Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Matemática

Laboratorio 3

Problemas de Tiempo Mínimo

Estudiante: Maximiliano S. Lioi
Isidora Reyes Zenteno
Profesor: Héctor Ramírez C
Auxiliar: Matías V. Vera
Ayudante de laboratorio: S. Adrián Arellano

Santiago de Chile

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Contexto

Se considera una partícula en el espacio, la cuál está sometida a las fuerzas dadas por $F_x = x + u$ y $F_y = y + v$, donde u y v son los controles del sistema. Además de lo anterior, la velocidad del eje z depende de la posición en los ejes x e y de la forma $x + y$. El sistema queda entonces descrito como sigue:

$$\ddot{x} = x + u; \quad \ddot{y} = y + v; \quad \dot{z} = x + y.$$

Para el sistema de medida usado, la partícula tiene masa unitaria, y en el instante inicial, se encuentra en reposo en la posición $(1, 1, 0)$. El objetivo de este problema es llevarla al origen en tiempo mínimo. Sin embargo, el módulo de cada control, no puede exceder las 5 unidades, es decir, para cada instante $t, u(t), v(t) \in [-5, 5]$.

2. Ejercicio 1

Se estudia la controlabilidad del sistema (sin restricciones) y la existencia de un control que lleve la partícula al origen en tiempo mínimo.

La dinámica del sistema escrita en forma matricial queda:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ x + u \\ y + v \\ x + y \end{bmatrix}$$

Equivalentemente

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \vec{X} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \vec{U}$$

Procedemos a calcular la matriz de Kalman para ver la controlabilidad del sistema lineal, para ello hacemos uso de la librería *control* de *Python*

```
from control import ctrb
Kalman = ctrb(A,B)

```

De donde se obtiene que la matriz de Kalman K viene dada por

$$K = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Ejecutando

```
rg_Kalman = np.linalg.matrix_rank(Kalman)

```

Se obtiene que el rango de la matriz de Kalman es 5 y por tanto el sistema es controlable.

Por lo demás, la existencia de un control que lleva el sistema al origen en tiempo mínimo se tiene de los siguientes puntos:

- La dinámica es de la forma $\dot{X} = AX + BU$ sin ruido $r(\cdot)$ y entonces sabemos que si el sistema es controlable, equivale a que el sistema es controlable a cero.
- El problema de tiempo mínimo al cero desde x_0

$$(P_{x_0 0}) \quad \min_{u(\cdot)} T; x(T, u(\cdot), x_0) = 0$$

  es factible por el punto anterior.

Por teoremas vistos en clase, si el problema $(P_{x_0 0})$ es factible, entonces existe un control óptimo para el problema, es decir, el problema tiene solución.

3. Ejercicio 2

Mediante la fórmula de discretización de Euler, se discretizan N puntos para la dinámica del problema en el intervalo de tiempo $[0, t_f]$..

Utilizando la formula de discretización de Euler, como $\dot{X} = AX + BU$, para $X(0) = 0$ aproximamos la solución a través de pequeños cambios en el tiempo, puesto que la derivada es la mejor aproximación lineal de la función entorno a un punto.

Se tiene entonces que para una discretización de N puntos, la discretización de la ecuación diferencial viene por:

$$X_{i+1} = X_i + \Delta t(Ax_i + Bu_i) \ \forall i \in \{0, 1, ..., N-2\}$$

donde $\Delta t = \frac{t_f}{N-1}$

Consideramos el siguiente código para resolver dados el control $U(t) = (u(t), v(t))$ como funciones, condición inicial $X(0) = X_0$ y t_f fijo.

Código 1: Método de Euler

```
# Método de Euler
def Euler(X_0, U, t_f):
    # Condiciones iniciales
    delta_t = t_f / (N - 1)
    X = np.zeros((5, N))
    X[:, 0] = X_0
    #Método de Euler
    for i in range(N - 1):
        X[:, i + 1] = X[:, i] + delta_t * (np.dot(A, X[:, i]) + np.dot(B, U(delta_t * i)))
    return X
```

En caso que el control $U = (u, v)$ (que se supone dado) esté discretizado para los N puntos de $[0, t_f]$, consideramos el siguiente código, que será útil para el Ejercicio 6

```
def EulerD(X_0, U, t_f):
    # Condiciones iniciales
    u = U[:N]
    v = U[N:]
    delta_t = t_f / (N - 1)
    X = np.zeros((5, N))
    X[:, 0] = X_0
    #Método de Euler
    for i in range(N - 1):
        X[:, i + 1] = X[:, i] + delta_t * (np.dot(A, X[:, i])  + np.dot(B,(u[i],v[i])))
    return X
```

4. Ejercicio 3

Usando Python, se grafica en 3D el cubo $[-1,1]^3$, de modo que se aprecien las líneas de flujo del sistema, para controles $(u,v) \equiv 0$ y uno constante haciendo uso de *matplotlib.axes.Axes.quiver*.

Para obtener los vectores correspondientes al campo vectorial en el que se ve sujeto la dinámica, discretizamos la región haciendo uso de *np.meshgrid()* y resolvemos en cada punto x_0 de la región la ecuación diferencial asociada, tomando dicho punto como condición inicial $x(0) = x_0$, luego se calcula la diferencia entre dicha condición inicial x_0 y la solución en un tiempo no muy grande $x(\Delta t) - x_0$ para apreciar el desplazamiento que toma la particula pasado un instante de tiempo.

```
# Función que calcula las diferencias entre la solución en dos puntos cercanos normalizada (campo vectorial)
def calculate_vector_field(X, i, j):
    delta_t = t[j] - t[i]
    return (X[:, j] - X[:, i]) / delta_t

# Discretización del cubo [-1, 1]^3
x = np.linspace(-1, 1, 10)
y = np.linspace(-1, 1, 10)
z = np.linspace(-1, 1, 10)
X_grid, Y_grid, Z_grid = np.meshgrid(x, y, z)

# Inicializar vectores de dirección en cada punto de la cuadrícula
Ux = np.zeros(X_grid.shape)
Uy = np.zeros(Y_grid.shape)
Uz = np.zeros(Z_grid.shape)

# Calcular el campo vectorial en la cuadrícula para el control U
for i in range(len(x)):
    for j in range(len(y)):
        for k in range(len(z)):
            X = Euler(np.array([x[i], y[j], 0, 0, z[k]]), U, t_f)
            diff = calculate_vector_field(X, 0, 3)
            Ux[i, j, k] = diff[0]
            Uy[i, j, k] = diff[1]
            Uz[i, j, k] = diff[4]
```

4.1. Caso U(t) = 0

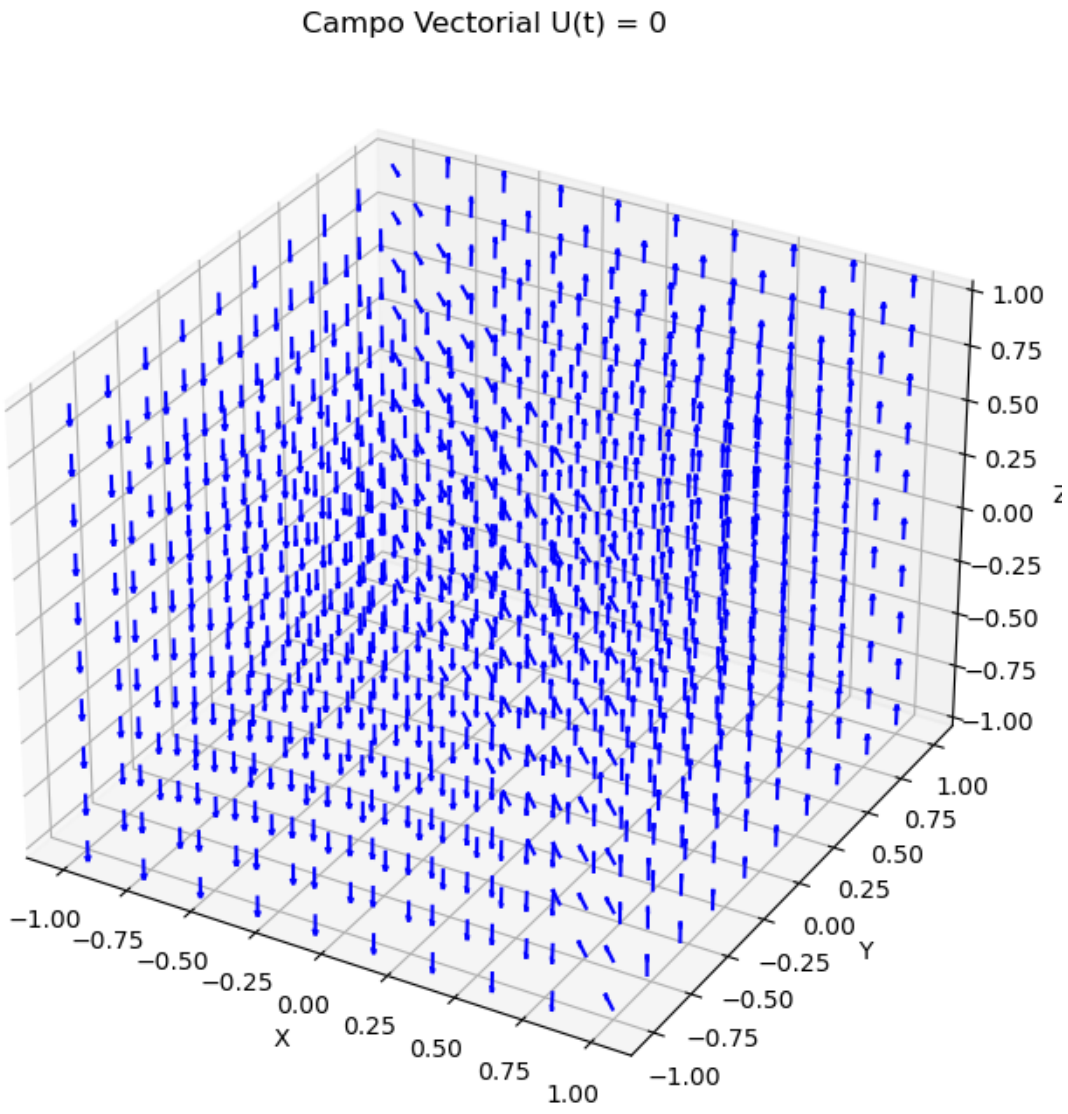


Figura 1: Lineas de flujo del sistema para control U(t) = 0

4.2. Caso U(t) = (4, -10) constante

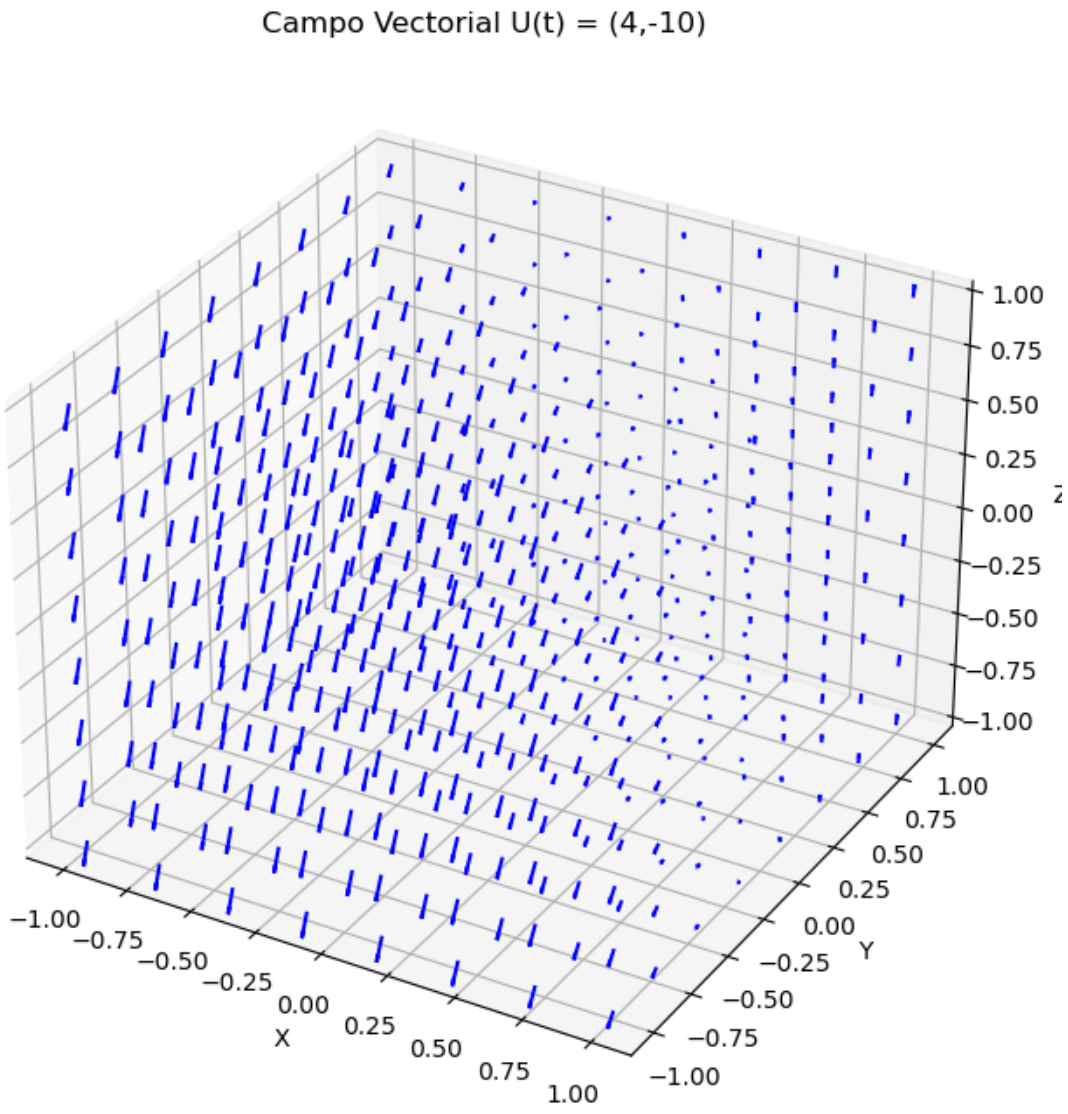


Figura 2: Lineas de flujo del sistema para control U(t) = (4,-10) constante

5. Ejercicio 4

Se grafica la trayectoria de la partícula sometida a ambos campos con condición inicial $(x(0), y(0), z(0)) = (1, 1, 0)$, identificando cuál es el punto de partida y cuál es el de término.

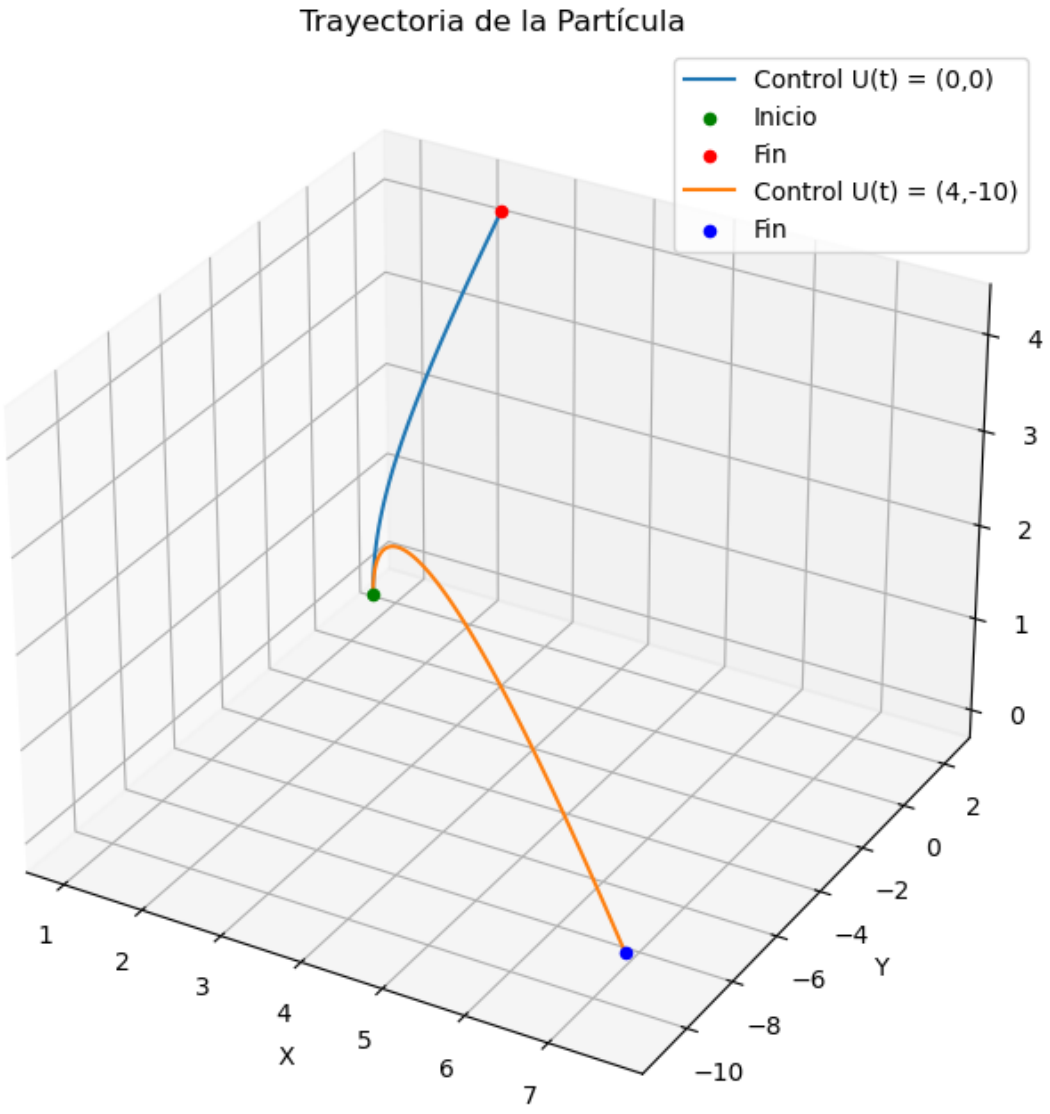


Figura 3: Trayectoria de la partícula sometida a los campos dados por el control U(t)=0 y U(t)=(4,-10)

6. Ejercicio 5

Se utiliza la función descrita en el Ejercicio 2 junto a *scipy.optimize.minimize* para encontrar numéricamente un control que lleva la partícula al origen en un tiempo factible, es decir, para el problema de tiempo mínimo, se busca un par factible (U, t_f) asociado.
Para ello, consideramos el siguiente problema de optimización

$$(P) \ \min_U ||(x(T), y(T), z(T))||_2$$

$$s.a \ \dot{X} = AX + BU \quad \text{en} \quad [0, T]$$

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq U \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix} ; \ x_0 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

para $T > t_f^* > 0$ donde t_f^* es el tiempo óptimo teórico asociado al problema de tiempo mínimo, el cual sabemos existe y es mayor que cero. Sabemos que para el problema (P) cualquier U tal que $||(x_U(T), y_U(T), z_U(T))||_2 = 0$ es solución, dicho esto, se procede a resolver el problema discretizado haciendo uso del método *minimize*.

```
# Discretización del tiempo
N = 100
t_f = 2
t = np.linspace(0, t_f, N)

# Función objetivo a minimizar a tiempo final es la distancia, esperamos un control que haga la distancia al origen cero
def objective_function(U):
    X = EulerD(X_0, U, t_f) # Solución de la EDO
    X_tf = X[:, -1] #Solución en tiempo final
    distance = np.linalg.norm(X_tf[[0, 1, 4]])  # Distancia hacia el origen
    return distance

# Elegimos iteración inicial para minimización con controles constantes u(t) = 0 y v(t) = 1
initial_u = np.zeros(N)
initial_v = np.ones(N)
initial_guess = np.concatenate((initial_u, initial_v))

# Definir límites de restricción para u y v
lower_bound = -5.0
upper_bound = 5.0
linear_constraint = LinearConstraint(np.identity(2 * N), lower_bound, upper_bound)

# Resolver el problema de optimización para encontrar el control que minimiza la distancia
result = minimize(objective_function, initial_guess, method='SLSQP', constraints=[linear_constraint])
optimal_control = result.x
```

6.1. Caso T = 2

Se resuelve el problema de optimización en tiempo $T = 2$, el cual entrega los siguientes controles para el problema de llevar la partícula al cero.

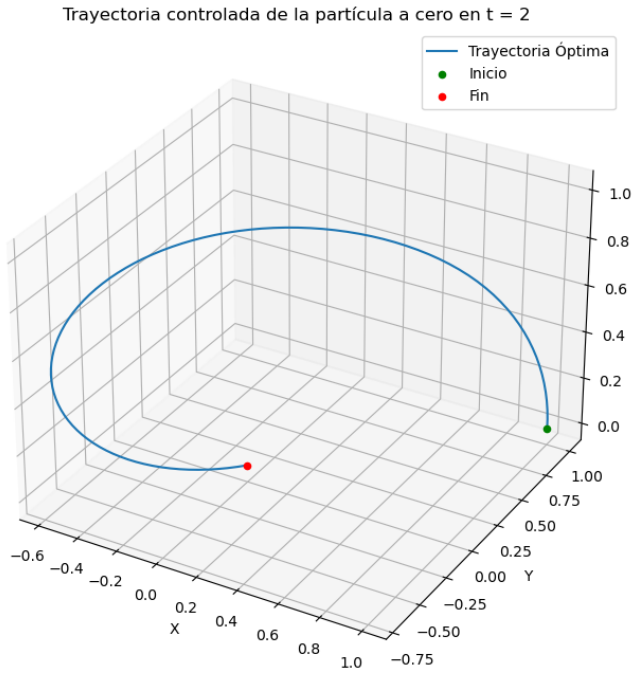


Figura 4: Trayectoria de la partícula para el control óptimo del problema (P) discretizado en [0,2]

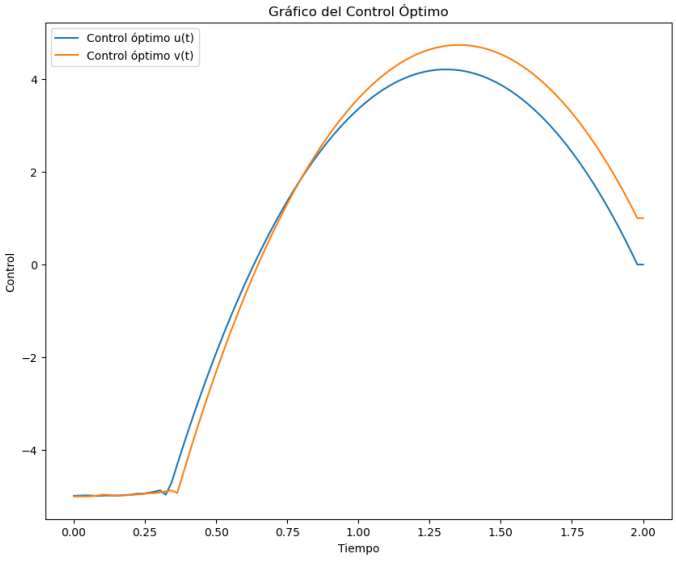


Figura 5: Control óptimo del problema (P) discretizado en [0,2]

6.2. Caso T = 1.5

Se resuelve el problema de optimización en tiempo $T = 1.5$, el cual entrega un óptimo que no lleva la partícula a cero.

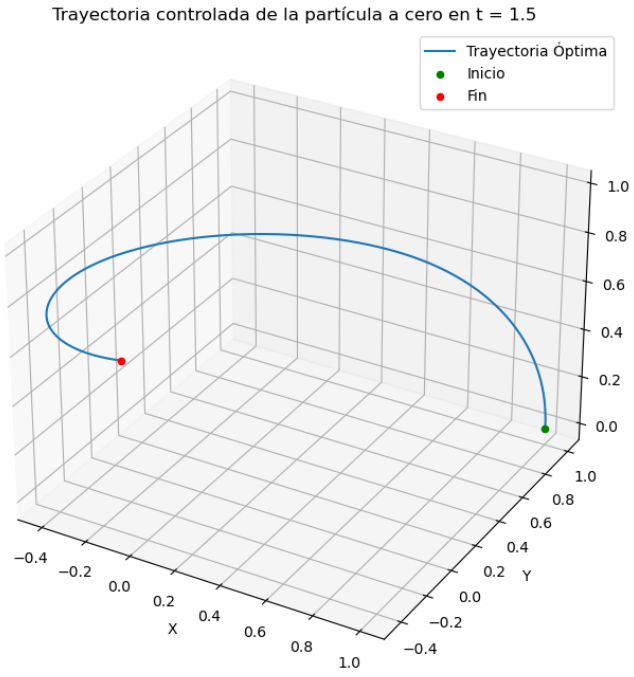


Figura 6: Trayectoria de la partícula para el control óptimo del problema (P) discretizado en [0,1.5]

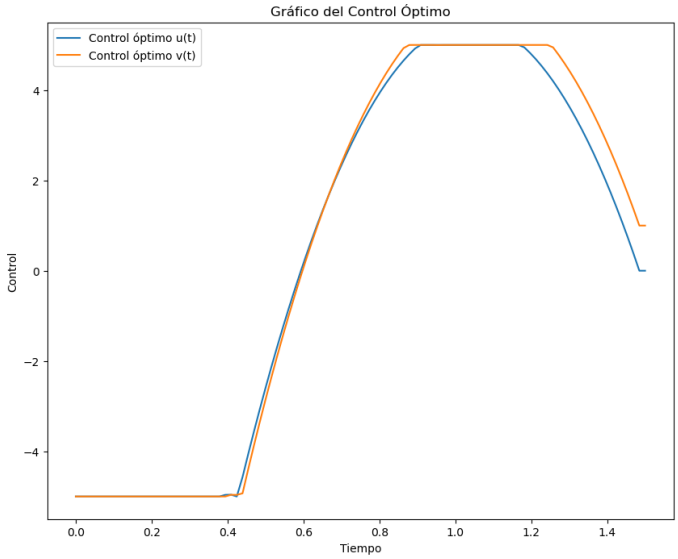


Figura 7: Control óptimo del problema (P) discretizado en [0,1.5]

Para este último caso, se tiene que el estado final de la partícula es

$$[x(2), y(2), z(2)] = \begin{bmatrix} -0.21823545 & -0.21220202 & 0.5201669 \end{bmatrix}$$

Se verifica numéricamente que para $T \leq 1.5$ no se logra encontrar un control que lleve la partícula a cero, esperamos que el control óptimo lleve la partícula en un tiempo óptimo $t_f \in [1.5, 2]$

7. Ejercicio 6

Se resuelve el problema de tiempo mínimo discretizado para varios valores de N y para varias condiciones iniciales u_i, v_i y t_f.

Consideramos ahora el problema de tiempo mínimo a cero

$$(P) \min_{U, t_f} t_f$$

$$s.a\ \dot{X} = AX + BU \quad \text{en} \quad [0, t_f]$$

$$(x(t_f), y(t_f), z(t_f)) = (0, 0, 0)$$

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq U \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix};\ x_0 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Solucionamos mediante el método *minimize* el problema discretizado

$$(P_D) \min_{U, t_f} t_f$$

$$s.a\ \dot{X} = AX + BU \quad \text{en} \quad \mathbb{R}^n$$

$$(x(t_f), y(t_f), z(t_f)) = (0, 0, 0)$$

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq U_i \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix} \forall i \in [N];\ x_0 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Puesto que consideramos el tiempo como dato a optimizar, el nuevo parámetro $U = (U, t_f)$ se encuentra ahora en $\mathbb{R}^{2N+1}$, consideramos la siguiente función

```
# Definimos función que toma un U y X_0 y entrega X(t_f)
def EulerTf(X_0, U):
    U_c = U[:2*N]
    t_f = U[-1]
    X = EulerD(X_0, U_c, t_f) # Solución de la EDO
    X_tf = X[:, -1] #Solución en tiempo final
    return X_tf[[0, 1, 4]]
```

Por lo que la sintáxis del problema discretizado, se traduce en el siguiente código

Código 2: Solución del problema de tiempo mínimo

```
# Función objetivo a minimizar es el tiempo final, esperamos un control que haga la distancia al origen cero en tiempo mínimo

def objective_function(U):
    t_f = U[-1]
    return t_f

# Elegimos iteración inicial para minimización con controles constantes u(t) = 0 y v(t) = 1 y t_f = 1
initial_u = -5.0*np.ones(N) # u(t) = -5, va variando
initial_v = -5.0*np.ones(N) #v(t) = -5, va variando
initial_tf = np.array([0.8]) #t_f va variando
initial_guess = np.concatenate((initial_u, initial_v, initial_tf))

# Definir límites de restricción para u , v en [-5,5] y t_f en [0, +inf]
lower_bound = -5.0*np.ones(2*N +1)
lower_bound[-1] = 0.001
upper_bound = 5.0*np.ones(2*N +1)
upper_bound[-1] = np.inf
linear_constraint = LinearConstraint(np.identity(2 * N + 1), lower_bound, upper_bound)

# Definimos restricción de llegar a cero en tiempo final
constraint_tf = ({'type': 'eq', 'fun': lambda U: EulerTf(X_0, U) - np.array([0,0,0])}) # Llegar a (0,0,0) en tiempo final

# Resolver el problema de optimización para encontrar el control que minimiza la distancia
result = minimize(objective_function, initial_guess, method='SLSQP', constraints=[linear_constraint, constraint_tf])
optimal_control_tf = result.x
optimal_control = optimal_control_tf[:2*N]
optimal_tf = optimal_control_tf[-1]
```

Respecto del funcionamiento del método, se verifica numéricamente que la convergencia del método a una solución, depende fuertemente de la iteración inicial para *minimize*. A continuación presentamos 3 configuraciones, para las cuales 2 llegan a una solución factible y la llega a una solución infactible, entregando un tiempo óptimo demasiado pequeño que no permite a la partícula cumplir la restricción a tiempo final de ser cero. Respecto a los casos en donde se entrega una solución factible, en el primero consideramos un control constante como iteración inicial, y en el otro, consideramos el control óptimo para llevar a la particula en $t_f = 2$ junto a tiempo $t_f = 2$ como iteración inicial. Los controles óptimos que se encuentran son casi de tipo Bang-Bang, en el Ejercicio 7 veremos que BOCOP entrega una solución de tipo Bang-Bang.

7.1. Solución con iteración inicial: N = 100, u(t) = -5, v(t) = -5, $t_f = 0.8$

Tiempo óptimo: $t_f = 1.8129337913154038$

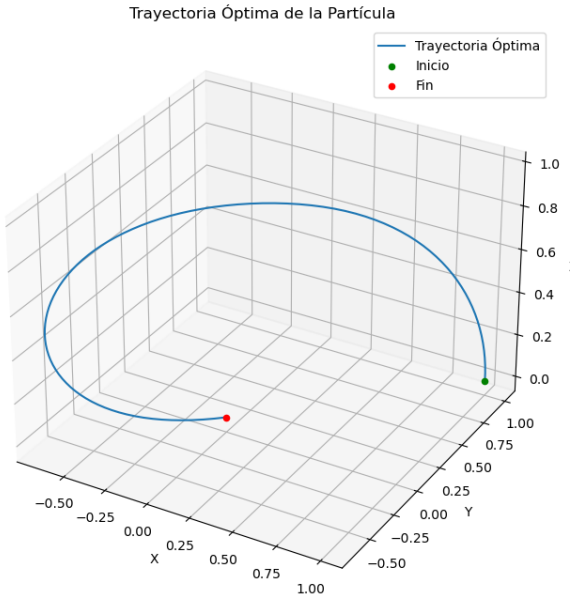


Figura 8: Trayectoria óptima del problema (P) discretizado en $[0,t_f]$

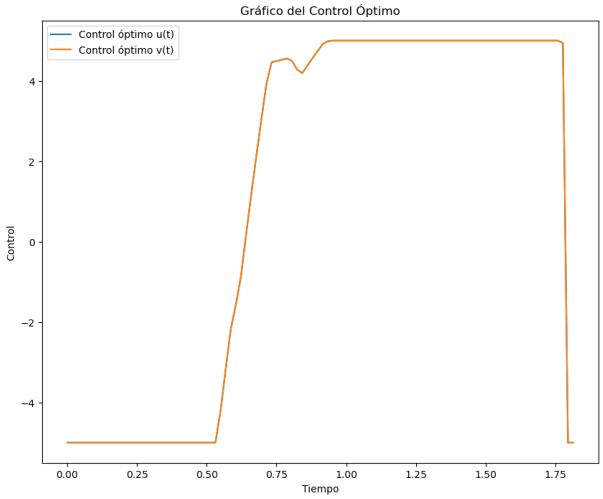


Figura 9: Control óptimo del problema (P) discretizado en $[0,t_f]$

Comentario: La inicialización $N = 100$, $u(t) = -5$, $v(t) = -5$, $t_f = 0.8$ entrega un par $(\bar{u}(\cdot), \bar{v}(\cdot), t_f)$ factible para el problema de tiempo mínimo, llevando la partícula a cero en tiempo $t_f \approx 1.81$, el cual se encuentra entre 1.5 y 2 como conjeturamos anteriormente.

7.2. Solución con iteración inicial: N = 150, u(t) = 0, v(t) = 1, $t_f = 3.5$

Tiempo óptimo: $t_f = 0.3450232089461417$

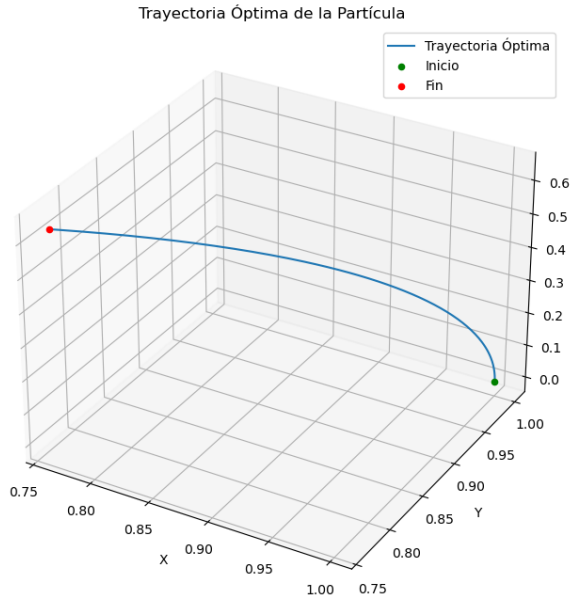


Figura 10: Trayectoria óptima del problema (P) discretizado en $[0,t_f]$

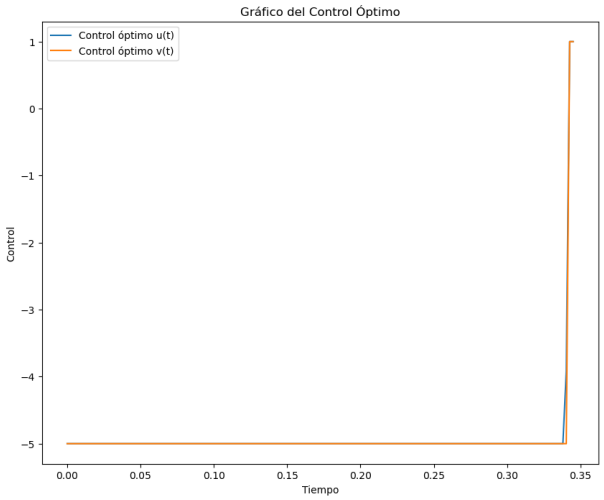


Figura 11: Control óptimo del problema (P) discretizado en $[0,t_f]$

Comentario: La configuración anterior entrega un par $(\bar{u}(\cdot), \bar{v}(\cdot), t_f)$ infactible para el problema de tiempo mínimo, pues no lleva la partícula a cero.

7.3. Solución con iteración inicial: N = 100, $u(t) = \bar{u}(t)$, $v(t) = \bar{v}(t)$, $t_f = 2$

Consideramos $\bar{u}(t)$ y $\bar{v}(t)$ los controles encontrados en el Ejercicio 5 para llevar la partícula a cero en tiempo $t_f = 2$

Usando esta inicialización, nos aseguramos de que el algoritmo comience a iterar con un par $(\bar{u}(\cdot), \bar{v}(\cdot), t_f)$ factible del problema de tiempo mínimo.

Tiempo óptimo: $t_f = 1.8271318930060751$

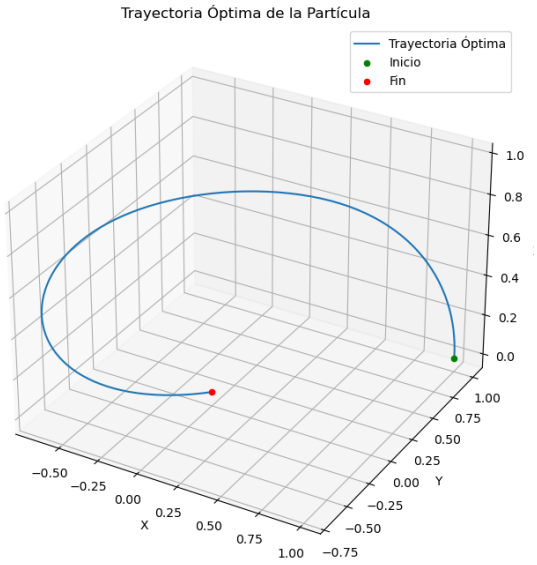


Figura 12: Trayectoria óptima del problema (P) discretizado en [0,t_f]

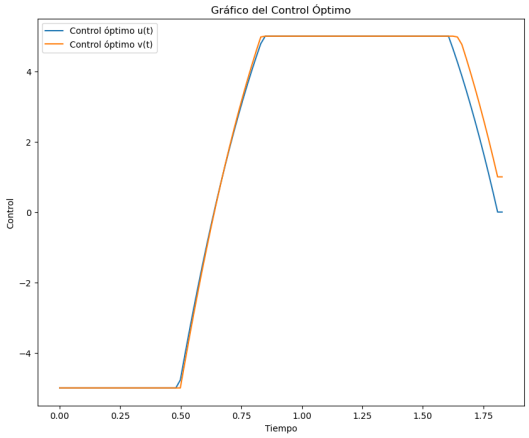


Figura 13: Control óptimo del problema (P) discretizado en [0,t_f]

Comentario: La configuración anterior entrega un par $(\bar{u}(\cdot), \bar{v}(\cdot), t_f)$ factible, muy similar al obtenido con controles iniciales $u(t) = v(t) = -5$ casi Bang-Bang.

8. Ejercicio 7

Haciendo uso de BOCOP se resuelve el problema de tiempo mínimo (sin discretizar).

Se plantean dos soluciones distintas de BOCOP, para la primera solución, no se respeta con exactitud la condición de término $X(t_f)$, en la segunda solución se encuentra un punto óptimo en un tiempo mayor, pero que lleva la partícula a cero, para este caso, se logra observar que el control óptimo y t_f óptimo del problema son muy similares que aquellos encontrados en *Python* mediante el método *minimize*, pero tomando un valor de t_f ligeramente menor, y los controles son de tipo Bang-Bang.

Se escribe el problema en los siguientes archivos, junto a los respectivos cambios que se hacen en la GUI de BOCOP.

Código 3: criterion.tpp

```
#include "header_criterion"
{
  // HERE : description of the function for the criterion
  // "criterion" is a function of all variables X[]
  criterion = final_time;
}
```

Código 4: boundarycond.tpp

```
#include "header_boundarycond"
{
  // HERE : description of the function for the initial and final conditions
  // Please give a function or a value for each element of boundaryconditions
   boundary_conditions[0] = initial_state[0];
  boundary_conditions[1] = initial_state[1];
  boundary_conditions[2] = initial_state[2];
  boundary_conditions[3] = initial_state[3];
  boundary_conditions[4] = initial_state[4];
  boundary_conditions[5] = final_state[0];
  boundary_conditions[6] = final_state[1];
  boundary_conditions[7] = final_state[4];
}
```

Código 5: dynamics.tpp

```
#include "header_dynamics"
{
  // HERE : description of the function for the dynamics
  // Please give a function or a value for the dynamics of each state variable

  Tdouble x = state[0];
  Tdouble y = state[1];
  Tdouble v_x = state[2];
  Tdouble v_y = state[3];
  Tdouble z = state[4];
  Tdouble control_u = control[0];
  Tdouble control_v = control[1];

  state_dynamics[0] = v_x;
  state_dynamics[1] = v_y;
  state_dynamics[2] = x + control_u;
  state_dynamics[3] = y + control_v;
  state_dynamics[4] = x + y;

}
```

8.1. Caso 1: $t_f = 0.693$ y $X(t_f)$ infactible

Al resolver el problema en BOCOP se encuentra una solución a tiempo $t_f = 0.693713$ con las configuraciones de optimización *mu strategy : Adaptive*

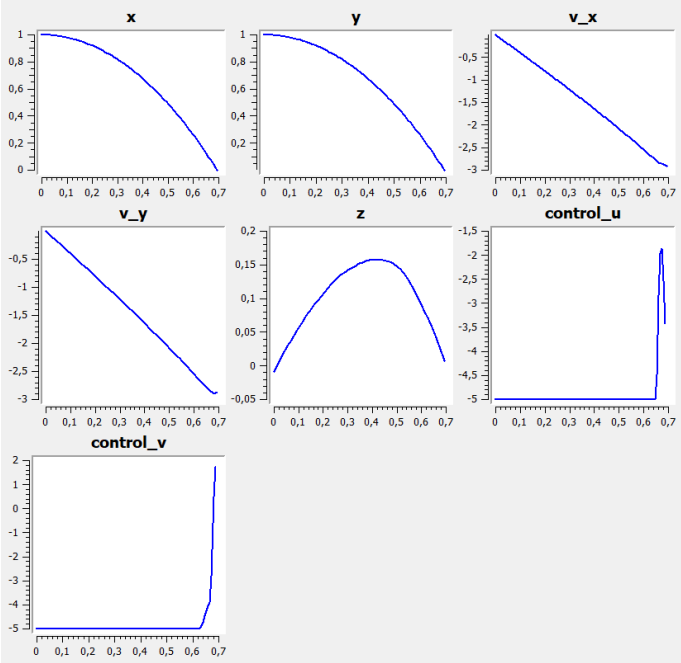
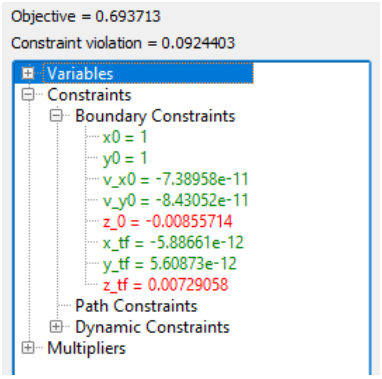


Figura 14: Caso 1: Solución de BOCOP para problema de tiempo mínimo configuración Adaptive

Dicha solución resulta infactible, pues dicha solución no lleva $z(t_f)$ a cero con una tolerancia aceptable, como se aprecia en la siguiente imágen.




```
Objective value: 6.937130e-001
Time taken: 225.10s
Ipopt solver returns 2: infeasible problem.
The restoration phase converged to a point that is a minimizer for the constraint violation (in the l_1-norm), but is not feasible for the original
problem.
This indicates that the problem may be infeasible (or at least that the algorithm is stuck at a locally infeasible point).
The returned point (the minimizer of the constraint violation) might help you to find which constraint is causing the problem.
If you believe that the NLP is feasible, it might help to start the optimization from a different point.
```

Considerando la configuración *mu strategy : Monotone* se tiene un resultado muy similar, pero igualmente infactible, con pequeñas variaciones en los controles óptimos encontrados.

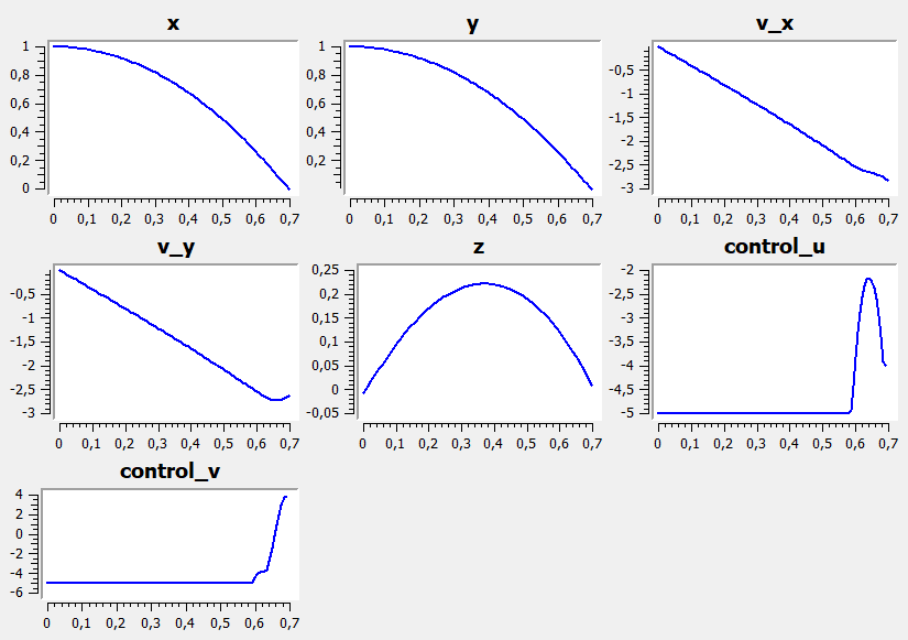
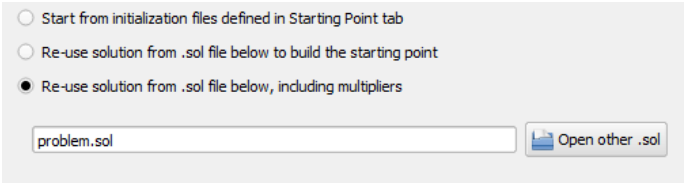


Figura 15: Caso 1: Solución de BOCOP para problema de tiempo mínimo configuración Monotone

Se reutiliza la solución encontrada y se resuelve varias veces el problema, sin embargo, no varían los resultados.



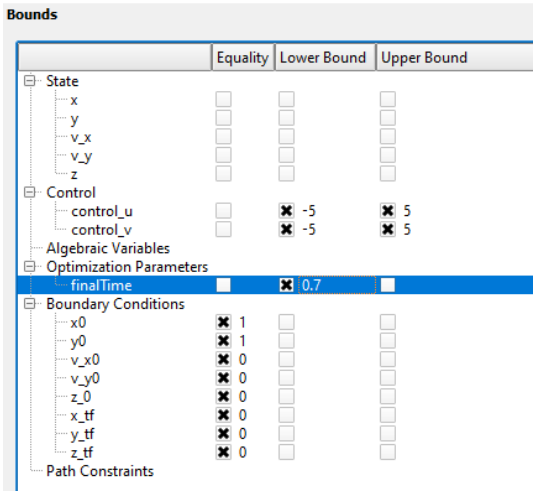
Se intentan distintos métodos de discretización que ofrece BOCOP, pero no varían las soluciones encontradas.

Se conjetura que la solución encontrada se encuentra atrapada en un mínimo local infactible, BOCOP intenta minimizar demasiado el tiempo final, y es ahí en donde encuentra el control que lleva a la particula al estado $(x(t_f), y(t_f), z(t_f)) \approx (0, 0, 0.01)$

Debido a los resultados anteriores obtenidos en Python, se cree que el tiempo óptimo debe ser mayor, y que no puede existir un control que lleve a cero la partícula en un tiempo $t_f \approx 0.69$, en el siguiente caso, impondremos como restricción que la variable t_f sea mayor que 0.69 para ver si encuentra el mismo control óptimo que dió *Python* y que este sea factible para una tolerancia pequeña en las restricciones.

8.2. Caso 2: $t_f = 1.77$ y $X(t_f)$ factible

Consideramos una restricción para t_f de modo que se encuentre acotada inferiormente por 0.7, se busca un tiempo final y control óptimo similar que aquel obtenido en el Ejercicio 6 con valor óptimo de $t_f \approx 1.82$, para ello, usamos la GUI de BOCOP como se aprecia en la siguiente figura



Se resuelve el problema tanto para las configuraciones Adaptive y Monotone, y en ambas se encuentran soluciones factibles.

```
Number of objective function evaluations             = 25
Number of objective gradient evaluations             = 25
Number of equality constraint evaluations            = 25
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 25
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 24
Total CPU secs in IPOPT (w/o function evaluations)   =      1.472
Total CPU secs in NLP function evaluations           =     16.123

EXIT: Optimal Solution Found.


Objective value: 1.776747e+000
Time taken: 17.65s
Ipopt solver returns 0: solve SUCCEEDED!
```

La solución entregada toma como tiempo óptimo $t_f \approx 1.77$, el cual es muy similar al tiempo óptimo obtenido en el Ejercicio 6.

Además, se verifica que para distintos métodos de discretización que ofrece BOCOP que se obtienen resultados símilares.

Se conjetura que la restricción $t_f \geq 0.7$ permite escapar a BOCOP del mínimo local anterior, que aparentemente resulta ser muy bueno, pues permite llevar la particula casi a cero en un tiempo $t_f \approx 0.69$ el cual es considerablemente menor, por la solución óptima real del problema, considerando una factibilidad estricta.

El control óptimo, tiempo óptimo, y trayectoria asociada obtenidos para esta configuración son los siguientes:

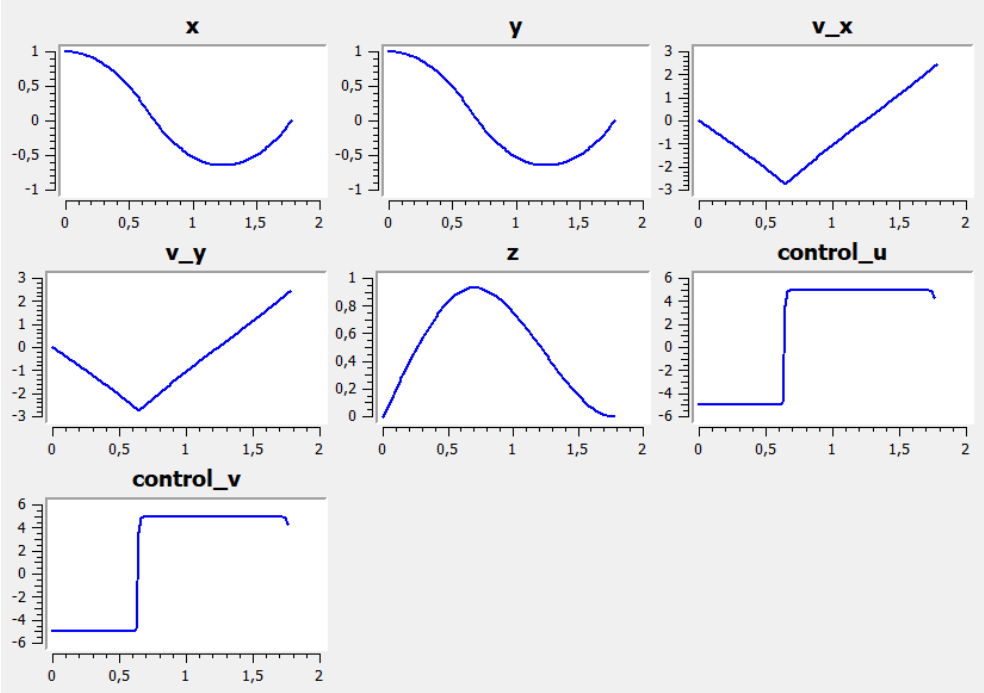
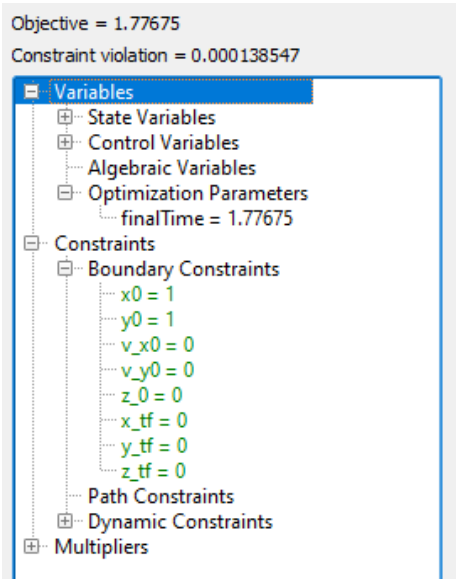


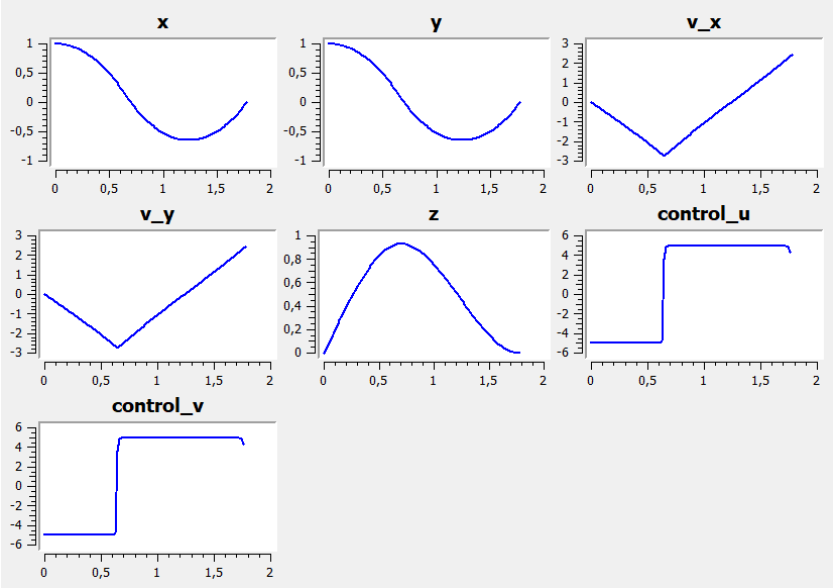
Figura 16: Caso 2: Solución de BOCOP para problema de tiempo mínimo

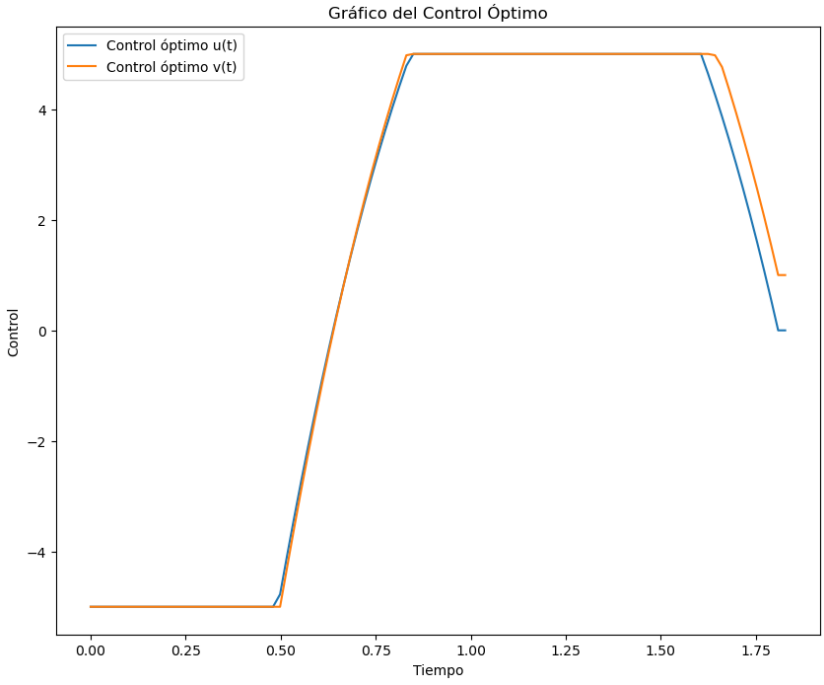
Se verifica que las condiciones finales se respetan con precisión observando la variable 'Contraint violation'.



Por lo demás, se obtienen resultados similares con los distintos métodos de discretización y la estrategia Adaptive y Monotone.

Notamos que el control óptimo resulta Bang-Bang, y el tiempo óptimo de $t_f = 1.776$ resulta mejor que aquel obtenido en el Ejercicio 6, a continuación se presentan ambas soluciones para apreciar la similitud de los controles, y en consecuencia, de la solución de la ecuación diferencial.





Observamos que la solución del Ejercicio 6, que es menos óptima que esta última entregada por BOCOP, tiene una forma muy similar que aquella dada por BOCOP, pero menos similar a un control Bang-Bang, por lo que suponemos que la solución mientras más óptima, más se asemeja a un control Bang-Bang, como es en el caso de BOCOP y que en el límite es de tipo Bang-Bang, lo anterior se respalda en el principio del máximo de Pontryagin.

Respecto a las trayectorias, se aprecia que aquellas dadas por BOCOP y las vistas en el Ejercicio 6 son similares, en este último, se aprecia que x e y van disminuyendo desde 1, pasando luego por cero hasta aproximadamente -0.5 y luego yendo hacia cero, por otro lado que en z, se aprecia que sube para luego bajar hasta cero, mismo comportamiento descrito por BOCOP.