Criterio de parada
        print('')
        if rfin == 6.371e6 and tfin == '':
            print('La simulación será llevada a cabo hasta llegar al nivel del mar terrestre')
            print(' Para establecer otra altidud de parada a ada un argumento rfin')
            print(' >> reentrada(self,Dt,Y0,rfin = 11e6)')
            print(' Para establecer un tiempo de parada a ada un argumento tfin')
            print(' >> reentrada(self,Dt,Y0,tfin = 42000)')

        print('')
        self.embedded_RK4(Y0,rfin,tfin)
        print('*** Simulación finalizada ***')

        self.u          = self.y[:,0]
        self.gamma      = self.y[:,1]
        self.r          = self.y[:,2]
        self.altitud    = self.y[:,2] - self.RT
        self.theta      = self.y[:,3]
        self.n          = -self.dy[:,0] / self.g0
        self.nmax       = self.n.max()
        print('Máxima carga alcanzada n=',self.nmax,'g')

```

```
        return self.u, self.gamma, self.r, self.theta

    # Soluciones analíticas para comparar, crea las figuras directamente
    def reentrada_analitica(self,Y0=[7e3,-10,100e3,0]):

        rho0 = self.RHO(0)
        zs = -1/log(self.RHO(1)/rho0)

        g0 = self.g0
        mu = self.mu
        RT = self.RT
        beta_val = self.beta(0)
        E_val    = self.E(0)

        # Representación gráfica
        print('')
        print('Se representará graficamente la solución numérica frente a las analítica')
        print(' Para desactivar esta opción modifique el booleano plt_flag')
        print(' >> reentrada.plt_flag = 0')
        if self.plt_flag == 1:
            import matplotlib.pyplot as plt
            if hasattr(self, 'u'):
                altitud = self.altitud.tolist()
                u       = self.u
                index   = altitud.index(min([value for index,value in enumerate(self.altitud) if value > 100e3]))
                z0      = self.altitud[index]
                ue      = self.u[index]
                gamma_e = self.gamma[index]
                z       = np.linspace(0,z0,100)
                n       = self.n

                print('Ue =',ue/1e3,'km/s')
                print('gamma_e =',rad2deg(gamma_e),'deg')
                print('z0 =',z0/1e3,'km')
                print('rho0 =',rho0,'kg/m^3')
                print('zs =',zs/1e3,'km')
```

```
        else:
            print('** Error: Corra primero la simulación numérica')
            exit()

        b = zs*rho0/(2*beta_val*sin(gamma_e))
        c = ue**2*rho0/(2*beta_val*g0)

        self.u_numerica     = u/ue
        self.u_balistica    = np.exp( b * np.exp(-z/zs) )
        self.u_planeo       = np.empty((0,1))
        for zz in z:
            self.u_planeo= np.append(self.u_planeo,(1+E_val/beta_val * self.
                rho_fun(zz)*RT/2)**-0.5)

        self.n_numerica     = n
        self.n_balistica    = c * np.exp(2*b*np.exp(-z/zs)-z/zs)
        self.n_planeo       = 1/E_val * (1-(self.u_planeo)**2)

        # U/Ue vs Z
        fig = plt.figure()
        plt.plot(self.u_numerica,self.altitud/1e3)
        plt.plot(self.u_balistica,z/1e3)
        plt.plot(self.u_planeo,z/1e3)
        plt.xlabel('U/Ue')
        plt.ylabel('Altitud [km]')
        plt.ylim([0, 100])
        plt.legend(['Numérica','Balística','Planeo'])
        plt.grid()
        plt.savefig(figures_dir+'u_analitica.pdf')
        plt.close()
        # n vs z
        fig = plt.figure()
        plt.plot(self.n_numerica,self.altitud/1e3)
        plt.plot(self.n_balistica,z/1e3)
        plt.plot(self.n_planeo,z/1e3)
        plt.xlabel('n')
        plt.ylabel('Altitud [km]')
        plt.ylim([0, 100])
```

```python
            plt.legend(['Numérica','Balística','Planeo'])
            plt.grid()
            plt.savefig(figures_dir+'n_analitica.pdf')
            plt.close()

    ## Esquemas numéricos

    def embedded_RK4(self,Y0,rfin,tfin):

        # Paso inicial
        Dt = 0.1

        # Condiciones iniciales
        Y = Y0
        t = 0

        # Ecuaciones y el solver RK4
        F = self.F
        RK4 = self.RK4

        # Vectores de estado, resultados
        self.y  = np.array(Y)
        self.dy = np.array([0, 0, 0, 0])
        self.t  = np.array(t)

        k = 1
        last_print = 0

        print('Iteracion:',0,'// Dt =',:.3e.format(Dt),'// Altitud:', round((Y[2]-self.RT)/1e3,3), 'km // Tiempo:',0,'s')
        # Criterio de parada espacial
        while Y[2]>=rfin:

            # Criterio de parada temporal
            if isinstance(tfin,float):
                if t >= tfin: break

            # 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))

        Dt_old = Dt

        if Dt <= Dt_new: # El paso de tiempo ha sido correcto y se usa la iteración
            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  # Para la siguiente vez se aumenta el paso
        elif Dt > Dt_new:             # El paso de tiempo era insuficiente y se repite la iteración
            Dt = Dt_new
        else:                         # Algún problema ha sucedido a la hora de calcular Dt_new
            Dt = 0.1

        # Criterio de parada Dt extremadamente peque o
        if Dt < 1e-28:
```

```
                print('Imposible dar el siguiente paso')
                break

            # Criterio de parada rebote
            if Y[2] > Y0[2] + 100e3:
                print('La cápsula ha rebotado')
                break

            # Criterio de parada por número de pasos
            if k>= self.k_lim:
                 print('Número máximo de pasos alcanzado')
                 break

            # Report de las iteraciones
            if k%self.verbose == 0 and k != last_print:
                print('Iteracion:',k,'// Dt =',:.3e.format(Dt_old),'// Altitud:',
                    round((Y[2]-self.RT)/1e3,3), 'km // Tiempo:',round(t,3),'s')
                last_print = k

    def RK4(self,Dt,Y):

        F = self.F

        # Runge Kutta 4
        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)


    # Ecuaciones de la dinámica de 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)

    ## Funciones de gravedad y densidad

    def G(self,z):

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

        return g

    def RHO(self,z):

        RT = self.RT
        if z <= 150e3: #m
            rho0 = 1.225 #kg/m^3
            z0 = 7.524e3 #m
        elif z> 150e3: #m
            rho0 = 3.875e-9 #kg/m^3
            z0 = 59.06e3 #m

        try:
            rho = rho0 * exp(-z/z0)
```

```
        except:
            rho = 1000 #kg/m^3
        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:

            landa=3e-3
            T25=216.65
            P25=2481
            rho25=0.0399

            T=T25+landa*(h-25000)
            P=P25*(T/T25)**(-g/(R*landa))
            rho=rho25*(T/T25)**(-g/(R*landa)-1)

    elif h<180e3 :

            #CIRA model
            a0 =  7.001985e-2
            a1 = -4.336216e-3
            a2 = -5.009831e-3
            a3 =  1.621827e-4
            a4 = -2.471283e-6
            a5 =  1.904383e-8
            a6 = -7.189421e-11
            a7 =  1.060067e-13

            h = h*1e-3

            polyh = ((((((a7*h + a6)*h + a5)*h + a4)*h + a3)*h + a2)*h + a1)*h + a0

            rho = 10**(polyh)

            #Esto no esta bien
            T = 900 + 2.5 * (F10 - 70) + 1.5*Ap
            P= rho*R*T

```

```
    elif h<=550e3:

        R = 287

        T = 900 + 2.5 * (F10 - 70) + 1.5*Ap
        mu = 27 - 0.012 * (h - 200)
        H = T / mu
        try:
            rho = 6*10e-10*exp( - (h - 175) / H )
        except:
            rho = 0

        #Esto no esta bien
        P= rho*R*T

    else:
        #print('Atmósfera no apreciable')

        rho=0
        P=0
        T=0

    return P, rho, T

def isa_P(z):
    P, rho, T = isa_altitude(z)
    return P

def isa_rho(z):
    P, rho, T = isa_altitude(z)
    return rho

def isa_T(z):
    P, rho, T = isa_altitude(z)
    return T
```

Bibliografía

[1] temisAP/Trabajos_AAV_123.
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