Modelado y simulación de una montaña rusa

Análisis dinámico en 2D y 3D mediante interpolación spline

Miguel Enterría Lastra
Grado en Física
Universidad de Oviedo

11 de enero de 2026

Índice

1. Introducción **3**

2. Modelo físico **3**

3. Descripción geométrica del recorrido **3**
3.1. Trayectoria 2D 3
3.1.1. Parametrización 3
3.1.2. Comparación de métodos 4
3.1.3. Análisis dinámico 5
3.1.4. Consideraciones de seguridad 9
3.2. Trayectoria 3D 10
3.2.1. Parametrización 10
3.2.2. Comparación de métodos 11
3.2.3. Análisis dinámico 11

4. Código **12**

1. Introducción

En esta práctica se estudia el comportamiento dinámico de un vagón que se desplaza a lo largo de una montaña rusa, modelada geométricamente mediante curvas paramétricas interpoladas con B-splines. El estudio se realiza primero en dos dimensiones y posteriormente se generaliza a tres dimensiones, permitiendo analizar trayectorias más realistas que incluyen giros y torsión.

El objetivo principal es simular el movimiento resolviendo la ecuación diferencial asociada, comparar distintos métodos numéricos y evaluar magnitudes físicas relevantes como velocidad, aceleración, fuerza normal y conservación de la energía.

2. Modelo físico

El vagón se modela como una partícula de masa constante que se desplaza a lo largo de una trayectoria prescrita. Las fuerzas consideradas son:

- Peso.
- Reacción normal de la vía.
- Fuerza de rozamiento.
- Resistencia aerodinámica.

La formulación final conduce a un sistema de ecuaciones diferenciales para el parámetro de la curva $u(t)$ y la velocidad $v(t)$, que se resuelve numéricamente mediante `solve_ivp`.

3. Descripción geométrica del recorrido

La trayectoria de la montaña rusa se define a partir de un conjunto de puntos de control, que posteriormente se interpolan mediante B-splines cúbicos.

3.1. Trayectoria 2D

3.1.1. Parametrización

En el caso bidimensional, el movimiento queda restringido a un plano vertical. La trayectoria incluye elementos básicos como descensos pronunciados, bucles y colinas. Siguiendo la geometría propeusta por el guión.

Se muestra a continuación la curva parametrizada de la siguente manera:

- Descenso inicial marcado por curva sigmoide.
- Loop hecho con un clotoide.
- Colina parametrizada como campana de Gauss.

Todas las geometrías se han hecho utilizando las funciones presentes en pkgcurvas.py

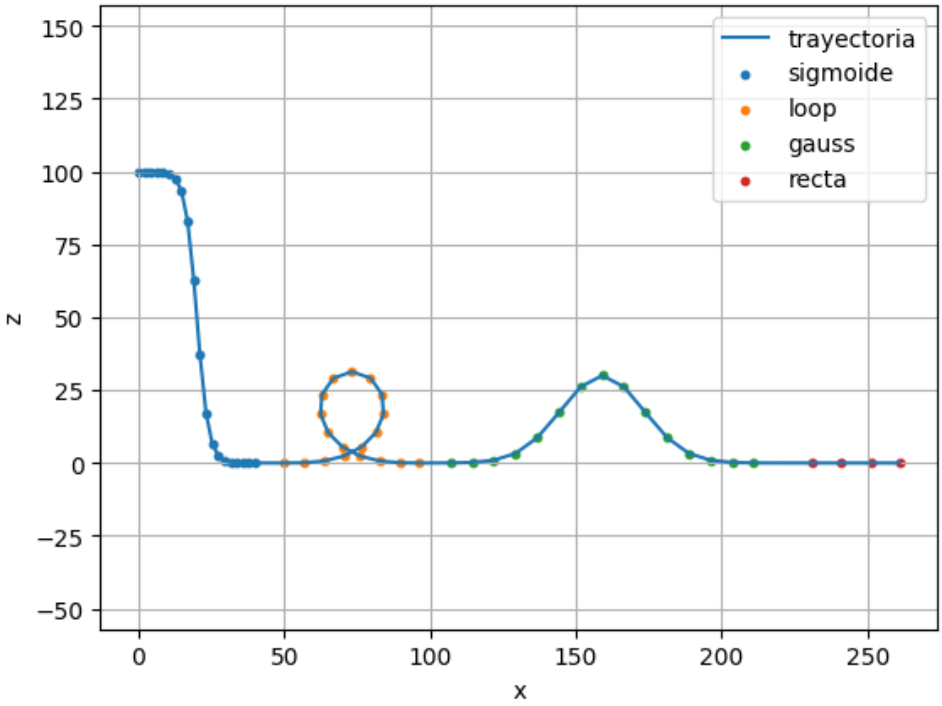

Figura 1: Geometría generada mediante curvas.

Y una vez generada se genera el B-spline interpolador para poder aplicar los métodos de resolución sobre él.

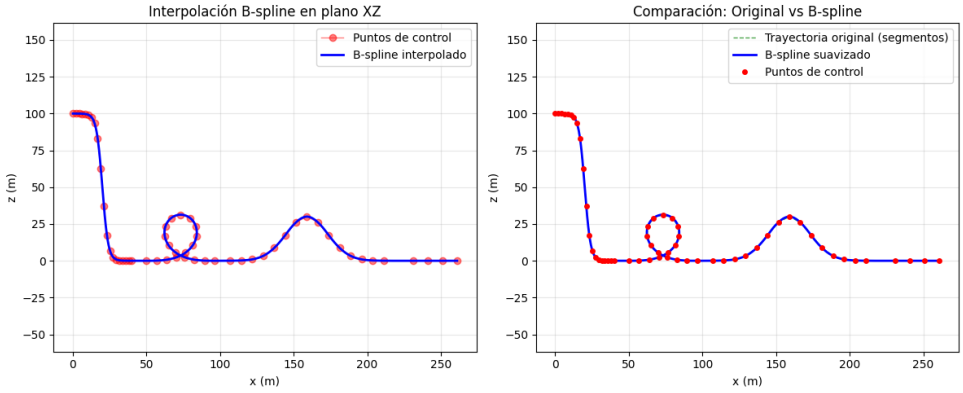

Figura 2: Trayectoria interpolada en 2D.

3.1.2. Comparación de métodos

Ahora ya se tiene lo necesario para empezar con el análisis dinámico del trayecto.
Se hace ahora una comparación entre distintos métodos de resolución de SEDOs, en específico los presentes en el módulo *scipy.integrate*. Para esto se utiliza el caso conservativo y se comprueba cúal es el método que introduce menor error en la conservación de la energía.

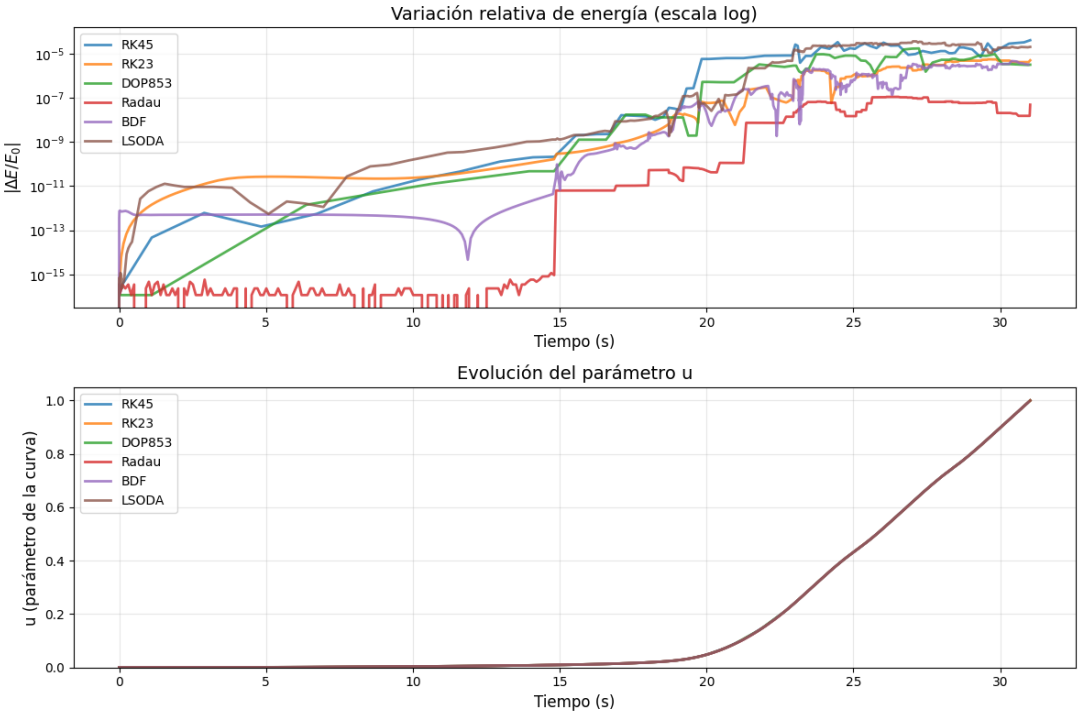

Figura 3: Caption

En estos gráficos se puede observar la evolución de la energía conservada en función del tiempo de simulación.

También se ha generado un gráfico de la parametrizzación (normalizada entre 0 y 1) en función del tiempo para comprobar que ningún método se rompe.ªntes de llegar al final.

Esto nos deja con la siguiente tabla comparativa:

Cuadro 1: Comparación de métodos numéricos en un sistema conservativo

Método	Pasos	t_{final}	u_{final}	$\Delta E_{\text{mín}}$	$\Delta E_{\text{máx}}$	ΔE_{final}
RK45	71	31.019	1.00000	$-3{,}3113 \times 10^{-5}$	$+4{,}0305 \times 10^{-5}$	$+4{,}0305 \times 10^{-5}$
RK23	431	31.019	1.00000	$-5{,}5066 \times 10^{-6}$	$+1{,}7534 \times 10^{-6}$	$-4{,}9371 \times 10^{-6}$
DOP853	71	31.019	1.00000	$-1{,}7090 \times 10^{-5}$	$+3{,}2255 \times 10^{-6}$	$-3{,}1351 \times 10^{-6}$
Radau	505	31.019	1.00000	$-6{,}6773 \times 10^{-8}$	$+1{,}0954 \times 10^{-7}$	$+4{,}8238 \times 10^{-8}$
BDF	829	30.955	0.99367	$-4{,}5920 \times 10^{-6}$	$+3{,}4135 \times 10^{-7}$	$-3{,}1753 \times 10^{-6}$
LSODA	434	31.019	1.00000	$-3{,}5889 \times 10^{-5}$	$+2{,}2398 \times 10^{-6}$	$-2{,}0042 \times 10^{-5}$

Se aprecia a simple vista tanto del gráfico como de la tabla que el método más apropiado es Radau con un error final del orden de (10^{-8})

3.1.3. Análisis dinámico

Una vez seleccionado el método más apropiado de resolución se pasa al análisis dinámico del sistema.

Para esto se van a contemplar tres casos distintos:

1. **Caso no conservativo:** Debería de observarse unáa pérdida mínima de energía.
2. **Caso rozamiento con la vía:** Se va a observar una perdida considerable de la energía por el efecto del rozamiento con la vía.

3. **Caso rozamiento viscoso:** Se contempla además del rozamiento con la vía una pérdida de energía por el aire.

Caso conservativo Las curvas obtenidas son las siguientes:

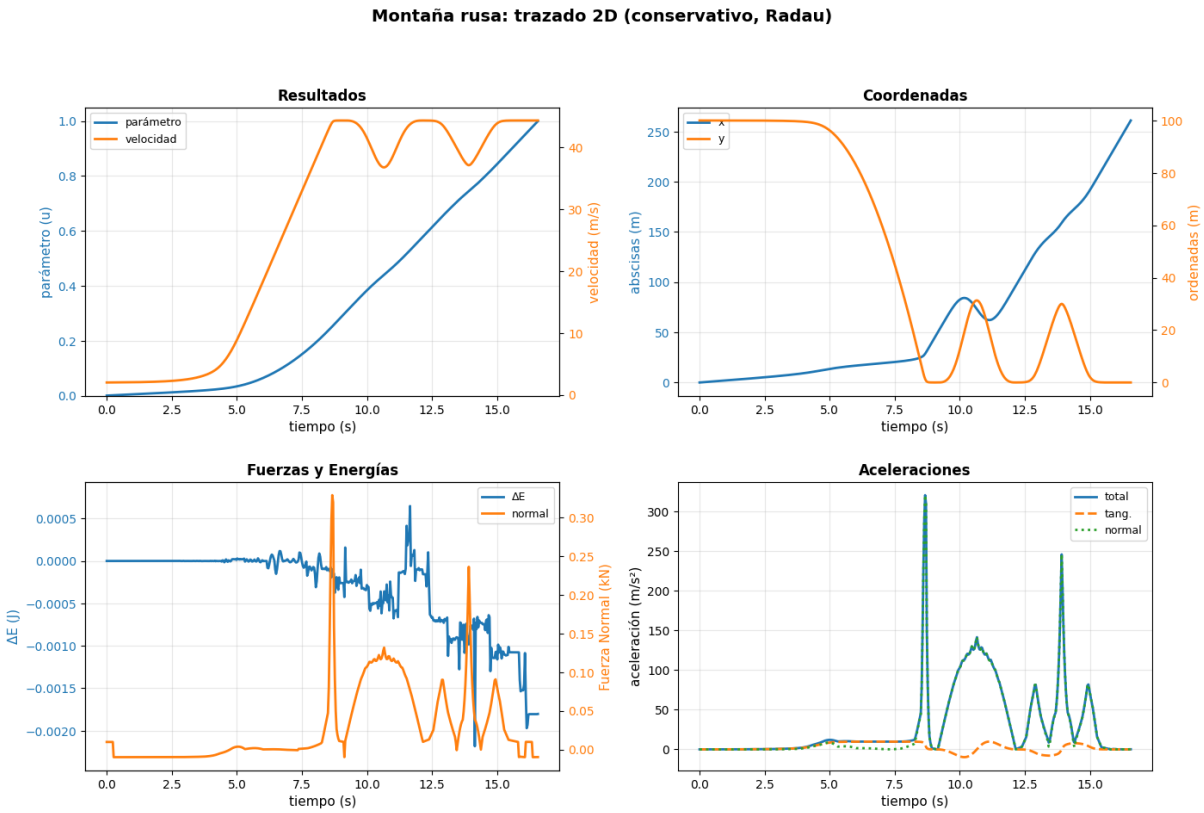

Figura 4: Caption

Se observa lo esperado de estas curvas.

1. Unas pérdidas de velocidad en subida ganadas en las bajadas.
2. Una energía total casi constante. Solo varia 0.002 Julios en todo el recorrido. Debido al modelo.

A continuación se presenta una tabla con las estadísticas del recorrido obtenidas donde se observan valores interesantes del recorrido.

Cuadro 2: Estadísticas del recorrido de la montaña rusa

Magnitud	Valor
Tiempo total	16,57 s
Velocidad máxima	44,34 m/s (159,63 km/h)
Velocidad media	26,85 m/s (96,66 km/h)
Aceleración máxima	320,51 m/s^2 (32,67 g)
Fuerza normal máxima	328,61 N/kg (33,50 g)
Energía inicial	982,96 J/kg
Energía final	982,95 J/kg
Perdida de energía	0,0002 % (conservativo)

Se puede observar que la pérdida de energía es mínima y que las aceleraciones máximas alcanzadas son inadmisibles, esto se estudiará mas a fondo en la sección 3.1.4

Caso no conservativo En los casos no conservativos se introduce una pérdida de energía en forma de fuerza con las siguientes fórmulas:

$$\vec{F}_{\text{roz}} = -\,\mu\, N\, \vec{t} \tag{1}$$

$$\vec{F}_{\text{visc}} = -\,\frac{1}{2}\,\rho\, C_a\, S_f\, v^2\, \vec{t} \tag{2}$$

Estos dos efectos sse introducen a nivel de código generando dos parámetros que se le pasan a la función *solve_ivp*

A continuación se presentan las curvas y estadísticas de los dos casos.

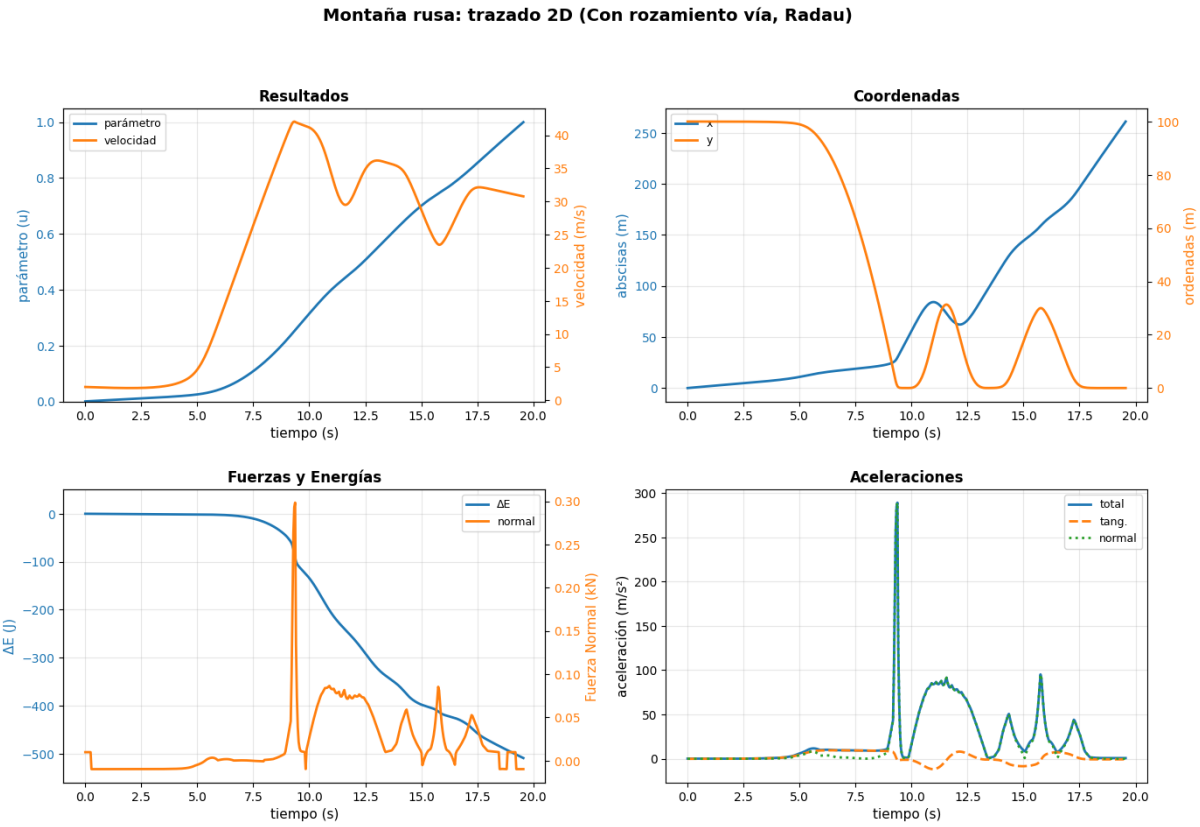

Figura 5: Curvas rozamiento seco.

Cuadro 3: Estadísticas del recorrido con rozamiento con la via

Magnitud	Valor
Tiempo total	18,15 s
Velocidad máxima	43,40 m/s (156,24 km/h)
Velocidad media	24,51 m/s (88,23 km/h)
Aceleración máxima	310,47 m/s^2 (31,65 g)
Fuerza normal máxima	318,59 N/kg (32,48 g)
Energía inicial	982,96 J/kg
Energía final	715,18 J/kg
Pérdida de energía	27,24 %

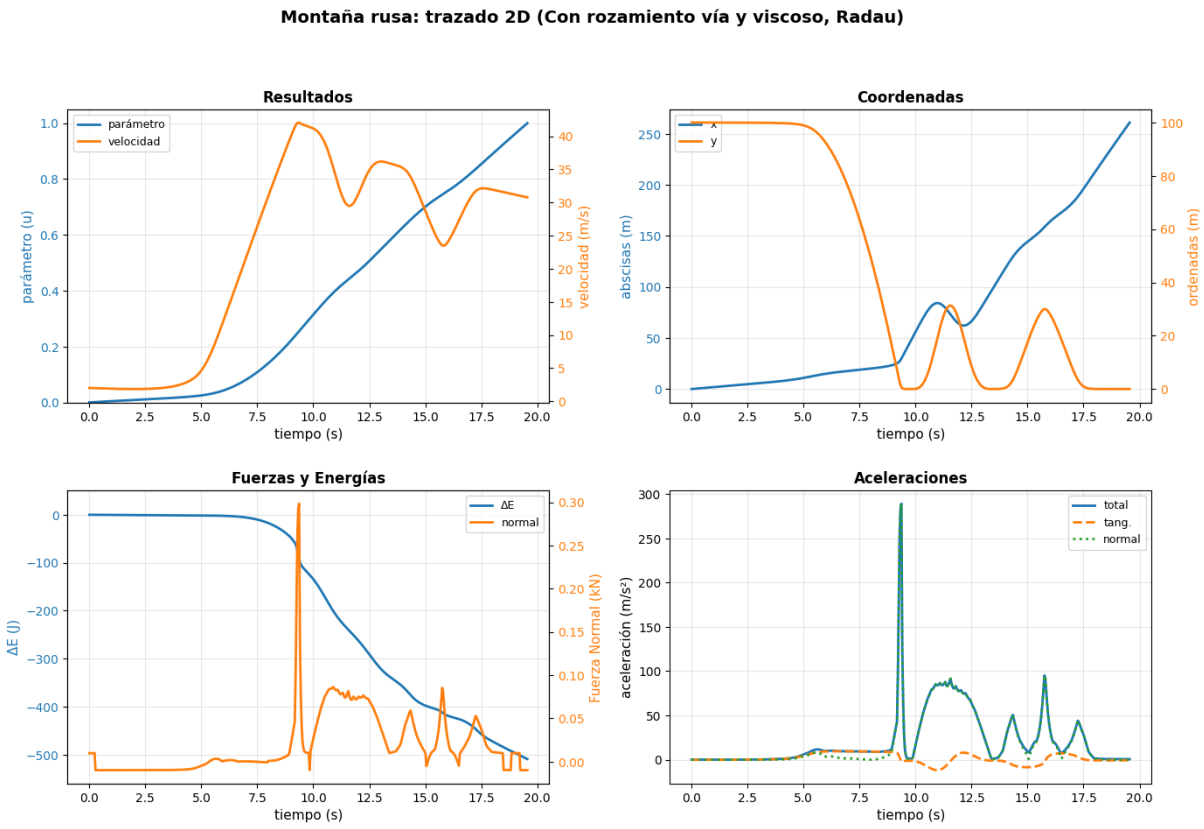

Figura 6: Curvas rozamiento viscoso.

Cuadro 4: Estadísticas del recorrido con rozamiento viscoso

Magnitud	Valor
Tiempo total	19,56 s
Velocidad máxima	42,08 m/s (151,50 km/h)
Velocidad media	22,74 m/s (81,86 km/h)
Aceleración máxima	289,23 m/s^2 (29,48 g)
Fuerza normal máxima	298,44 N/kg (30,42 g)
Energía inicial	982,96 J/kg
Energía final	473,82 J/kg
Pérdida de energía	51,80 %

1. Perdida de velocidad después de cada loop o colina.
2. Pérdida de energía a lo largo del trayecto.
3. Mayor pérdida de energía para el caso de ambos rozamientos.

3.1.4. Consideraciones de seguridad

Según la norma UNE-EN 13814:2006, la aceleración total experimentada por los pasajeros no debe superar valores del orden de $4g$ durante intervalos prolongados de tiempo. Aceleraciones superiores solo son admisibles de forma puntual y durante tiempos muy breves.

En los resultados obtenidos, se observan valores de aceleración normal y total que alcanzan decenas de veces la aceleración de la gravedad, superando ampliamente los límites admisibles. Esto implica que el diseño inicial de la trayectoria no cumple los criterios de seguridad y debe ser modificado para reducir la curvatura y suavizar los cambios geométricos.

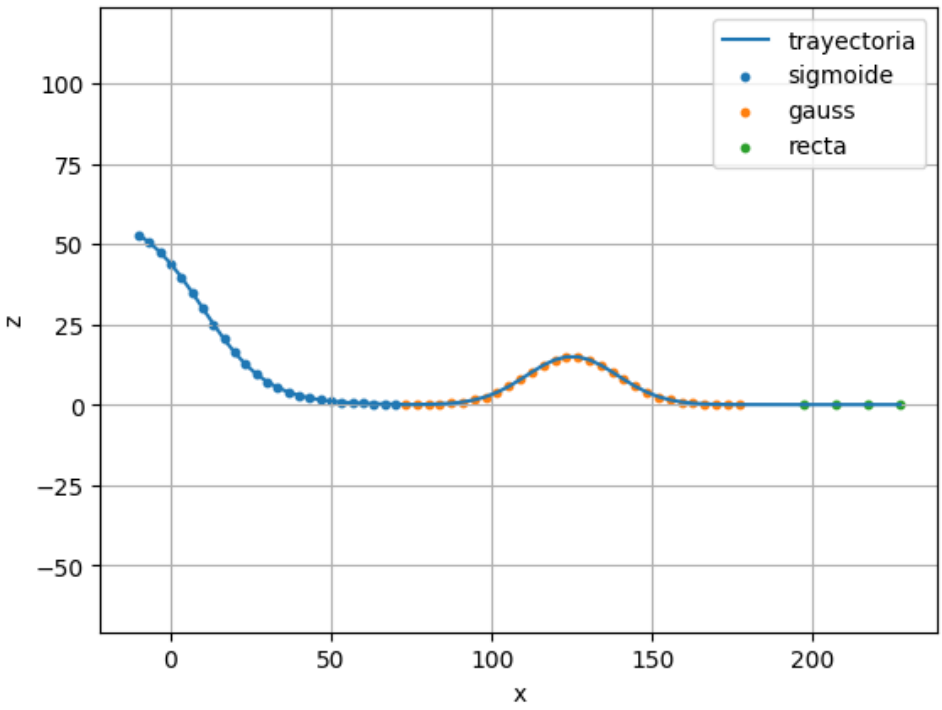

Figura 7: Nueva geometría segura

Cuadro 5: Resultados dinámicos del recorrido con consideraciones de seguridad

Magnitud	Valor
Longitud total del recorrido	265,92 m
Tiempo total	10,58 s
Velocidad máxima	32,26 m/s (116,15 km/h)
Velocidad media	25,12 m/s (90,42 km/h)
Aceleración máxima	55,13 m/s^2 (5,62 g)
Fuerza normal máxima	45,37 N/kg (4,62 g)
Energía inicial	520,44 J/kg
Energía final	520,44 J/kg

Vemos que con esta nueva geometría se obtienen unas acelaraciones máximas de unos 5g que duran menos de 1 segundo si se observan las gráficas.

3.2. Trayectoria 3D

El modelo tridimensional incorpora giros horizontales y tramos no planos. El recorrido completo se construye concatenando segmentos geométricos suaves.

3.2.1. Parametrización

La geometría ya parametrizada sería la siguiente:

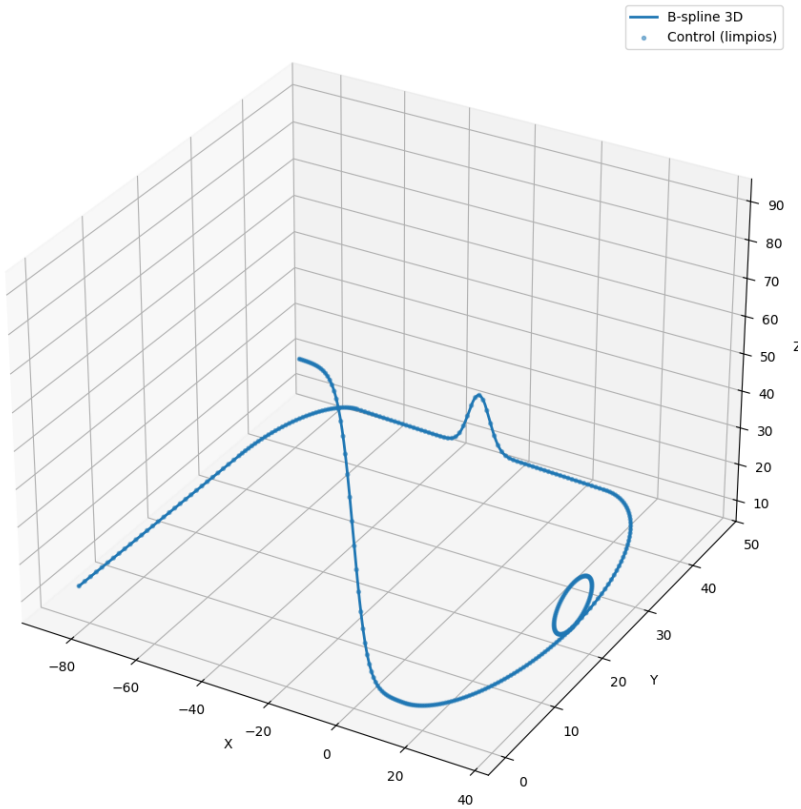

Figura 8: Parametrización del recorrido en 3D

Es necesario mencionar que todas las geometrías se han generado utilizando el script *pkgcurvas.py* menos la clotoide que no funcionaba bien en 3D y se ha parametriazado con una elipse ajustada a mano.

Esta geometría es realmete parecida al caso 2D, entonces, sería normal sacar resultados parecidos en el análisis dinámico.

3.2.2. Comparación de métodos

Se procede de la misma manera que para el caso 2D y se obtienen los siguientes resultados:

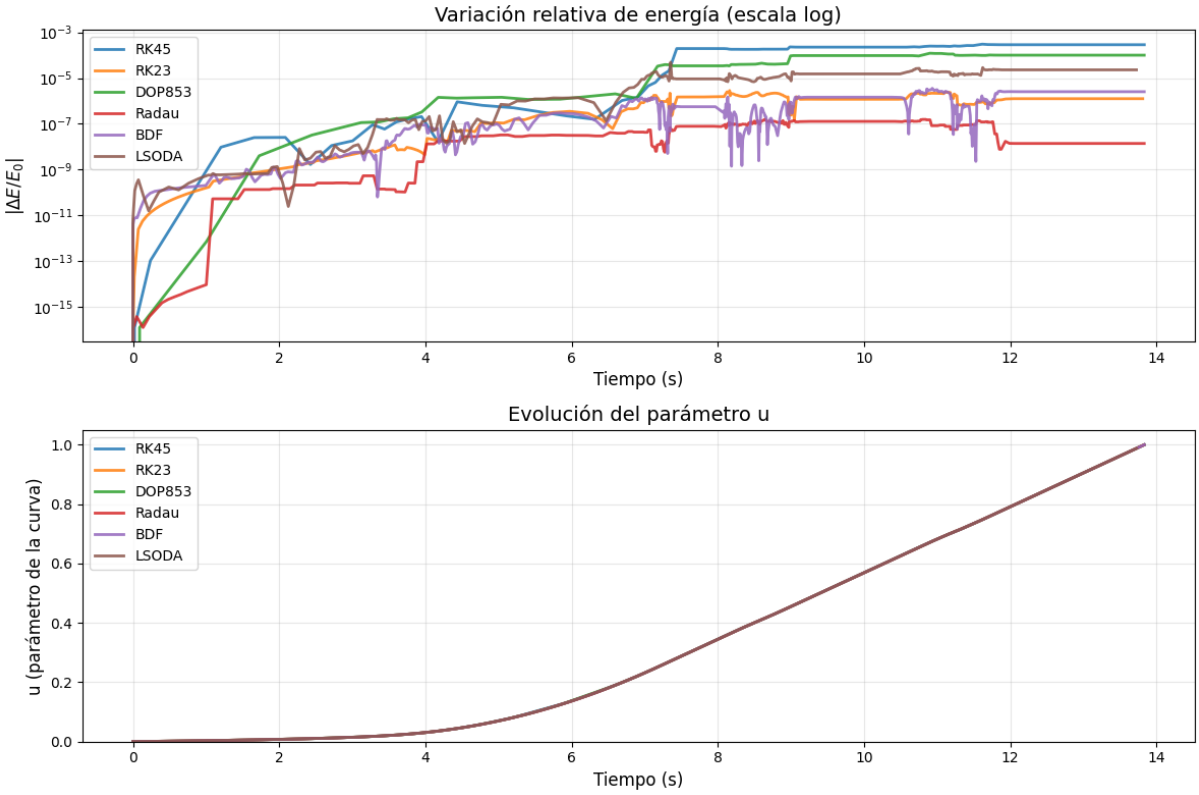

Figura 9: Comparación de distintos métodos de resolución

Cuadro 6: Comparación de métodos numéricos (sistema conservativo 3D). Curva: B-spline 3D interpolado de la trayectoria de montaña rusa. Tolerancias: rtol $= 10^{-6}$, atol $= 10^{-9}$.

Método	Status	Pasos	t_{final}	u_{final}	$\Delta E_{\text{mín}}$	$\Delta E_{\text{máx}}$	ΔE_{final}
RK45	OK	72	13.833	1.00000	$-3{,}0261 \times 10^{-4}$	$+2{,}0693 \times 10^{-5}$	$-2{,}8584 \times 10^{-4}$
RK23	OK	327	13.812	0.99766	$-2{,}8501 \times 10^{-6}$	$+2{,}3953 \times 10^{-6}$	$+1{,}2585 \times 10^{-6}$
DOP853	OK	71	13.832	1.00000	$-4{,}0467 \times 10^{-9}$	$+1{,}2154 \times 10^{-4}$	$+9{,}9940 \times 10^{-5}$
Radau	OK	419	13.832	1.00000	$-1{,}2876 \times 10^{-16}$	$+1{,}5650 \times 10^{-7}$	$+1{,}3966 \times 10^{-8}$
BDF	OK	703	13.832	1.00000	$-3{,}5576 \times 10^{-6}$	$+1{,}5580 \times 10^{-6}$	$-2{,}5417 \times 10^{-6}$
LSODA	OK	406	13.726	0.98787	$-4{,}9378 \times 10^{-5}$	$+1{,}3878 \times 10^{-6}$	$-2{,}2878 \times 10^{-5}$

Al igual que en el caso 2D el método más adecuado para el análisis dinámico es el Radau.

3.2.3. Análisis dinámico

Al igual que con el modelo 2D se han realizado varios análisis con distintos modelos de rozamiento.

Se pasa a exponer directamente los resultados del caso no conservativo con rozamiento seco y viscoso.

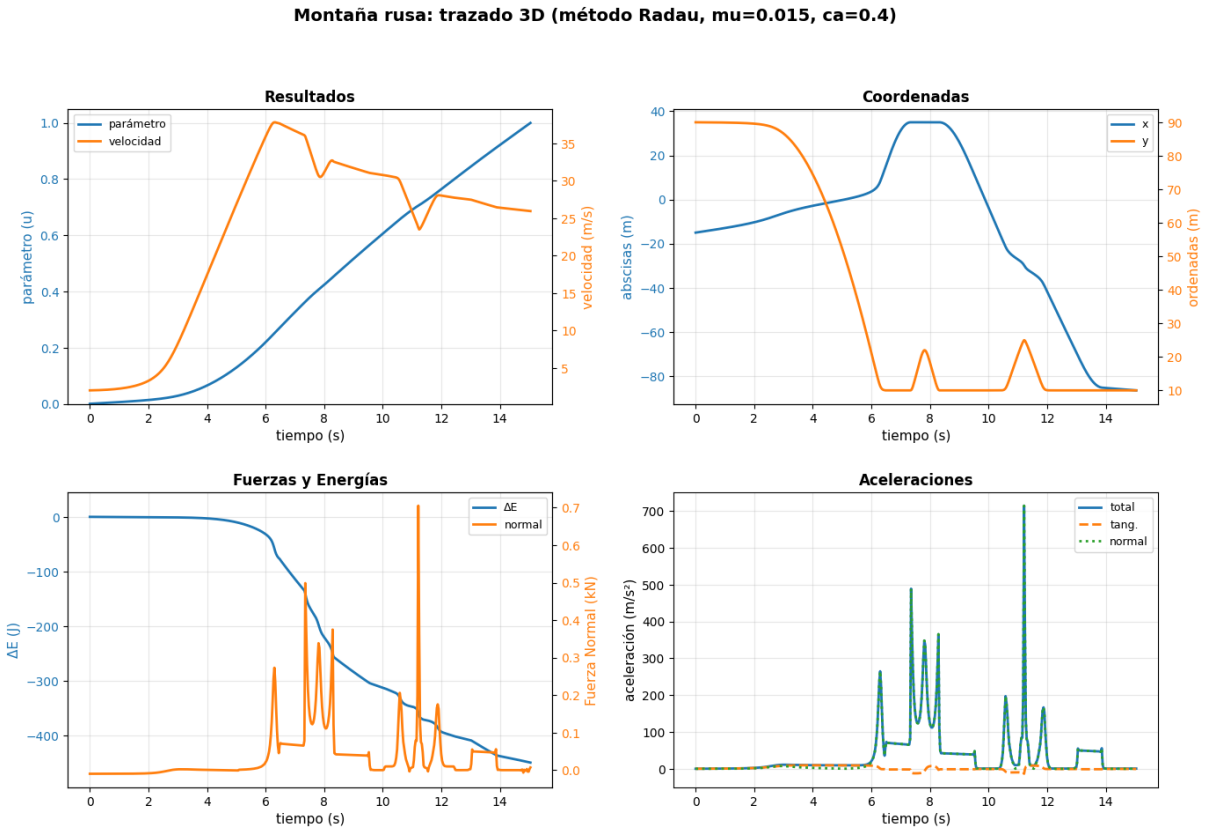

Figura 10: Resultados análisis 3D

Los resultados son casi idénticos al caso 2D y no son para nada sorprendentes.

4. Código

En este capítulo se comentarán las partes más importatntes del código.

El código está hecho en lenguaje python compilado en una libreta jupyter para tener claridad en el proceso.

Los principales módulos utilizados son *numpy.py* para funciones y operaciones matemáticas, *scipy.py* para interpolaradores y métodos de resolución de SEDOs y *matplotlib.py* para los gráficos.

Para el B-spline se ha creado una función a mano para 2D y otra para 3D utilizando *splerp* y *Bspline* de *scipy.interpolate*.

Listing 1: Function p-spline en 2D

```
def spline_xz(puntos_xz, s=0.0, k=3):
    """
    puntos_xz: array-like de shape (N,2) con columnas (x,z)
    Devuelve:
      curva: BSpline 3D (x,y,z) en u$\in [0,1]$, con y=0 siempre
      u: parametros asociados a los puntos (los que usa splprep)
    """
    p = np.asarray(puntos_xz, dtype=float)
    x, z = p[:, 0], p[:, 1]
    y = np.zeros_like(x)
```

```
    tck, u = splprep([x, y, z], s=s, k=k)
    t, c_list, k_out = tck
    c = np.vstack(c_list).T
    curva = BSpline(t, c, k_out, extrapolate=False)

    return curva, u
```

Listing 2: Function p-spline en 3D

```
def spline_xyz(puntos_xyz, s=0.0, k=3):
    """
    puntos_xyz: array-like de shape (N,3) con columnas (x,y,z)
    Devuelve: (curva, u) igual que spline_xz pero en 3D real
    """
    P = np.asarray(puntos_xyz, dtype=float)
    x, y, z = P[:, 0], P[:, 1], P[:, 2]

    tck, u = splprep([x, y, z], s=s, k=k)
    t, c_list, k_out = tck
    c = np.vstack(c_list).T
    curva = BSpline(t, c, k_out, extrapolate=False)
    return curva, u
```

El resto del código es realtivamente sencillo y directo, para la geometría se generan una serie de puntos y luego se los asocia a una curva y luego se juntan todos para obetenr la geometría.

Listing 3: Generación de geometría en 3D

```
x0, z0, A, m = 20, 0, 100, 0.5
u1 = np.linspace(-20, 20, 20)
seg1 = CV.curva3d("s", u1, A=A, C=[x0, 0, z0], paso=0, plano="xz", args
    =[m])

# ---------- 2) Loop (clotoide sim trica) ----------
# en tu script se usa tipo "l" para hacer el looping sim trico
    autom ticamente
x0, z0, A = 50, 0, 35
u2 = np.linspace(0, np.sqrt(np.pi), 10)
seg2 = CV.curva3d("l", u2, A=A, C=[x0, 0, z0], paso=0, plano="xz", args
    =[0])

# ---------- 3) Colina (gaussiana) ----------
# x(u) = x0 + u ; z(u) = z0 + A exp(-(u/s)^2)
x0, z0, A, s = 159, 0, 30, 20
u3 = np.linspace(-52, 52, 15)
seg3 = CV.curva3d("g", u3, A=A, C=[x0, 0, z0], paso=0, plano="xz", args
    =[s])

# ---------- 4) Recta final ----------
# aqu  la hago manual (x lineal, z constante)
x0, z0 = 231, 0
u4 = np.linspace(0, 30, 4)
x4 = x0 + u4
z4 = z0 + 0*u4
y4 = 0*u4
seg4 = np.vstack([x4, y4, z4])

# ---------- concatenaci n ----------
tray = np.hstack([seg1, seg2, seg3, seg4])  # shape (3, N)
```

```
x, y, z = tray[0], tray[1], tray[2]
```

Es importante comentar que para que se pudieran realizar la resolución por los métodos *Radau* y *BSF*, se ha tenido que hacer un wrapper para la función *edofun_mr* para que no diese error por culpa de introducir en el método NaNs o infs.

Listing 4: Función wrapper para edofun_mr

```
def edofun_mr_wrapper(t, y, posyder, coefRoz, coefVis, grav):
"""
Wrapper seguro que evita evaluaciones fuera del dominio [0,1]
"""
u, v = y
# CR TICO: Proyectar u al dominio v lido
u = np.clip(u, 0.0, 1.0)
try:
    fuerzaN, baseLocal, ctes = fuerzaNormal(u, v, posyder, grav)
    # Evitar divisi n por cero
    if abs(ctes[0]) < 1e-12:
        return [0.0, 0.0]
    du = v / ctes[0]
    dv = aceleratg(v, baseLocal, coefRoz, coefVis, fuerzaN, grav)
    # Verificar que no hay NaN o Inf
    if not (np.isfinite(du) and np.isfinite(dv)):
        return [0.0, 0.0]
    return [du, dv]
except:
    return [0.0, 0.0]
```