# Aterrizaje vertical de cohetes espaciales

Código del grupo: CMM916132

1 de Octubre de 2023

## Índice

# 1. Descenso con el mínimo consumo de combustible.

Para solucionar el primer apartado del problema, hemos utilizado Matlab. Pero antes de entrar en lo que hemos programado, hay varios pasos que hemos seguido para poder llegar a los resultados deseados.

## 1.1. Planteamiento del problema.

Empezaremos analizando el movimiento de la nave. La dinámica de la nave viene dada por la ecuación $m \cdot \ddot{p} = f - m \cdot g \cdot e_3$ teniendo $p(t) = (x(t), y(t), z(t))$. El tiempo del aterrizaje está dividido en $K$ periodos de tiempo de tamaño $\Delta t$.

Como la función $f$ es constante en $K$ periodos de tiempo, podemos integrar la ecuación para conseguir la velocidad, $\dot{p}$, y la posición, $p$ en cada momento. Dichas ecuaciones son las siguientes (nos las da el enunciado):

$$v_{i+1} = v_i + (\Delta t/m) \cdot f_i - \Delta t \cdot g \cdot e_3 \tag{1}$$

$$p_{i+1} = p_i + (\Delta t/2) \cdot (v_i + v_{i+1}) \tag{2}$$

Por otro lado, tenemos las siguientes restricciones:

$$\|f(t)\|_2 \leq F^{max} \tag{3}$$

$$z(t) \geq \alpha \cdot \|(x(t), y(t))\|_2 \tag{4}$$

Además, conocemos los valores $p(0) = p_0$ y $\dot{p}(0) = v_0$ y sabemos que la posición final y la velocidad final han de ser 0, es decir, $p(K\Delta t) = 0$ y $\dot{p}(K\Delta t) = 0$. Sabiendo todo esto, nuestro objetivo es minimizar la siguiente función:

$$\int_0^{T^{at}} \gamma \|f(t)\|_2 dt$$

Donde $T^{at} = K\Delta t$, es decir, el tiempo en el que la nave aterriza.

Al desarrollar el problema observamos que la integral a minimizar se convierte en un sumatorio finito, ya que, la función $f_i$ es constante en $t \in [i\Delta t, (i+1)\Delta t]$ para todo $i = 0, 1, ..., K-1$. Es decir, podemos hacer el siguiente desarrollo para conseguir el sumatorio:

$$\int_0^{T^{at}} \gamma \|f(t)\|_2 dt = \sum_{i=0}^{K-1} \int_{i\Delta t}^{(i+1)\Delta t} \gamma \|f_i\|_2 dt = \gamma \Delta t \sum_{i=0}^{K-1} \|f_i\|_2 \tag{5}$$

Teniendo en cuenta todo lo anterior, al discretizar el tiempo tenemos lo siguiente: para cada $t = i\Delta t$, $i = 0, 1, ..., K$ tenemos un valor para $p$ y otro valor para $v$, es decir, en total tenemos $K+1$ valores para $v$ y $p$. Por otro lado, como tenemos un valor de $f$ para cada intervalo de tiempo y $K$ intervalos de tiempo, es decir, en total tenemos $K$ valores diferentes para $f$. Además, $f$, $v$ y $p$ se dividen en coordenadas $x$, $y$ y $z$. Por lo tanto, tenemos que encontrar $2 \cdot (K+1) \cdot 3 + K \cdot 3$ variables tal que minimicen la expresión (5) y que cumplan las condiciones que hemos descrito anteriormente.

Denotamos para todo $i = 0, 1, \ldots, K-1$ que $p_i = (x_i, y_i, z_i)$. Entonces, concluimos que tenemos que resolver el siguiente problema de optimización:

$$
\begin{array}{lll}
\min\limits_{f_i} & \gamma\Delta t \sum_{i=0}^{K-1} \|f_i\|_2 & \\
\text{s.a} & \|f_i\|_2 \leq F^{max} & i = 0, 1, \ldots, K-1 \\
& z_i \geq \alpha \cdot \|(x_i, y_i)\|_2 & i = 0, 1, \ldots, K \\
& p(K\Delta t) = p_K = 0 & \\
& \dot{p}(K\Delta t) = v_K = 0 & \\
& p(0) = p_0 & \\
& \dot{p}(0) = v_0 & \\
& v_{i+1} = v_i + (\Delta t/m) \cdot f_i - \Delta t \cdot g \cdot e_3 & i = 0, 1, \ldots, K-1 \\
& p_{i+1} = p_i + (\Delta t/2) \cdot (v_i + v_{i+1}) & i = 0, 1, \ldots, K-1
\end{array}
\tag{6}
$$

## 1.2. Explicación del código paso a paso

Implementaremos el problema de optimización (6) en Matlab. Para ello, haremos uso del software "Optimization Toolbox". Vamos a explicar qué hemos programado en cada punto, recomendamos tener abiertos los programas que hemos facilitado.

En la carpeta del apartado "a)", hay tres programas de Matlab y una carpeta con figuras (son las soluciones gráficas). Los programas son "Algoritmo", "uniform_descent" e "Implementacion_del_problema". Vamos a explicar cada uno paso por paso:

### 1.2.1. Algoritmo.m

Éste es el algoritmo que resuelve el problema (6). Utiliza las funciones "optimproblem", "optimvar", "solve" y alguna más de "Optimization Toolbox" para hacerlo.

Líneas 6-14. Empezamos creando un problema de optimización y definiendo las variables a optimizar: P, V, F y Fnorm. Las primeras dos son matrizes de tamaño $3 \times (K+1)$ que representan los vectores de posición y velocidad respectivamente. La segunda es una matriz de tamaño $3 \times K$ que representa los vectores de fuerza. La última variable es una variable de tamaño $1 \times K$ extra que hemos introducido para escribir la función a minimizar de forma simple (para ello hemos introducido una restricción extra (7) que explicaremos más adelante). Con todo ello podemos definir la función a minimizar con la sumatoria de Fnorm.

Líneas 15-80. Ahora, una vez definida la función a minimizar, introducimos las restricciones que tienen. Lo único que hay que comentar sobre éstas líneas de código es que hemos introducido una nueva restriccion: que Fnorm tiene que ser la raíz de la suma de los cuadrados de F. Es decir, tiene que ser la norma de F en cada iteración. Esto se describe de la siguiente manera:

$$
Fnorm(i) = \sqrt{F(1,i)^2 + F(2,i)^2 + F(3,i)^2} \tag{7}
$$

Líneas 81-97. Para iniciar nuestro algoritmo, necesitamos unos valores iniciales para las variables. Hemos pensado que para la posición y velocidad, los valores iniciales idoneos sean los siguientes: calcular la diferencia entre el valor inicial y el final (0) y crear un vector de tamaño K+1 tal que la distancia entre las posiciones $i$ y $i+1$ sean constantes para cualquier $i = 0, 1, ..., K$. Para ello hemos creado la función auxiliar "uniform descent". Por otro lado, hemos fijado los valores iniciales de F todos a unos y al de Fnorm a todo $\sqrt{3}$.

Línea 99. Resuelve el problema usando "solve" de "Optimization Toolbox".

### 1.2.2. uniform_descent.m

Primero calcula la distancia constante que va a haber entre $v(i)$ y $v(i+1)$ para cualquier $i = 1, 2, ..., K$. Luego crea un vector de unos de tamaño $K+1$ y lo va llenando con la fórmula $v(i) = x_0 + d \cdot (i-1)$ para $i = 1, 2, ..., K$. El valor de $v(K+1)$ se queda 1 ya que si no hacemos esto e introducimos un valor nulo, el algoritmo de optimización devolvería un error.

### 1.2.3. Implementacion_del_problema.m

Líneas 1-14. Introduce los valores iniciales para el caso propuesto.

Línea 17. Resuelve el problema con "Algoritmo.m" y consigue la solución y la cantidad de combustible (mínima/óptima).

Líneas 22-42. Dibuja la gráfica de la solución del problema.

## 1.3. Resultados finales

La solución a este apartado está resumida en esta gráfica generada en Matlab.

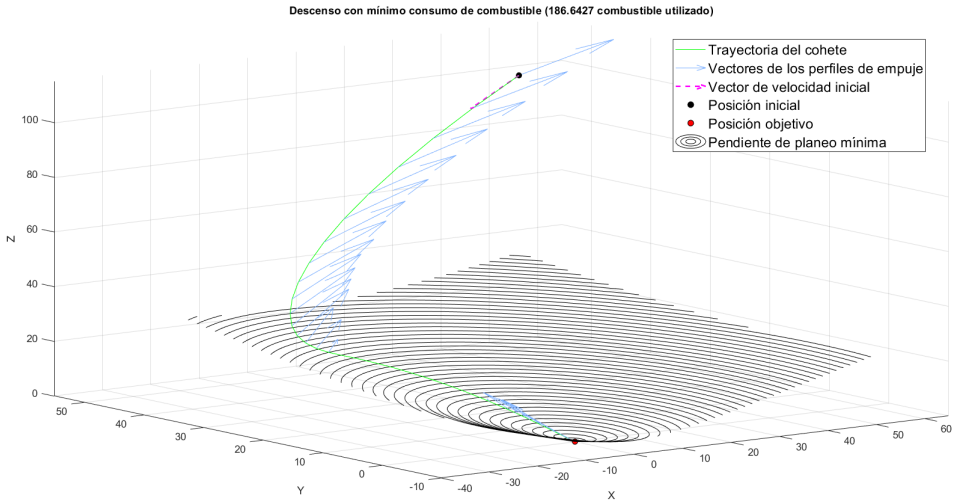


Figura 1: Trayectoria del cohete con las respectivas fuerzas de empuje y el total de combustible utilizado.

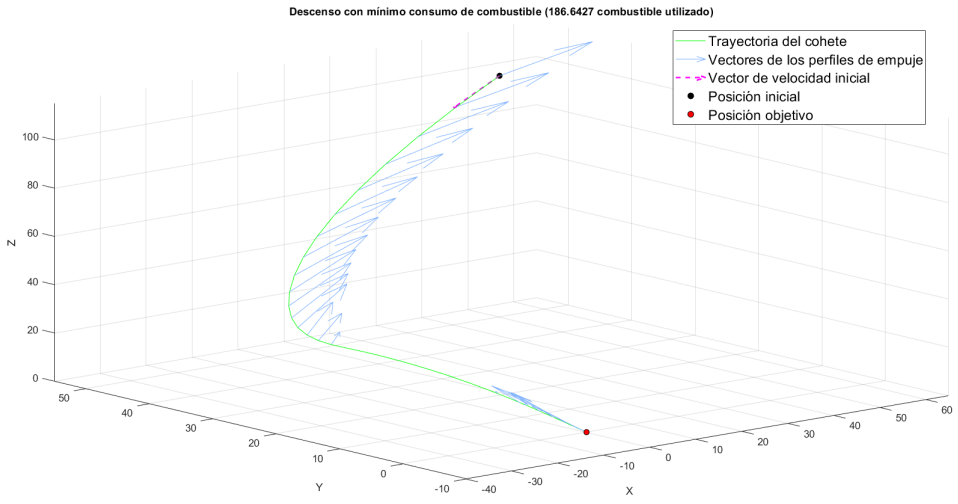


Figura 2: Misma gráfica del recorrido pero sin el plano de planeo mínimo, para poder observar el recorrido y los vectores de los perfiles de empuje con mayor claridad.

Obtenemos que el cohete utilizará un total de $186,6475$ de combustible en su recorrido desde la posición inicial hasta la posición final. Manteniéndose siempre encima del plano de planeo mínimo y aterrizando en el punto objetivo. Para ver más en detalle los resultados, en los archivos adjuntados, en el apartado a), en la carpeta de "figuras", están ambas gráficas.

# 2. Tiempo mínimo de descenso.

Por suerte para este segundo apartado la gran mayoría del trabajo ya lo tenemos hecho al tener programado el problema de optimización. La única diferencia es que la variable $K$, el tiempo de aterrizaje, es desconocida: tenemos que encontrar una $K$ mínima para la cual el problema siga cumpliendo sus restricciones.

Como el algoritmo que hemos diseñado no te advierte si la solución que ha devuelto no cumple las restricciones, hemos creado una función auxiliar para detectar si las soluciones son realmente soluciones o no. Es decir, si cumplen las restricciones o no.

## 2.1. Explicación del código paso a paso

En la carpeta del apartado "b)", hay cinco programas de Matlab y una carpeta con figuras (son las soluciones gráficas). Los programas son por una parte "Algoritmo" y "uniform_descent", que ya han sido explicadas anteriormente, y por otra parte "verificar_restricciones", "K_minimo" y "Graficar_el_K_minimo". Vamos a explicar cada uno paso por paso:

### 2.1.1. verificar_restricciones.m

Este es el código que detecta si una solución es correcta o no. Es decir, verifica que las restricciones que tiene nuestro problema de optimización se cumplen. Para ello, las restricciones (3), (4), (7), $p(0) = p_0$, $\dot{p}(0) = v_0$, $p_K = 0$ y $v_K = 0$ se verifican directamente en las soluciones que devuelve el algoritmo.

Sin embargo, para verificar (1) y (2) hemos hecho lo siguiente: a partir de los valores de $f_1, f_2, ..., f_K$ que nos devuelve el algoritmo, utilizando las fórmulas (1) y (2) hemos generado unos matrices $Pnew$ y $Vnew$ con los valores reales que deberían tener las matrices $P$ y $V$ de las soluciones. Luego, hemos comparado ambos pares de matrices.

A la hora de comparar matrices, hemos calculado la diferencia de ambos (que idealmente sería 0). Al hacer esto, tenemos un problema: los ordenadores trabajan en punto flotante y pueden haber errores. Para solucionarlo, hemos introducido un valor de tolerancia de $10^{-3}$.[1]

Líneas 5-7. Como el único objetivo de esta función auxiliar es decirnos si la solución que ha devuelto el Algoritmo cumple las restricciones o no, creamos la variable bool que incialmente será True, a no ser que alguna de las restricciones no se cumpla, en cuyo caso se convertirá en False. Por otra parte creamos las nuevas matrices $Pnew$ y $Vnew$ para poder luego compararlos como hemos dicho anteriormente.

Líneas 9-18. Empezamos comprobando si de verdad nuestro cohete aterriza en la posición objetivo y con velocidad cero, calculando la diferencia entre nuestra posición y velocidad final respecto a la posición y velocidad objetivo. Como hemos dicho antes, asumiremos un margen de error máximo de $10^{-3}$ por si el algoritmo redondea al hacer los cálculos.

Líneas 19-48. Ahora comprobaremos uno por uno si para cada $i = 0, 1, ..., K$ se cumplen las restricciones de $\|f_i\|_2 \leq F^{max}$, $z_i \geq \alpha \cdot \|(x_i, y_i)\|_2$ y comprobaremos la restricción extra (7). Aparte de eso, calculamos las nuevas matrices $Pnew$ y $Vnew$ para los distintos valores de $K$ para luego

[1] En las notas de pie de página de la hoja 6 hemos explicado por qué hemos decidido usar una tolerancia de $10^{-3}$

poder compararlos con los introducidos en el problema.

Líneas 52-66. Comparamos los valores de las nuevas matrices $Pnew$ y $Vnew$ con los introducidos en el programa. Si la distancia entre alguno de los valores es superior a la tolerancia establecida, las restricciones no se cumplen y la varibale bool se convierte en false.

Diremos que "el algoritmo falla" si al ejecutarse "verificar_restricciones" nos devuelve False.

### 2.1.2. K_minimo.m

Esta función recive los datos necesarios para ejecutar el programa "Algoritmo" y lo itera mientras reduce la cantidad de $K$ en una unidad hasta que $K$ sea 0, cuando $K = 0$, el algoritmo ("while") deja de ejecutarse. Al principio, se crea un vector $v$ de tamaño $K$ y en cada iteración, si el algoritmo resulta válido, introduce un 1 en el vector, y, en caso contrario, introduce un 0. Es decir, a nosotros nos interesa saber el índice minimo en el que el vector $v$ sea 1. [2] [3]

### 2.1.3. Graficar_el_K_minimo.m

Líneas 17-20. Ejecuta la función del K_minimo para conseguir nuestro $K$ mínimo y resuelve el mismo problema del primer apartado pero con una $K$ menor. En este caso, si ejecutamos K_minimo.m conseguiremos el valor 20. Es decir, el valor minimo de $K$ en el que el problema del apartado a) puede plantearse es 20, por ende, el tiempo de aterrizaje mínimo será de $K\Delta t = 20\Delta t = 20$. Como el código tarda mucho en ejecutarse, hemos dejado comentado el código para ejecutar K_minimo.m y hemos puesto directamente $Kmin = 20$.

Líneas 23-42. Dibuja la gráfica de la solución del problema.

[2] Podríamos pensar que el algoritmo hace demasiados calculos, ya que, en principio, empezando las iteraciones en $K = 35$ y disminuyendolas hasta encontrar un $K$ (por ejemplo $K = 30$) en el que el algoritmo falle debería de ser suficiente. Siguiendo la lógica, para todo $K = 1, 2, ..., 29$ el algoritmo debería de fallar, sin embargo, con tolerancia de $10^{-5}$ el algoritmo falla en $K = 30$ pero no en $K = 20$. Por eso, para saber el $K$ mínimo, hemos calculado todos los $K = 1, 2, ..., 35$.

[3] Del razonamiento anterior viene el hecho de usar una tolerancia de $10^{-3}$, ya que, al usar dicha tolerancia, todos los algoritmos de $K = 20, 21, ..., 35$ no fallan y todos los algoritmos $K = 1, 2, ..., 19$ fallan. Sin embargo, el $K$ mínimo no cambia si usamos una tolerancia de $10^{-5}$, ya que, para $K = 20$ el algoritmo no falla y para todo $K = 1, 2, ..., 19$ el algoritmo falla. Este hecho se debe a que para las iteraciones $1, 2, ..., 19$, el algoritmo devuelve fuerzas con norma mayor a Fmax. Además, si las unidades son metros, una tolerancia de $10^{-3}$ sería una tolerancia de milímetros.

## 2.2. Resultados finales

La solución a este apartado también está resumida en esta gráfica generada en Matlab.

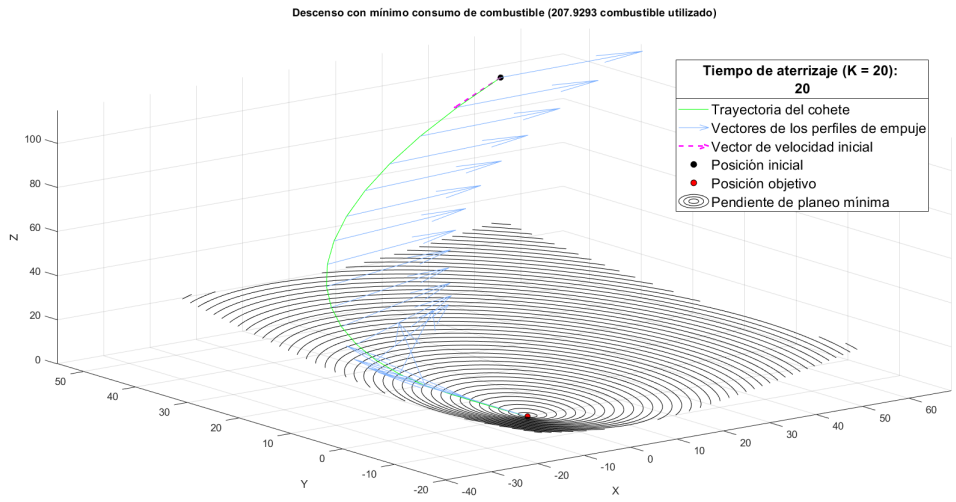


Figura 3: Trayectoria del cohete con las respectivas fuerzas de empuje, el total de combustible utilizado y el tiempo total de aterrizaje.

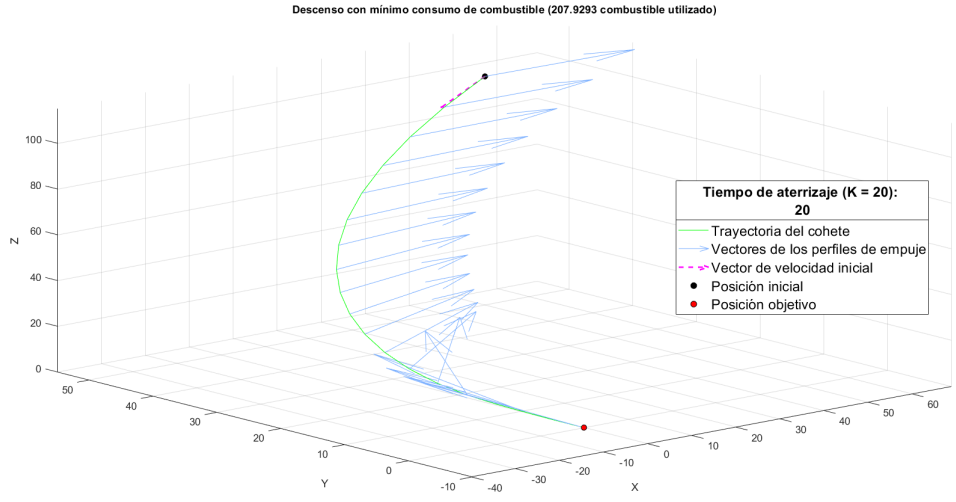


Figura 4: Misma gráfica del recorrido pero sin el plano de planeo mínimo, para poder observar el recorrido y los vectores de los perfiles de empuje con mayor claridad.

Como podemos observar, la trayectoria del cohete se parece al conseguido en el anterior apartado, pero existe una gran diferencia, los vectores de los perfiles de empuje. En la gráfica se aprecia que son claramente mayores que los conseguidos en el primer apartado,como es lógico, ya que para llegar antes al destino el cohete ha de frenar más rápido. Además, observamos que en este proceso usa más combustible que para $K = 35$.

# 3. Gráficos extra de proyecciones de las soluciones

## 3.1. Gráficos extra del apartado a)

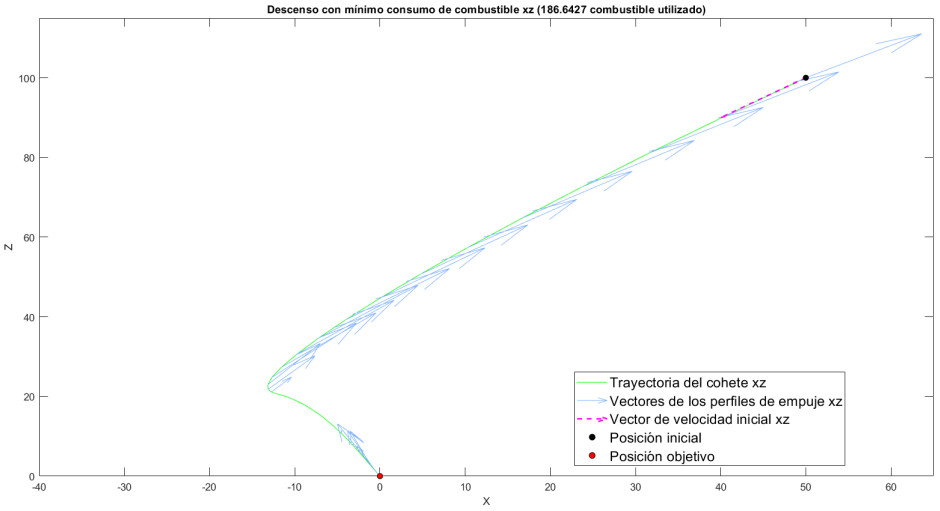


Figura 5: Proyección en el plano xz de la solución del apartado a)

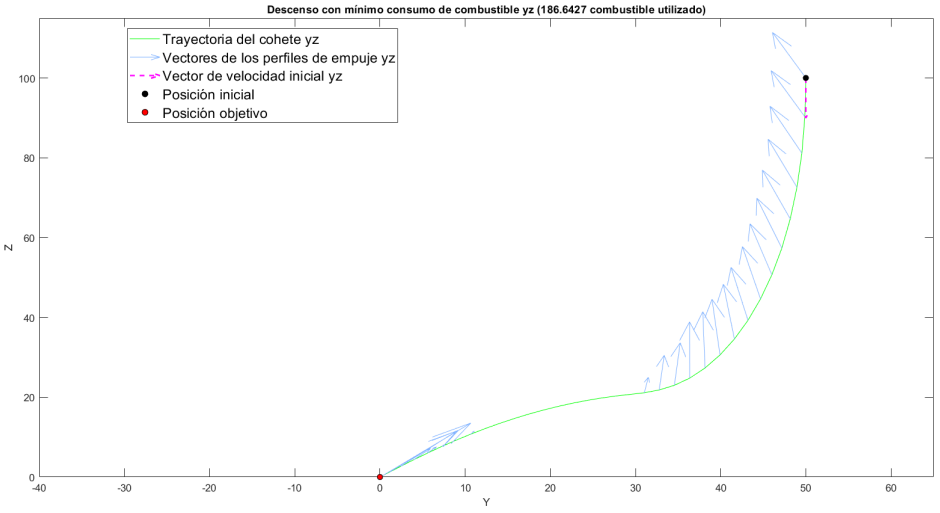


Figura 6: Proyección en el plano yz de la solución del apartado a)

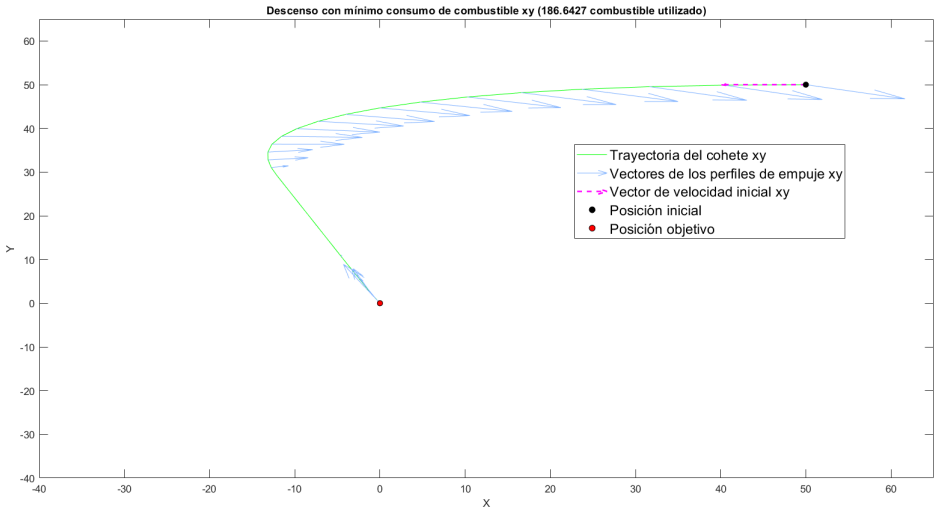


Figura 7: Proyección en el plano xy de la solución del apartado a)

## 3.2. Gráficos extra del apartado b)

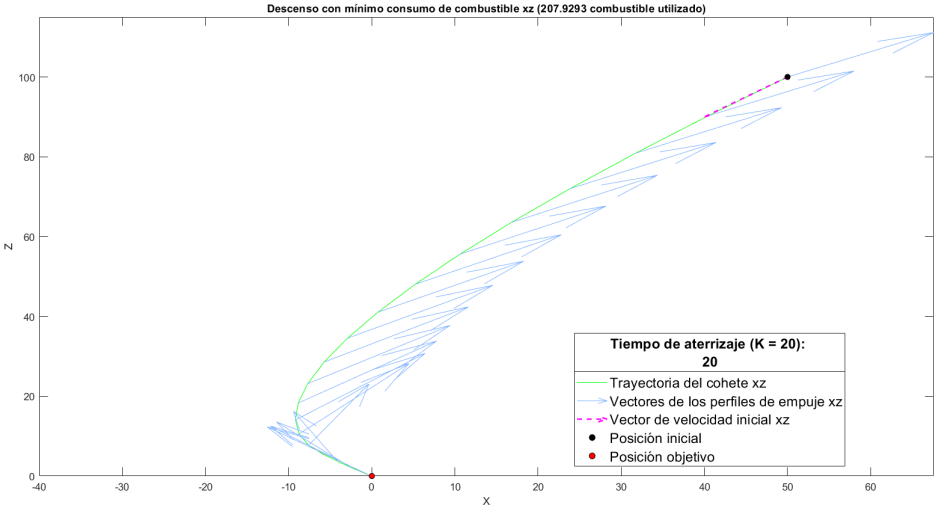


Figura 8: Proyección en el plano xz de la solución del apartado b)

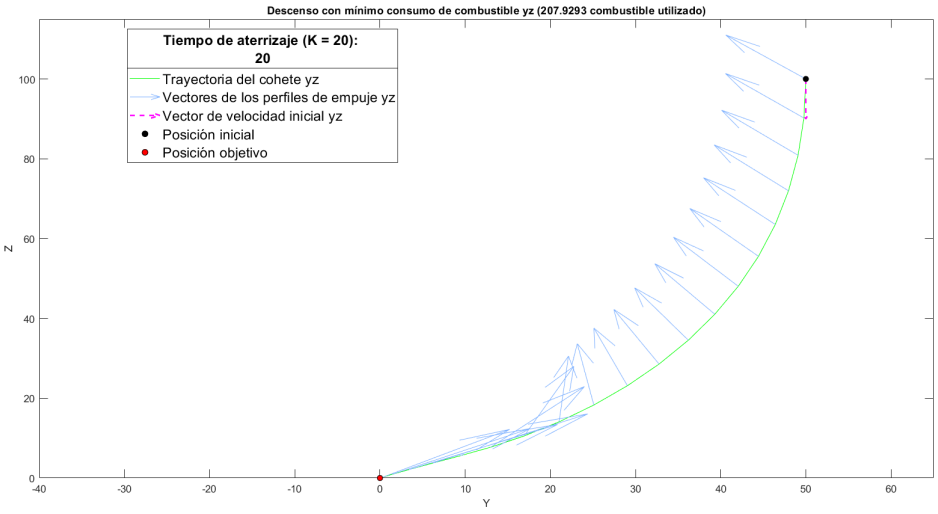


Figura 9: Proyección en el plano yz de la solución del apartado b)

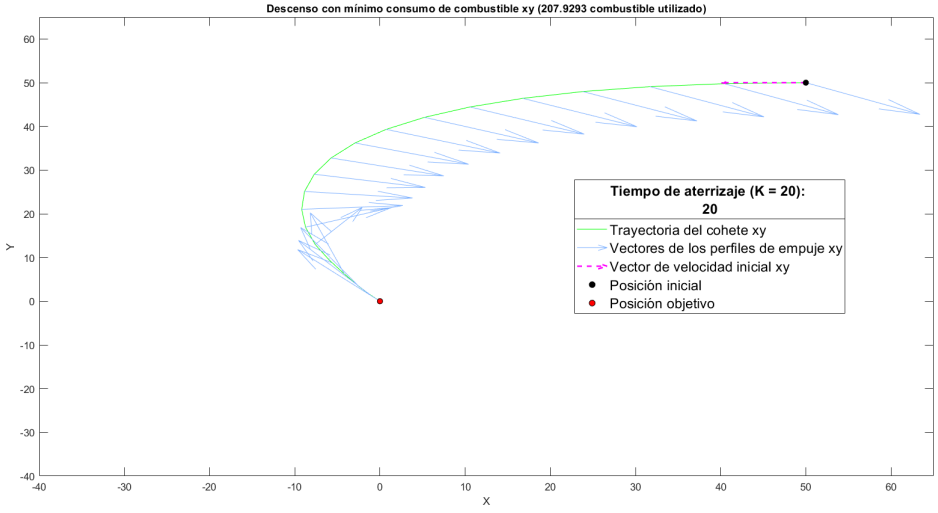


Figura 10: Proyección en el plano xy de la solución del apartado b)