

FARADAY ROCKETRY UPV

MANUAL DE LA AUTOMATIZACIÓN DE DATCOM

VERSIÓN 2.0

+

GUÍA DE SIMULACIÓN EN ROCKETPY

Valencia, marzo de 2025

Índice

1. Introducción	1
2. Generar los coeficientes de un cohete mono-etapa	2
2.1. Añadir un aerofreno	5
3. Generar los coeficientes de un cohete doble-etapa	6
4. Generar los coeficientes de un booster	7
5. Cómo utilizar estos coeficientes	8
5.1. RocketPy	8
5.1.1. Power on/off drag	8
5.1.2. <i>LinearGenericSurface</i>	8
5.2. ASTOS	9
6. Cómo usar la automatización de RocketPy	9
7. Métodos de simulación recomendados en RocketPy	12
8. Errores frecuentes	14
8.1. <i>LongitudExcedidaError</i>	14
9. Lista de nuevas versiones	16
9.1. Versión 1.1	16
9.2. Versión 2.0	16
10. Futuras versiones	17

1. Introducción

El objetivo de este documento es explicar detalladamente cómo funciona la versión 2.0 del módulo de automatización de DATCOM desarrollado en el equipo así como sus diferentes formas de uso. Con este documento aprenderás a:

- Generar archivos CSV con los coeficientes de un cohete **mono-etapa** para utilizarlos tanto en ASTOS como en RocketPy (sección 2).
- Generar archivos CSV con los coeficientes de un cohete **doble-etapa** para utilizarlos tanto en ASTOS como en RocketPy (sección 3).
- Generar archivos CSV con los coeficientes de un **booster** (mono-etapa con nariz truncada) para utilizarlos tanto en ASTOS como en RocketPy (sección 4).
- Cómo introducir estos coeficientes tanto en ASTOS como en RocketPy para que la simulación se realice correctamente (sección 5).
- Cómo calcular en tiempo real los coeficientes de un cohete durante una simulación de RocketPy a partir de las variables del vuelo (sección 6).
- Qué métodos se recomienda utilizar al simular en RocketPy y por qué (sección 7).
- Cómo lidiar con algunos errores que puedan surgir (sección 8).

Antes de todo es necesario que tengas en tu ordenador la aplicación de DATCOM. La puedes encontrar en un archivo comprimido ZIP en la carpeta *E25 - EUROOC25/E25_AER - AERODINÁMICA/E25_AER_S05 - DATCOM/DATCOM.zip*. Únicamente es necesario que descomprimas este archivo ZIP y guardes la carpeta *DATCOM* en algún lugar de tu ordenador.

IMPORTANTE: para que no haya problemas asegúrate de que la aplicación de DATCOM, *datcom.exe*, se encuentre en el directorio *DATCOM/datcom.exe*. Al descomprimir la carpeta puede que se te guarde como *DATCOM/DATCOM/datcom.exe*.

Dentro de la carpeta *DATCOM* encontrarás múltiples archivos. Los importantes son *datcom.exe* y *user_manual_mdacm97.pdf*. El primero se trata de la aplicación de DATCOM y el segundo de su documentación. Allí encontrarás toda la información necesaria para ejecutar DATCOM por ti mismo de forma manual. La documentación será imprescindible en el futuro si se necesitan añadir nuevas funciones a este módulo de automatización. Para operar de forma normal con él y hacer uso de sus características actuales no es necesario leerlo en absoluto.

Cuando hablamos del módulo de automatización de DATCOM nos referimos al archivo *datcom.py*. Allí es donde se encuentran definidas todas las funciones que permiten ejecutar la aplicación de DATCOM de forma automática desde un script cualquiera. Cada una de estas funciones se encuentra minuciosamente documentada dentro del propio módulo. En primer lugar, se recoge una explicación de la función que realiza. En segundo lugar, se repasan uno por uno los inputs que necesita. Se especifica el tipo de objeto que hay que introducir y se adjunta una breve descripción del mismo. Por último, se especifica el tipo de output que produce más una rápida explicación. Si tienes cualquier duda abre el módulo y consúltalo.

Cuando hablamos del módulo de automatización de RocketPy nos referimos al módulo *auto_rocketpy.py*. Este módulo contiene todas aquellas funciones que permiten interactuar a RocketPy y a DATCOM de manera automática. Su objetivo es calcular los coeficientes en tiempo real durante el transcurso de la simulación a partir de las variables del vuelo y la geometría del cohete sin que deba realizarse ninguna acción de forma manual.

Por último, cuando hablamos del módulo con los datos del cohete nos referimos a *rocket_data.py*. Este es un módulo que contiene todos los datos del cohete que se pretende simular, además de una función, *load_rocket_data*, que permite cargarlos automáticamente en cualquier script.

Siempre que se use el módulo de automatización de DATCOM el **área de referencia** será la mayor área transversal del cohete y la **longitud de referencia** el mayor diámetro del cohete.

2. Generar los coeficientes de un cohete mono-etapa

Entendemos por cohete mono-etapa a aquel que cuenta únicamente con un motor a bordo. Su estructura está compuesta por un nose cone, un fuselaje y un conjunto de aletas. Ejemplos de un cohete mono-etapa serían ASPERA, ASTRA o GENESIS.

Para generar sus coeficientes es necesario que abras el script *genera_coeficientes_mono_etapa.ipynb*. Los dos primeros bloques de código corresponden con la descripción del script y con la carga de los módulos necesarios para ejecutarlo, entre ellos el módulo de automatización de DATCOM. El tercer bloque de código, que es el que puedes ver en la figura 1, es el que debes modificar.

En primer lugar, debes definir un objeto de la clase DATCOM con la dirección de la carpeta *DATCOM* que contiene a *datcom.exe*. Búscala en tu ordenador y copia el mismo directorio. A continuación, debes cargar los datos del cohete mediante la función *load_rocket_data*. En este caso, esta función tiene un argumento que es el grado de apertura del aerofreno. Esto depende de como se haya definido la función, pero podría no tener ningún argumento de entrada o tener más de uno. Los datos que vas a necesitar definir como mínimo se muestran en la figura 3. Los del aerofreno únicamente serán necesarios si la propiedad de aerofreno se define como *True* (véase la sección 2.1. En el siguiente paso

debes definir las variables independientes para las que se van a calcular los coeficientes. Estos serán función del número de Mach, del ángulo de ataque y del ángulo de derrape.

IMPORTANTE: machs, alphas y betas deben ser listas. Si los defines como otro objeto no va a funcionar. Aunque solo quieras un valor de alguna de estas variables defínelo como lista (p. ej., si quieres solo datos para un ángulo de ataque de 0° , entonces debes escribir `alphas = [0]`).

```
#### RELLENA ESTOS DATOS ANTES DE CONTINUAR ####

# Define la clase DATCOM con la ubicación de la carpeta 'datcom.exe'
datcom = DATCOM('D:\\DATCOM')

# Carga los datos del cohete
delta = 0 # Posición del aerofreno
datos_cohete = data.load_rocket_data(delta)

# Define las variables independientes para las que quieres calcular los coeficientes
machs = [0.01,0.1,0.2,0.5,0.75,1]
alphas = [-2,-1,0,1,2] # En deg
betas = [-2,-1,0,1,2] # En deg

# ¿Qué altitudes corresponden a cada número de Mach?
altitudes = [0,100,200,300,400,500] # En metros

# Motor encendido (True), motor apagado (False)
motor = True
```

Figura 1: Bloque de código que hay que modificar.

En cuarto lugar, debes definir las alturas a las que quieres calcular los coeficientes. Esto es necesario para que DATCOM sea capaz de calcular el número de Reynolds, del cual también dependen los coeficientes. Debe haber el mismo número de altitudes que de números de Mach y deben ser correspondientes. Esto quiere decir que cada número de Mach debe tener una altitud. Para asegurar la precisión de los resultados, esa altitud debe ser aquella a la que el cohete alcanza dicho número de Mach. Para seleccionar las altitudes, es recomendable tener una estimación del gráfico altitud vs mach.

Otra forma de definir el número de Reynolds es introduciendo la presión y la temperatura en cada punto en lugar de la altitud. Si se dispone del pronóstico de estos datos atmosféricos, esta opción resultará más precisa. Con la opción de las altitudes se utiliza la atmósfera US Standard Atmosphere de 1962.

Por último, queda definir si el motor del cohete está encendido o apagado. *True* corresponde con motor encendido y *False* con motor apagado. Principalmente los coeficientes de resistencia dependen mucho de si el motor está encendido o apagado.

En el cuarto, y último, bloque de código se ejecuta DATCOM y se calculan los coeficientes. No es necesario tocar nada aquí.

Los coeficientes se almacenan dentro de la carpeta de *COEFICIENTES/MONO ETAPA*. Estos se generan en dos formatos diferentes, en el formato de DATCOM y en el formato de RocketPy. Deben ser distintos ya que no comparten los mismos ejes. En el caso de ASTOS se utilizan los mismos ejes que DATCOM, por lo tanto, para este simulador se debe utilizar el formato de DATCOM.

Independientemente del formato, se calculan un total de 24 coeficientes. Estos son:

■ DATCOM

- Coeficiente de fuerza axial (CA).
- Coeficiente de fuerza normal (CN).
- Coeficiente de fuerza lateral (CY).
- Coeficiente de momento de cabeceo (CM).
- Coeficiente de momento de guiñada (CLN).
- Coeficiente de momento de alabeo (CLL).

■ RocketPy

- Coeficiente de resistencia (CD).
- Coeficiente de sustentación (CL).
- Coeficiente de fuerza lateral (CQ).
- Coeficiente de momento de cabeceo (Cm).
- Coeficiente de momento de guiñada (Cn).
- Coeficiente de momento de alabeo (Cl).

Además, también se pueden encontrar las derivadas dinámicas con respecto a las velocidades angulares. Véase p para la velocidad de alabeo, q para la de cabeceo y r para la de guiñada.

IMPORTANTE: como puedes observar no se calculan las derivadas de los coeficientes con respecto al ángulo de ataque ni al ángulo de derrape. Esto es porque los coeficientes ya son función de estos ángulos. Esto es más preciso que el cálculo lineal a partir de las derivadas.

2.1. Añadir un aerofreno

Para añadir un aerofreno, simplemente hay que incluir una entrada adicional en la función *datcom.for005_mono_etapa*. Esta nueva entrada se llama 'aerofreno' y admite dos valores: 'False' si no se desea evaluar el aerofreno (valor por defecto) y 'True' para evaluar el efecto aerodinámico del aerofreno. El uso de esta nueva entrada se puede ver en la figura 2.

```
# Instrucciones DATCOM
for005 = datcom.for005_mono_etapa(datos_cohete,machs,alphas,betas,altitudes,motor_on=motor,aerofreno=True)
datcom.escribir_for005(for005)
datcom.ejecutar_datcom()
for004 = datcom.leer_for004()
coeficientes_datcom = datcom.definir_coeficientes_datcom(for004,machs,alphas,betas)
coeficientes_rocketpy = datcom.transformar_coeficientes_a_rocketpy(coeficientes_datcom,machs,alphas,betas)
datcom.generar_csvs(coeficientes_datcom,'COEFICIENTES/MONO ETAPA/DATCOM')
datcom.generar_csvs(coeficientes_rocketpy,'COEFICIENTES/MONO ETAPA/ROCKETPY')
```

Figura 2: Cómo usar *datcom.for005_mono_etapa* para incluir un aerofreno.

Además, para poder realizar el cálculo del aerofreno en DATCOM se deberán introducir datos adicionales relacionados con su geometría como, por ejemplo, la posición longitudinal, el número de superficies, el espesor, la anchura y la altura. El diccionario con los datos del cohete necesario en este caso se muestra en la figura 3.

```
datos_mono_etapa = {
    'radius':datos_cohete.Skybreaker_radius,
    'center_of_dry_mass':datos_cohete.Skybreaker_center_of_dry_mass_position,
    'RHR':datos_cohete.Skybreaker_RHR,
    'tipo_nose_cone':datos_cohete.Skybreaker_nosecone_DATCOM,
    'longitud_nose_cone':datos_cohete.Skybreaker_nosecone_length1,
    'longitud':datos_cohete.Skybreaker_length,
    'fins_span':datos_cohete.Skybreaker_fins_span,
    'fins_root_chord':datos_cohete.Skybreaker_fins_root_chord,
    'fins_tip_chord':datos_cohete.Skybreaker_fins_tip_chord,
    'fins_posicion':datos_cohete.Skybreaker_fins_position,
    'fins_sweep_lenght':datos_cohete.Skybreaker_fins_sweep_length,
    'fins_number':datos_cohete.Skybreaker_fins_number,
    'fins_roll_position':[45,135,225,315],
    'fins_zupper':datos_cohete.Skybreaker_fins_zupper,
    'fins_lmaxu':datos_cohete.Skybreaker_fins_lmaxu,
    'fins_lflatu':datos_cohete.Skybreaker_fins_lflatu,
    'motor_nozzle_radius':datos_cohete.Skybreaker_motor_nozzle_radius,
    'aerofreno_posicion':datos_cohete.Skybreaker_aerofreno_posicion,
    'aerofreno_n_superficies':datos_cohete.Skybreaker_aerofreno_n_superficies,
    'aerofreno_espesor':datos_cohete.Skybreaker_aerofreno_espesor,
    'aerofreno_altura':datos_cohete.Skybreaker_aerofreno_altura,
    'aerofreno_ancho':datos_cohete.Skybreaker_aerofreno_ancho,
}
```

Figura 3: Cómo introducir los datos del cohete con el aerofreno.

Las nuevas variables representan lo siguiente:

- **'aerofreno_posicion'**. Distancia longitudinal desde la punta de la ojiva hasta el aerofreno.
- **'aerofreno_n_superficies'**. Número de superficies aerodinámicas del aerofreno.
- **'aerofreno_espesor'**. Espesor de la superficie del aerofreno.
- **'aerofreno_altura'**. Distancia desde el fuselaje del cohete hasta la punta de la superficie del aerofreno.
- **'aerofreno_ancho'**. Ancho del aerofreno. Se recomienda que este parámetro se calcule según Skyward Experimental Rocketry (2021) en la sección 2.6.2.

3. Generar los coeficientes de un cohete doble-etapa

Entendemos por cohete doble-etapa a aquel que cuenta con dos motores a bordo. Un ejemplo podría ser el cohete ORIGIN.

El procedimiento para calcular sus coeficientes es prácticamente análogo al del caso del cohete mono-etapa, así que nos vamos a centrar únicamente en las diferencias. Este procedimiento se encuentra recogido en *genera_coeficientes_doble_etapa.ipynb*.

En primer lugar, DATCOM no permite definir este tipo de cohetes (con interstage) a partir de cuerpos predefinidos, por lo que habrá que definir su contorno a partir de los puntos en el plano longitudinal-radial. Este procedimiento, debido a los métodos de cálculo que utiliza DATCOM, hace que los resultados no sean precisos para valores de $M > 1$.

La primera diferencia vendrá justo en este aspecto, en el módulo de los datos del cohete, donde se necesitarán cargar los puntos del contorno del cohete. Para ello se deberán definir dos listas de puntos, una con las coordenadas longitudinales y otra con las radiales (véase cómo en la figura 4). Para hacerse una idea de cómo calcular las coordenadas de estos puntos del contorno véase el módulo de datos del cohete ORIGIN, *datos_ORIGIN.py*.

En cuanto al diccionario con la geometría del cohete, este será muy similar al del cohete mono-etapa, pero con algunos datos adicionales.

Por último, las instrucciones de ejecución de DATCOM, en el cuarto bloque, han de tener en cuenta que las funciones a utilizar deben ser las especializadas en cohetes doble-etapa.

```
contorno_cohete_x = (  
    list(datos_cohete.nc_x)  
    + [datos_cohete.css_posicion]  
    + [datos_cohete.css_posicion + datos_cohete.css_longitud]  
    + [datos_cohete.interstage_posicion]  
    + [datos_cohete.interstage_posicion + datos_cohete.interstage_longitud]  
    + [datos_cohete.origin_longitud]  
)  
contorno_cohete_r = (  
    list(datos_cohete.nc_r)  
    + [datos_cohete.sustainer_radius_superior]  
    + [datos_cohete.sustainer_radius_inferior]  
    + [datos_cohete.sustainer_radius_inferior]  
    + [datos_cohete.booster_radius]  
    + [datos_cohete.booster_radius]  
)
```

Figura 4: Cómo definir el contorno de un doble-etapa.

4. Generar los coeficientes de un booster

Entendemos por booster a un cohete mono-etapa pero con la nariz truncada debido al interstage.

El procedimiento para calcular los coeficientes de este tipo de cohetes es análogo al del cohete

mono-etapa. La única diferencia es que hay que activar la opción de truncado en la función *datcom.for005_mono_etapa* y proporcionar los datos geométricos de dicho truncamiento. El script *genera_coeficientes_booster.ipynb* es el que se debe usar en este caso. El motor permanece apagado por defecto.

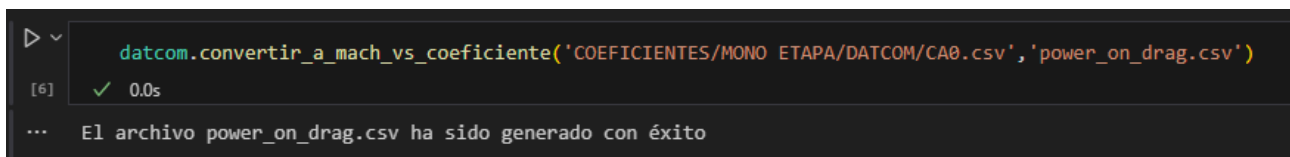
5. Cómo utilizar estos coeficientes

5.1. RocketPy

5.1.1. Power on/off drag

Para generar los CSV con los coeficientes de resistencia power on y power off de RocketPy existe una función concreta del módulo de automatización, esta es *datcom.convertir_a_mach_vs_coeficiente*.

Para obtenerlos, primero se han de calcular los coeficientes, ya sea con el motor encendido o apagado. A continuación, habrá que ejecutar la función tal y como se muestra en a figura 5.



```
datcom.convertir_a_mach_vs_coeficiente('COEFICIENTES/MONO ETAPA/DATCOM/CA0.csv', 'power_on_drag.csv')
```

[6] ✓ 0.0s

... El archivo power_on_drag.csv ha sido generado con éxito

Figura 5: Cómo generar el CSV *power_on_drag.csv* de RocketPy.

Como se puede observar, primero habrá que introducir la ubicación del coeficiente CA0. Para ángulos de ataque y derrape nulos, CA0 en el formato DATCOM y CD0 en el formato RocketPy son exactamente iguales, ya que el primero trabaja en ejes Cuerpo y el segundo en ejes Viento. Cualquiera de los dos sería válido. En segundo lugar, habrá que introducir la ubicación y el nombre con el que se quiere guardar el CSV resultante. En este caso, se ha generado en la misma carpeta en la que se encuentra el script.

IMPORTANTE: los coeficientes hay que calcularlos únicamente para alpha y beta nulos.

De no ser así, aparecerán varios valores del coeficiente para un mismo número de Mach.

5.1.2. *LinearGenericSurface*

Para introducir todos los coeficientes aerodinámicos calculados con DATCOM en RocketPy hay que hacer uso de una de las funciones del módulo de automatización, esta es *datcom.ubicacion_csv_coeficientes*. En la figura 6 se muestra cómo implementarla.

En primer lugar hay que introducir el área de referencia y la longitud de referencia. Siempre que se use el módulo de automatización de DATCOM el **área de referencia** será la mayor área transversal del cohete y la **longitud de referencia** el mayor diámetro del cohete.

```
ORIGIN_on_nominal = LinearGenericSurface(  
    reference_area= np.pi * datos_cohete.booster_radius**2,  
    reference_length= 2*datos_cohete.booster_radius,  
    coefficients= datcom.ubicacion_csvs_coeficientes('COEFICIENTES/ORIGIN_motor_on_nominal'),  
    name= ' Coeficientes ORIGIN on',  
)
```

Figura 6: Cómo implementar los coeficientes en *LinearGenericSurface*.

En el apartado de coeficientes hay que añadir un diccionario con la ubicación de cada uno de los archivos CSV. Para automatizar esta tarea se utiliza la función del módulo de automatización mencionada anteriormente. Como argumento únicamente hay que introducir la ubicación del directorio donde se encuentran todos los coeficientes y automáticamente se creará el diccionario con todos los coeficientes. Por ejemplo, en el caso de la sección 2 la carpeta con los coeficientes sería *COEFICIENTES/MONO ETAPA/ROCKETPY*.

Como nombre se puede utilizar cualquiera, no afecta en nada a la simulación.

5.2. ASTOS

Aún no se ha recibido ningún feedback por parte de ASTOS, así que no es posible asegurar que el formato de DATCOM se pueda introducir directamente en ASTOS.

Probablemente en el futuro haya que realizar algunos ligeros cambios para acomodar el formato de DATCOM al formato aceptado por ASTOS.

6. Cómo usar la automatización de RocketPy

El módulo de automatización de RocketPy consiste en un conjunto de funciones que permiten la interacción entre RocketPy y DATCOM para que los coeficientes se calculen de forma automática durante la simulación del vuelo. De esta forma, no será necesario calcularlos previamente. Esto resultará especialmente útil para implementar algoritmos de control en cuyas simulaciones los coeficientes variarán constantemente. También resultará útil para realizar estudios de optimización en los que se hagan variar una serie de parámetros del cohete de forma recurrente para buscar la combinación más óptima. De no existir esta automatización, los coeficientes deberían calcularse a mano para cada una de las combinaciones simuladas.

Este módulo permite dos métodos diferentes de cálculo y simulación:

- **Método del *drag*.** El único coeficiente que se calcula automáticamente es el de resistencia. En RocketPy, el parámetro *power on/off drag*. Consecuentemente, el resto de coeficientes se obtendrán a partir de las superficies aerodinámicas predefinidas de RocketPy.

- **Método de coeficientes.** Todos los coeficientes aerodinámicos del cohete se calculan automáticamente y se introducen en la simulación a través de una *LinearGenericSurface*.

En ambos casos se calcula un valor constante para los coeficientes a partir de las condiciones del vuelo, por lo que la frecuencia de ejecución de DATCOM será muy importante de cara a la precisión de los resultados. Este parámetro debe estar definido en el módulo de datos del cohete como *frequency* y se define como la frecuencia con la que quieren recalcularse los coeficientes aerodinámicos. En hercios.

Independientemente del método que se decida utilizar, primero de todo, habrá que inicializar la clase *Flight*. Esto significa simular la fase del raíl. Esto es necesario ya que actualmente en *RocketPy* no es posible iterar el vuelo en esta fase ya que cuando se introduce como solución inicial una clase *Flight* previa se considera automáticamente que el cohete ha salido del raíl.

A continuación, ya hay que definir el bucle de la simulación de vuelo. Para finalizar este bucle debe elegirse alguna condición. En el caso de la figura 8, se ha escogido que la simulación termine cuando el cohete llega al apogeo, pero también podría escogerse la condición de que el cohete impacte con el suelo con *flight.impact_time == 0*. En el interior del bucle se debe ejecutar la simulación de vuelo.

Entre los argumentos que hay que introducir a la simulación de vuelo se encuentran:

- **data.** Se trata del diccionario con los datos del cohete generado como resultado de la función *load_rocket_data*.
- **results.** Objeto de la clase *Results* (definida en *auto_rocketpy.py*) en el que se desea que se almacenen los datos de la simulación. Este objeto se inicializa en *flight_initialization*.
- **rocket.** Objeto de la clase *Rocket* que contiene las características del cohete que se pretende simular. En este respecto es necesario aclarar ciertas cosas. El cohete se debe configurar en consonancia con el método de simulación que se vaya a realizar. Si se desea emplear el **método del drag** habrá que definir las superficies aerodinámicas de *RocketPy*. Si se desea emplear el **método de coeficientes** no habrá que definir las. Hay que notar que en este segundo caso tampoco será necesario definir los coeficientes en ningún momento. Se recomienda que las propiedades *power on/off drag* se definan como nulas. Esto solo será vigente durante el raíl y es una aproximación bastante acertada ya que la velocidad es baja. En la figura 7 se puede ver cómo definir este objeto.

```
# SE DEFINE EL COHETE
```

```
rocket = Rocket(  
    radius = data['radius'],  
    mass = data['no_motor_mass'],  
    inertia = (  
        data['inercia_I11'],  
        data['inercia_I11'],  
        data['inercia_I33'],  
    ),  
    power_off_drag = 0,  
    power_on_drag = 0,  
    center_of_mass_without_motor = data['cdg_no_motor'],  
    coordinate_system_orientation = "nose_to_tail",  
)  
rocket.add_motor(motor, data['longitud'])  
rocket.add_surfaces(nose_cone, 0) # Si método drag  
rocket.add_surfaces(aletas, data['fins_posicion']) # Si método drag
```

Figura 7: Cómo definir la clase *Rocket*.

- **env.** Objeto de la clase *Environment* con las condiciones atmosféricas.
- **flight.** Solución inicial. Se trata del objeto de la clase *Flight* de la iteración anterior.
- **datcom.** Objeto de la clase *DATCOM* inicializado previamente con la dirección del directorio de *datcom.exe*.
- **method.** El que se desee *drag* o *coefficients*.

El bucle de la simulación consiste en lo siguiente. A partir de las condiciones iniciales *flight* y el *environment* se recalculan los coeficientes aerodinámicos. A continuación, se define la duración de la simulación, que no será más que la inversa de la frecuencia. Por último, se ejecuta la simulación de vuelo durante ese tiempo y se almacenan los datos en un objeto de la clase *Results*. La función devuelve el objeto *results* actualizado y el resultado de la simulación de vuelo *flight* para repetir el bucle. El objeto *flight* se puede reutilizar fuera del bucle por si, por ejemplo, se pretende calcular el descenso con un paracaídas.

Por último, se exportan los resultados de la simulación de vuelo en formato .csv. Es importante notar que es *results* el objeto que contiene todos los resultados de la simulación. El objeto *flight* solo contiene los resultados del último bucle.

El procedimiento completo se puede observar en la figura 8.

```
## BUCLE DE LA SIMULACIÓN

# Inicialización de la simulación
results,flight = sim.flight_initialization(data,rocket,env)

# Bucle de la simulación
while flight.apogee_time == 0:

    results,flight = sim.flight_simulation(
        data,
        results,
        rocket,
        env,
        flight,
        datcom,
        method='coefficients',
    )

results.export('auto_coefficients_test.csv','t','x','y','z','vx','vy','vz','w1','w2','w3','R1','R2','R3','M1','M2','M3')
```

Figura 8: Bucle de la simulación de vuelo con la automatización de RocketPy.

Se puede consultar la implementación de ambos métodos en los scripts *metodo_drag.ipynb* y *metodo_coeff.ipynb*.

7. Métodos de simulación recomendados en RocketPy

El objetivo de esta sección es establecer los métodos de simulación recomendados en RocketPy a partir de la experiencia obtenida en el equipo hasta el momento para agilizar el trabajo futuro y evitar que se pierda el tiempo utilizando prácticas ya demostradas como ineficientes.

En primer lugar, se van a definir los métodos de simulación disponibles hasta el momento:

- **Simulación original.** Consiste en definir las superficies aerodinámicas predeterminadas de RocketPy y en introducir el coeficiente de resistencia como *power on/off drag*. Un ejemplo de implementación de este método se puede ver en *simulacion.ipynb*.
- **Simulación con coeficientes.** En vez de definir las superficies aerodinámicas predefinidas se definen los coeficientes aerodinámicos como función del mach, el ángulo de ataque y el ángulo de derrape mediante *LinearGenericSurface*. Un ejemplo de implementación se puede ver en *simulacion_coeff.ipynb*.
- **Método del drag.** Los coeficientes *power on/off drag* se calculan automáticamente durante la simulación de vuelo.
- **Método de coeficientes.** Todos los coeficientes aerodinámicos se calculan automáticamente durante la simulación de vuelo.

¿Cuáles son las características de cada método?:

	Tiempo Ejecución	Precisión
Original	<1s	Alta
S. Coeficientes	<10s	Alta
Método Drag	<3min	Alta
Método Coeficientes	>20min	Baja

¿Qué conclusiones hemos extraído?:

- Tanto por precisión de los resultados como por tiempo de ejecución, a la hora de hacer una única **simulación nominal**, lo más eficiente es utilizar el método original de RocketPy. Las simulaciones con coeficientes son también muy precisas pero ya son un poco más lentas y engorrosas en el sentido de que hay que calcular bastantes coeficientes aerodinámicos. Si no se dispone de los coeficientes *power on/off drag*, y hay que calcularlos, el método de *drag* puede ser más rápido.
- La **diferencia entre el método original y el de coeficientes** es que el segundo es más sensible al viento y por lo tanto se desvía más, pero el **apogeo es muy similar** en ambos casos.
- Las simulaciones con coeficientes a veces **divergen durante la caída balística**, por lo que se recomienda no utilizarlas en este caso.
- Para **simulaciones en casos de fallo** si que puede ser muy interesante simular con coeficientes, sobre todo si no hay otra forma de modelar el cohete, como es el caso de fallo de una aleta.
- Para **simular el control del aerofreno** se puede usar tanto el método de drag como el método original con la clase *AirBrakes*.
- El **método de drag** empieza a dar resultados altamente precisos a partir de una frecuencia de cálculo de 10Hz.
- El **método de automatización de RocketPy con coeficientes** no es recomendable en ningún caso debido a su alto coste computacional. Además, es bastante poco preciso ya que requiere de frecuencias altísimas de cálculo para que la solución no diverja. Hay que tener en cuenta que los coeficientes calculados son constantes y si la frecuencia de refresco no es lo suficientemente alta el cohete puede volverse inestable. Resultados con sentido se pueden obtener a partir de 100Hz, lo que supone un cálculo de aproximadamente 30min. Para simulaciones realmente precisas se requeriría aún más que eso. Debido a que el resto de métodos son bastante precisos, este método no se recomienda en ningún caso.

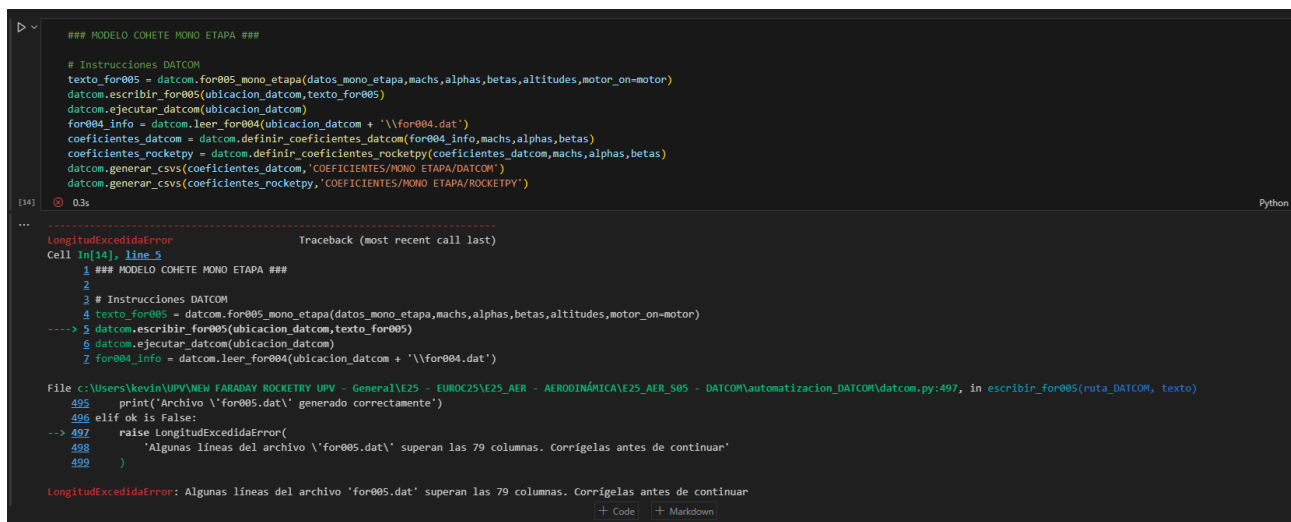
- Las simulaciones se puede realizar también desde **MATLAB** si ese lenguaje es más cómodo para ti. Se deja un ejemplo de simulación en MATLAB en el script *simulacion_matlab.m*.

8. Errores frecuentes

8.1. *LongitudExcedidaError*

En el módulo de automatización de DATCOM se ha creado un error propio con el nombre de *LongitudExcedidaError*. El motivo de definir este error es que DATCOM solo es capaz de leer el archivo *for005.dat* hasta la columna 79. Toda aquella información que exceda la columna 79 no se lee. Es por eso que para evitar posibles fallos se ha incorporando este error a modo de alarma.

No es normal exceder la columna 79. Solo ocurrirá en contadas ocasiones cuando se pidan muchos números de Mach o ángulos de ataque. Si se pide un número razonable es muy raro que llegue a saltar este error, el cual se puede observar en la figura 9.



```

## MODELO COHETE MONO ETAPA ##

# Instrucciones DATCOM
texto_for005 = datcom.for005_mono_etapa(datos_mono_etapa,machs,alphas,betas,altitudes,motor_on=on-motor)
datcom.escribir_for005(ubicacion_datcom,texto_for005)
datcom.ejecutar_datcom(ubicacion_datcom)
for004_info = datcom.leer_for004(ubicacion_datcom + '\\for004.dat')
coeficientes_datcom = datcom.definir_coeficientes_datcom(for004_info,machs,alphas,betas)
coeficientes_rocketpy = datcom.definir_coeficientes_rocketpy(coeficientes_datcom,machs,alphas,betas)
datcom.generar_csvs(coeficientes_datcom,'COEFICIENTES/MONO ETAPA/DATCOM')
datcom.generar_csvs(coeficientes_rocketpy,'COEFICIENTES/MONO ETAPA/ROCKETPY')

[14]: 0.3s Python

LongitudExcedidaError                                Traceback (most recent call last)
Cell In[14], line 5
      1 ## MODELO COHETE MONO ETAPA ##
      2
      3 # Instrucciones DATCOM
      4 texto_for005 = datcom.for005_mono_etapa(datos_mono_etapa,machs,alphas,betas,altitudes,motor_on=on-motor)
----> 5 datcom.escribir_for005(ubicacion_datcom,texto_for005)
      6 datcom.ejecutar_datcom(ubicacion_datcom)
      7 for004_info = datcom.leer_for004(ubicacion_datcom + '\\for004.dat')

File c:\Users\Kevin\UPV\VEN FARADAY ROCKETRY UPV - General\VE25 - EURO25\VE25_AER - AERODINÁMICA\VE25_AER_S05 - DATCOM\automatizacion_DATCOM\datcom.py:497, in escribir_for005(ruta_DATCOM, texto)
    495     print('Archivo \'for005.dat\' generado correctamente')
    496     elif ok is False:
--> 497         raise LongitudExcedidaError(
    498             'Algunas líneas del archivo \'for005.dat\' superan las 79 columnas. Corrigelas antes de continuar'
    499         )

LongitudExcedidaError: Algunas líneas del archivo 'for005.dat' superan las 79 columnas. Corrigelas antes de continuar
+ Code | + Markdown

```

Figura 9: Alarma de *LongitudExcedidaError*.

Este no es un error catastrófico, lo que quiere decir que se puede solucionar fácilmente. Para ello, deberás seguir las siguientes instrucciones:

1. **Abre el archivo *for005.dat*.** Este se encontrará en tu carpeta *DATCOM* donde se aloja la aplicación *datcom.exe*.
2. **Observa que efectivamente hay una línea que excede la columna 79.** Como la que puedes ver en la figura 10.


```

$FLTCON NALPHA=1.0,$
$FLTCON ALPHA=0.0,$
$FLTCON NMACH=17.0,$
$FLTCON MACH=0.2,0.5,0.75,1.0,1.25,1.5,1.75,2.0,2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9,$
$FLTCON BETA=0.0,$
$REFQ SREF=0.00515,LREF=0.081,LATREF=0.081,XCG=1.197,$
$REFQ BLAYER=TURB,RHR=400.0,$
$AXIBOD TNOSE=HAACK,LNOSE=0.45,DNOSE=0.081,$

```

Figura 10: Se puede observar como la cuarta línea excede la columna 79.

3. **Separa esa línea en dos filas.** De modo que se reduzca su longitud. Para ello hay que cortar la fila en una coma, añadir un símbolo \$ al final y, en la fila siguiente, retomar la definición de la variable. No olvides indicarle a DATCOM en qué número de la lista te habías quedado. Si no se lo indicas te sobrescribirá los valores. En la figura 11 se puede ver como proceder.

```

$FLTCON NALPHA=1.0,$
$FLTCON ALPHA=0.0,$
$FLTCON NMACH=17.0,$
$FLTCON MACH=0.2,0.5,0.75,1.0,1.25,1.5,1.75,2.0,$
$FLTCON MACH(9)=2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9,$
$FLTCON BETA=0.0,$
$REFQ SREF=0.00515,LREF=0.081,LATREF=0.081,XCG=1.197,$
$REFQ BLAYER=TURB,RHR=400.0,$
$AXIBOD TNOSE=HAACK,LNOSE=0.45,DNOSE=0.081,$
$AXIBOD LGCENTR=1.47052,PGCENTR=0.081,$

```

Figura 11: Corrección de la cuarta línea.

4. **Ejecuta las instrucciones que quedaban.** El problema debería desaparecer y los coeficientes deberían calcularse correctamente. Para saber que instrucciones quedan por ejecutar observa la figura 12.

```
datcom.ejecutar_datcom(ubicacion_datcom)
for004_info = datcom.leer_for004(ubicacion_datcom + '\\for004.dat')
coeficientes_datcom = datcom.definir_coeficientes_datcom(for004_info,machs,alphas,betas)
coeficientes_rocketpy = datcom.definir_coeficientes_rocketpy(coeficientes_datcom,machs,alphas,betas)
datcom.generar_csvs(coeficientes_datcom,'COEFICIENTES/MONO ETAPA/DATCOM')
datcom.generar_csvs(coeficientes_rocketpy,'COEFICIENTES/MONO ETAPA/ROCKETPY')
```

[17] ✓ 0.5s

... ¡DATCOM se ha ejecutado con éxito!
Archivo 'for004.dat' leído con éxito
Coeficientes de DATCOM generados con éxito
Coeficientes de RocketPy generados con éxito
Archivos CSV generados con éxito en: COEFICIENTES/MONO ETAPA/DATCOM
Archivos CSV generados con éxito en: COEFICIENTES/MONO ETAPA/ROCKETPY

Figura 12: Ejecuta las instrucciones que quedan y genera los coeficientes exitosamente.

9. Lista de nuevas versiones

9.1. Versión 1.1

En esta versión se ha añadido la posibilidad de incluir un aerofreno en el cálculo de los coeficientes aerodinámicos de un cohete mono-etapa. Las modificaciones, respecto de la versión 1.0, se encuentran únicamente en la función *datcom.for005_mono_etapa*:

- Se ha añadido una nueva entrada llamada 'aerofreno'. Su objetivo es introducir un nuevo método para incorporar los aerofrenos en el cálculo aerodinámico.
- Se ha actualizado la documentación de la función.
- Se ha incluido el método del aerofreno.

El correcto funcionamiento de los cambios ha sido validado.

9.2. Versión 2.0

En esta versión se han desarrollado nuevas funciones que permiten obtener los coeficientes del cohete en tiempo real a partir de las variables del vuelo calculadas en cada paso de integración de una simulación. El objetivo final de estas funciones es automatizar completamente las simulaciones de RocketPy, ya que no será necesario definir los archivos CSV con los coeficientes previamente. Únicamente, se deberán introducir los datos del cohete y ya está. Esto no es solamente útil para las labores de simulación de vuelo, sino también para el desarrollo y validación del aerofreno, ya que permitirá variar automáticamente los coeficientes a medida que el algoritmo de control extiende las superficies del aerofreno. Esta automatización también aumentará la precisión de las simulaciones ya que los coeficientes se calcularán exactamente con las condiciones reales del vuelo. Anteriormente, se tenían que

calcular únicamente para ciertos valores del número de Mach, ángulo de ataque y de derrape y luego se interpolaban linealmente. También se han corregido ligeramente otras funciones del módulo para adaptarse a esta nueva forma de trabajar. El correcto funcionamiento de los cambios y de las nuevas funciones implementadas ha sido validado.

10. Futuras versiones

La idea es ir mejorando y ampliando la librería de automatizaciones a medida que las vayamos necesitando en el equipo. Es por ello que el feedback es **esencial** de cara a mejorar el módulo.

Por favor, si encuentras cualquier error en el código comunícalo en seguida al equipo y lo resolveremos cuanto antes. También, si hay alguna característica que no tiene el módulo todavía pero que te gustaría que tuviese en el futuro, dilo. La idea de todo esto es facilitarnos el trabajo lo máximo posible, así que cuanto antes lo digas, antes tendremos lista la función para hacer tu tarea de forma automática.

Espero que te sea útil y que puedas dedicar tu tiempo a aquello que realmente aporta valor.

Kevin Alcañiz

Referencias

Skyward Experimental Rocketry (2021). Project Lynx - Technical Report. *EuRoC*.