

Laboratorio 1

Uso de Python - BOCOP

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

Santiago de Chile

Índice de Contenidos

1. Ejercicio 1: (Ecuaciones Diferenciales Ordinarias)	1
2. Ejercicio 2 (Sistemas Dinámicos Controlados)	5
3. Ejercicio 3 (Optimización lineal)	8
4. Ejercicio 4 (Optimización no lineal)	9
5. Ejercicio 5: (solve - Sympy , fsolve - Scipy)	10
6. Uso de BOCOP	12
6.1. Problema 6: Problema de Goddard	12
6.2. Problema 7	17

Índice de Figuras

1. Solución del sistema de ecuaciones diferenciales	2
2. Solución del sistema de ecuaciones diferenciales	2
3. Solución del sistema de ecuaciones diferenciales	3
4. Solución del sistema de ecuaciones diferenciales para 10 condiciones iniciales distintas	3
5. Solución del sistema de ecuaciones diferenciales para 10 condiciones iniciales distintas	4
6. Solución del sistema controlado para $u(t) = 0$	6
7. Solución del sistema controlado para $u(t) = 1$	6
8. Solución del sistema controlado para $u(t) = -\cos(t)$	7
9. Solución del sistema controlado para $u(t) = t^2$	7
10. Gráfico del sistema	10
11. Resolución de BOCOP para $m = 0.5$	13
12. Resolución de BOCOP para $m = 0.1$	14
13. Resolución de BOCOP para $m = 2.7$	14
14. Visualización por NotePad++ del archivo boundarycond.tpp	15
15. Resolución de BOCOP con $v(t_f) = 0.1$	15
16. Visualización por NotePad++ del archivo criterion.tpp	16
17. Resolución de BOCOP con la función objetivo de la forma $m(t_f) + v(t_f)/5.2$.	16
18. Resolución de BOCOP del problema equivalente tipo Mayer	18
19. Resolución de BOCOP del problema modificado	21

1. Ejercicio 1: (Ecuaciones Diferenciales Ordinarias)

Dado el sistema de ecuaciones diferenciales no lineal

$$\begin{cases} \dot{x} = 3x - xy \\ \dot{y} = y + \log(y) \end{cases}$$

con condición inicial $(x_0, y_0) = (-5, 1)$, queremos describir su comportamiento para $t \in [0, 3]$.

a) Encontrar la solución general del sistema (sin considerar la condición inicial) usando `dsolve` de `sympy`. Utilice la fórmula obtenida para graficar la solución del sistema en el intervalo $[0, 3]$ (ahora sí considerando la condición inicial).

Resolviendo el sistema de EDOs mediante *Sympy* se entregan las siguientes soluciones incorrectas:

$$x(t) = e^{C_1 + e^{C_2 + t} - 3} ; \quad y(t) = C_1 - \log(e^{C_1 + e^{C_2 + t} - 3})$$

En primera instancia notamos que $y(t)$ depende de dos constantes, pero se puede llegar a una bajo simplificaciones, por lo demás, al intentar evaluar las condiciones iniciales se llega a un sistema infactible. Más aún, al imprimir las soluciones añadiendo condiciones iniciales en la función `dsolve`, el código arroja presencia de números complejos.

Por otro lado, notando que la EDO de $y(t)$ no depende de $x(t)$, intentamos resolver el sistema de forma independiente, a lo que *Sympy* entrega como solución la relación que tiene $y(t)$ al hacer variables separables

$$- \int_1^{y(t)} \frac{1}{y + \log(y)} dy = C - t$$

Usando que $y(0) = 1$ se puede deducir que $C_1 = 0$ y entonces se tiene la relación

$$\int_1^{y(t)} \frac{1}{y + \log(y)} dy = t$$

La cual no entrega mucha información respecto de la solución.

Se documentan los errores dentro del código.

b) Resolver el sistema de forma numérica en el intervalo de tiempo $[0,3]$ mediante *solve_ivp* de *scipy*. Graficar las soluciones.

Formulando el problema bajo la estructura

```
1 diff_eq_sol = solve_ivp(diff_eq, t_span, z0, dense_output = True)
```

```
2
```

Se obtiene la siguiente solución para la ecuación diferencial

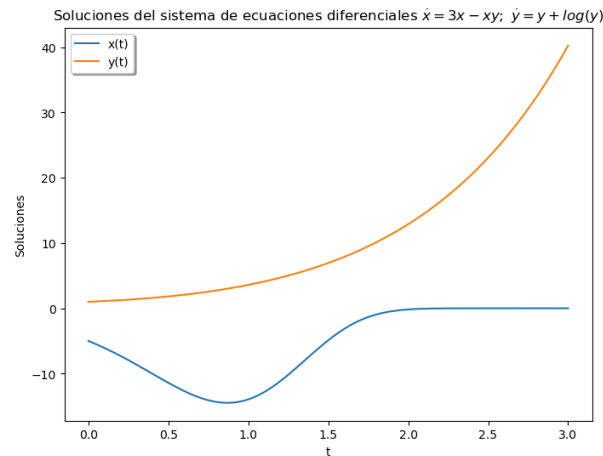
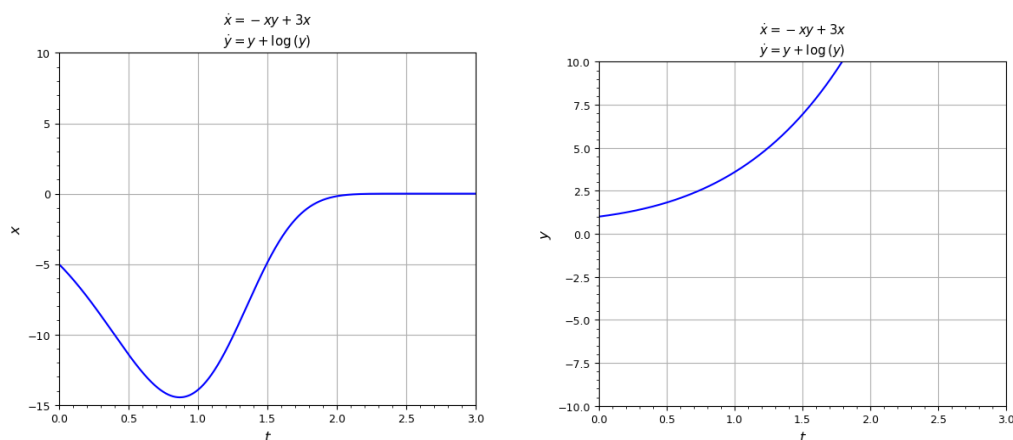


Figura 1: Solución del sistema de ecuaciones diferenciales

c) Utilice el applet *pyplane* para dibujar los diagramas de fase del sistema para al menos 10 distintas condiciones iniciales. En particular muestre el mismo punto inicial utilizado antes.

Se procede a resolver la ecuación mediante *PyPlane* con condiciones iniciales $(x_0, y_0) = (-5, 1)$, en donde se obtienen los mismos resultados que la solución entregada por *Scipy*



(a) Solución para $x(t)$

(b) Solución para $y(t)$

Figura 2: Solución del sistema de ecuaciones diferenciales

Por lo demás *PyPlane* permite encontrar el diagrama de fases de la ecuación diferencial y la trayectoria de la solución en el tiempo.

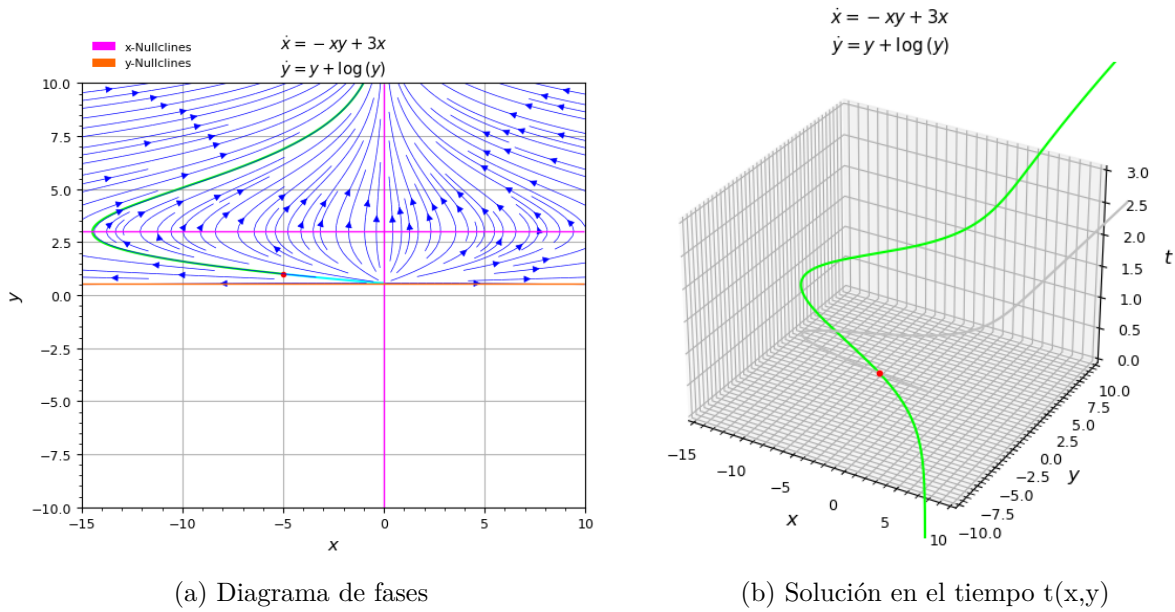


Figura 3: Solución del sistema de ecuaciones diferenciales

Además, se realizan simulaciones con 10 condiciones iniciales distintas

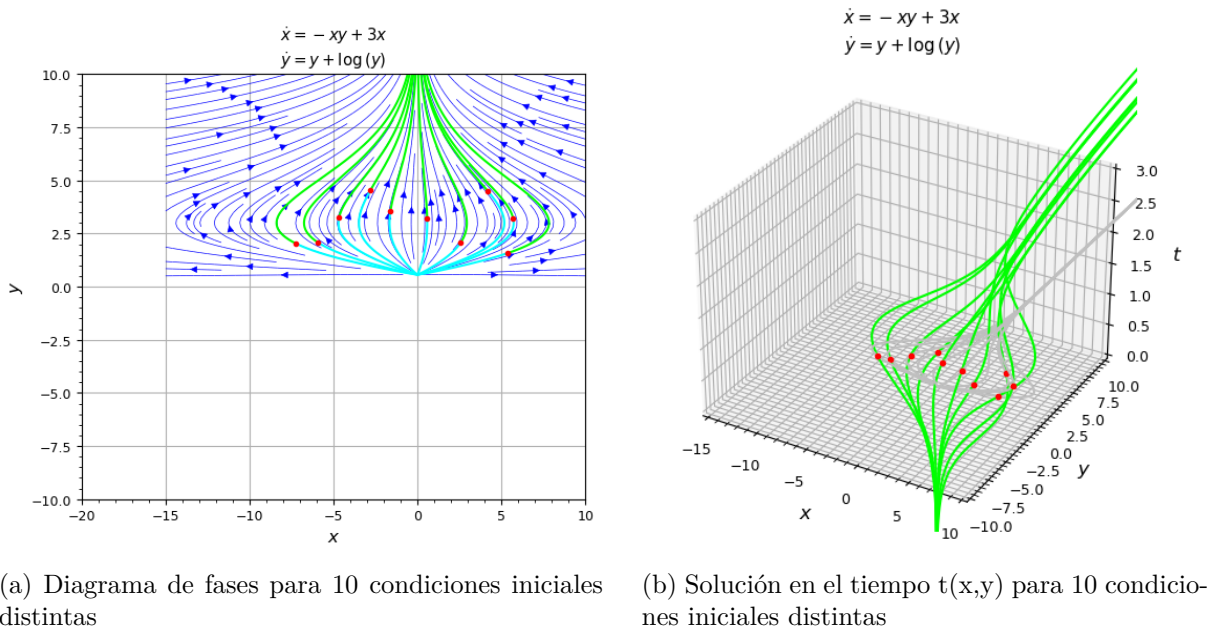
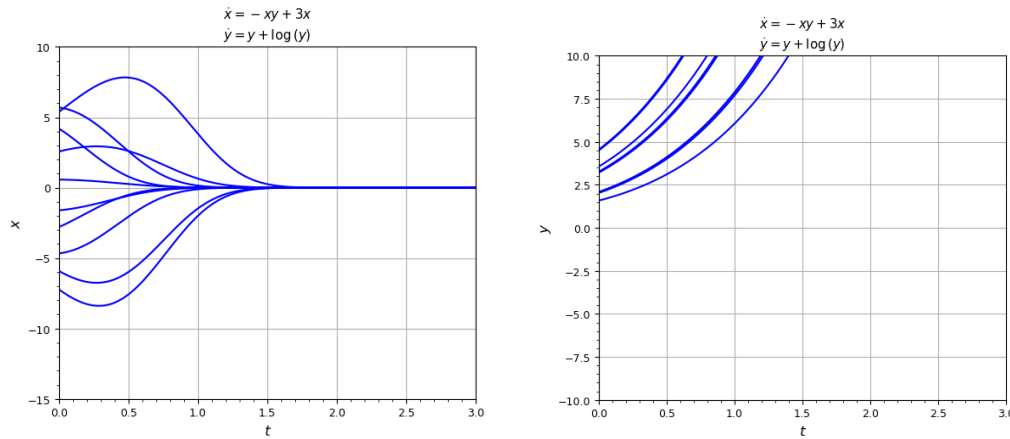


Figura 4: Solución del sistema de ecuaciones diferenciales para 10 condiciones iniciales distintas

Las soluciones para $x(t)$ y $y(t)$ de manera particular toman las siguientes formas:



(a) Solución de $x(t)$ para 10 condiciones iniciales distintas

(b) Solución de $y(t)$ para 10 condiciones iniciales distintas

Figura 5: Solución del sistema de ecuaciones diferenciales para 10 condiciones iniciales distintas

Notamos que todas las soluciones de $y(t)$ crecen de forma exponencial, podemos ver de forma específica que considerando $\dot{y} = y + \log(y)$ con condición inicial $y(0) = 1$, luego $\log(y(0)) = 0$ y como $\exp(t)$ es solución de $\exp(t) = \exp(t)$ con $\exp(0) = 1$ deducimos que $\dot{y}(t) \geq \exp(t) \implies y(t) \geq \exp(t) \forall t \in [0, 3]$

Por otro lado, notamos que todas las soluciones de $x(t)$ tienden a cero, esto pues el crecimiento de x cumple la ecuación $\dot{x} = -xy + x$, sabemos que $y(t) \rightarrow \infty$ cuando $t \rightarrow \infty$ por lo que si $x(t) \gg 0$, entonces $\dot{x}(t) \ll 0$ y entonces $x(t)$ tendera a decrecer, análogamente con $x(t) \ll 0$, y notamos que $x = 0$ es punto de equilibrio de $x(t)$ pues soluciona $\dot{x}(t) = 0 = -xy + 3x$.

d) Compare los resultados de los tres métodos y escriba sus conclusiones.

Respecto a *SymPy*, no se logran obtener soluciones para la ecuación diferencial, pues al tratar el sistema de ecuaciones de manera conjunta entrega soluciones infactibles, por lo demás, cuando intenta resolverse la ecuación solo para $y(t)$ que es independiente de $x(t)$, entrega una expresión implícita de la solución

$$\int_1^{y(t)} \frac{1}{y + \log(y)} dy = t$$

Podemos decir que la ecuación no fue resuelta mediante esta librería

En cuanto a *Scipy*, se logra resolver la ecuación diferencial mediante la función *solveivp*, por lo demás, a diferencia de *Scipy*, la solución se entrega de manera numérica mientras que *Scipy* entrega una formula para las soluciones.

Finalmente se hace uso de *PyPlane*, se clona el repositorio y se instala mediante *Anaconda Prompt* accediendo a la carpeta del repositorio y escribiendo el comando *pip install .* para luego ejecutar el programa escribiendo el comando *python run pyplane.py*.

Se escribe el sistema de ecuaciones con condiciones iniciales $(x_0, y_0) = (-5, 1)$ y se obtienen las mismas soluciones que aquellas obtenidas mediante *Scipy*, luego se prueban distintas condiciones iniciales.

2. Ejercicio 2 (Sistemas Dinámicos Controlados)

Consideramos el sistema de control en \mathbb{R}^2

$$\dot{x}(t) = Ax(t) + Bu(t) \quad \text{con} \quad A = \begin{pmatrix} 1 & 2 & 2 \\ -5 & 3 & 4 \\ 0 & 2 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}$$

con condiciones iniciales $x_0 = (0, 0, 0)$. Resolver el sistema de forma numérica en el intervalo de tiempo $[0, 1]$ en Python mediante *solveivp* para los siguientes controles de lazo abierto (open loop):

$$u(t) = 0; \quad u(t) = 1; \quad u(t) = -\cos(t); \quad u(t) = t^2.$$

Grafique las trayectorias.

Formulamos el sistema de ecuaciones como $w(t) = (x(t), y(t), z(t)) \in \mathbb{R}^3$ como

$$\dot{w}(t) = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix} = \begin{pmatrix} x(t) + 2y(t) + 2z(t) + 2u(t) \\ -5x(t) + 3y(t) + 4z(t) \\ 2y(t) + u(t) \end{pmatrix}$$

Con el fin de escribir las dinámicas de $w(t)$ bajo la estructura de *solveivp*.

```

1  # Condiciones iniciales
2  t_span2 = np.array([0,1])
3  x_0 = np.array([0, 0, 0])
4
5  # Resolvemos el sistema controlado para u
6  diff_sol_u = solve_ivp(diff_eq_u_control, t_span2, x_0, dense_output=True)
7

```

Para cada uno de los controles $u_i(t)$ y se procede a crear funciones que grafiquen $(w(t), u(t))$ en función de t y la trayectoria de $w(t)$ a lo largo del tiempo $t \in [0, 1]$

A continuación se presentan las soluciones del sistema y las trayectorias que siguen para cada uno de los controles.

Caso 1: $u(t) = 0$

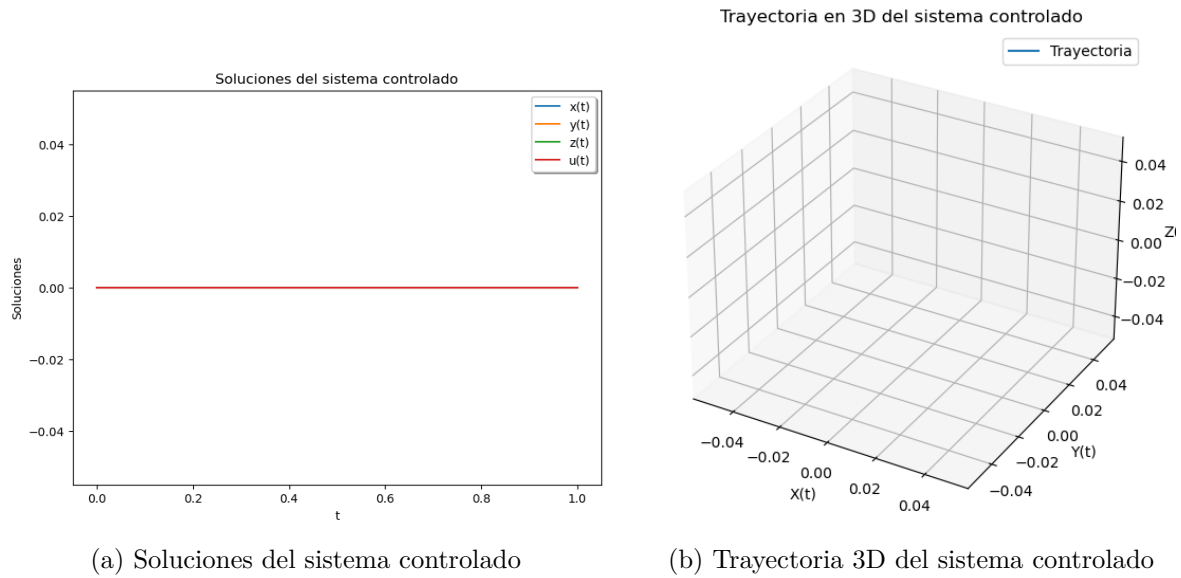


Figura 6: Solución del sistema controlado para $u(t) = 0$

Caso 2: $u(t) = 1$

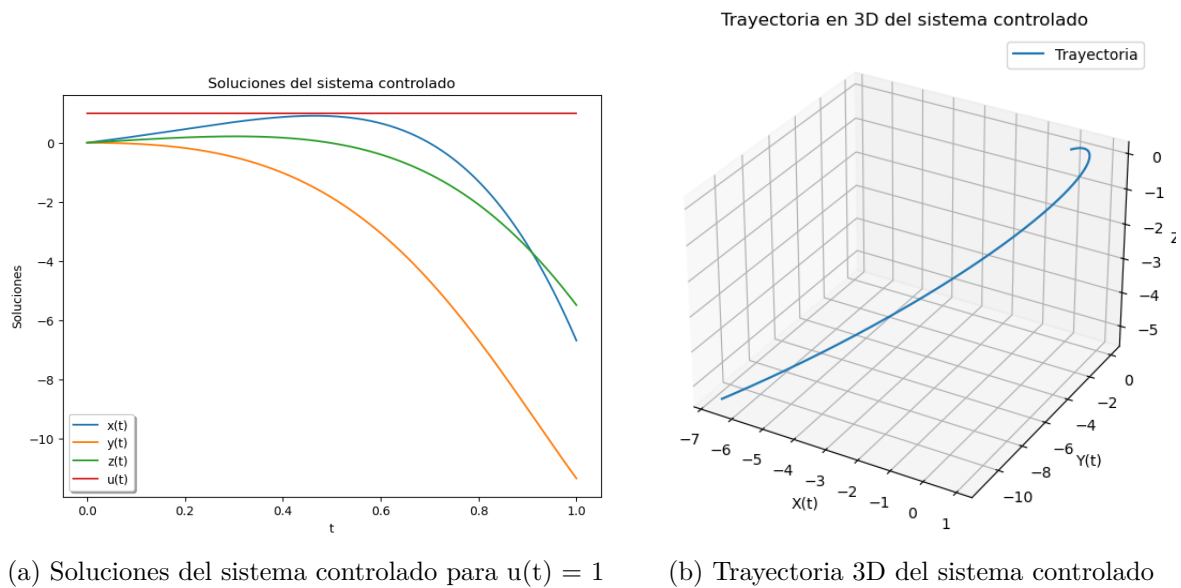
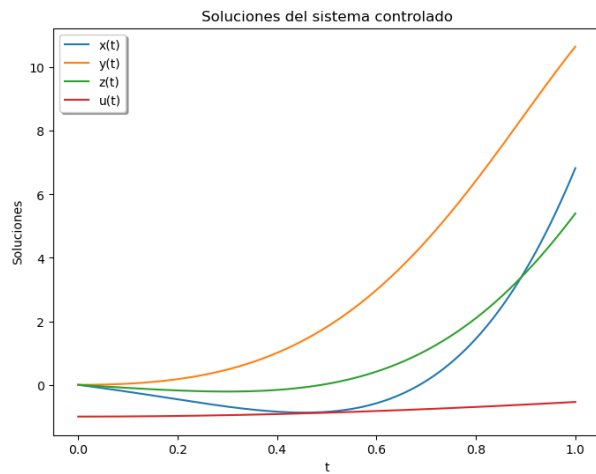
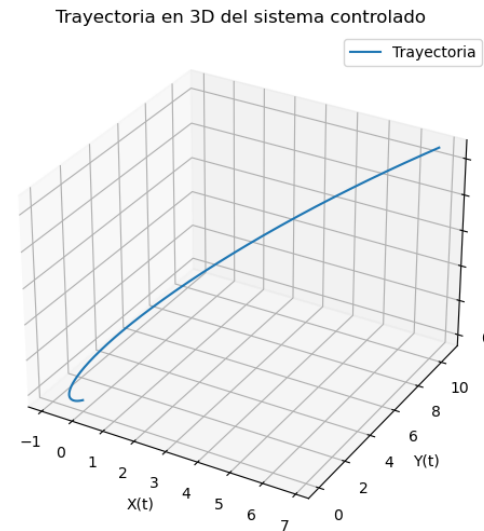
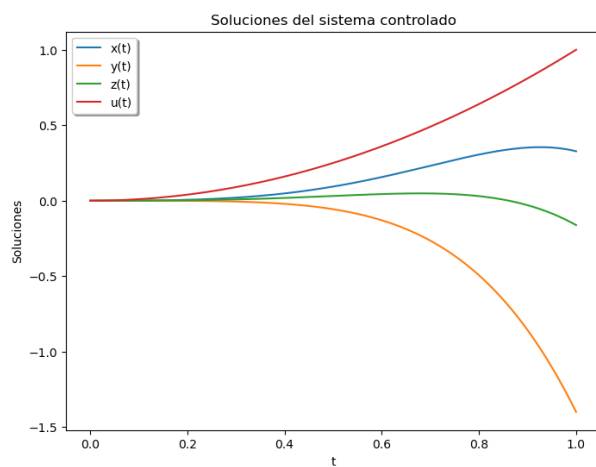
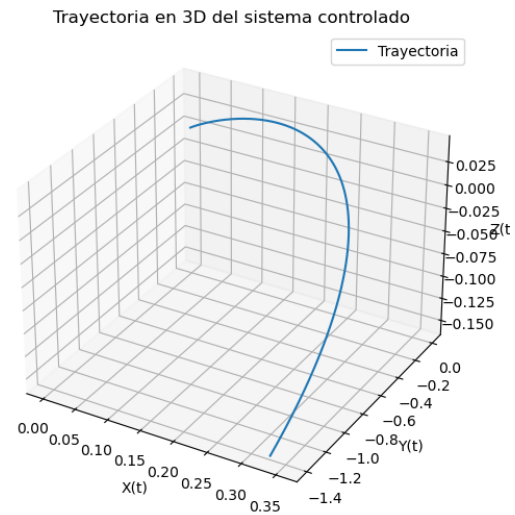


Figura 7: Solución del sistema controlado para $u(t) = 1$

Caso 3: $u(t) = -\cos(t)$ (a) Soluciones del sistema controlado para $u(t) = -\cos(t)$ 

(b) Trayectoria 3D del sistema controlado

Figura 8: Solución del sistema controlado para $u(t) = -\cos(t)$ **Caso 4:** $u(t) = t^2$ (a) Soluciones del sistema controlado para $u(t) = t^2$ 

(b) Trayectoria 3D del sistema controlado

Figura 9: Solución del sistema controlado para $u(t) = t^2$

3. Ejercicio 3 (Optimización lineal)

Resolver el problema de programación lineal (PL).

$$\left\{ \begin{array}{ll} \min_{(x,y,z) \in \mathbb{R}^3} & f(x,y,z) := -3x + y - 11z \\ & -x + y - 7z = 13 \\ & 3x - y + 2z \leq 0 \\ \text{sa.} & 2x + 4z \leq 3 \\ & 2x - 4y + z \leq 3 \\ & x, y, z \geq 0 \end{array} \right.$$

Para ello se utiliza *linprog* de *Scipy* y se exploran los distintos métodos disponibles. Buscamos resolver el problema de programación lineal planteándolo en la forma:

$$\begin{aligned} & \min_x c^T x \\ & \text{such that } A_{ub}x \leq b_{ub}, \\ & A_{eq}x = b_{eq}, \\ & l \leq x \leq u, \end{aligned}$$

donde x es la variable de decisión; c, b_{ub}, b_{eq}, l , y u son vectores; y A_{ub} , A_{eq} son matrices, con el fin de seguir la estructura de *linprog* que es de la forma.

```

1 from scipy.optimize import linprog
2 #Estructura
3 scipy.optimize.linprog(c, A_ub=None, b_ub=None, A_eq=None, b_eq=None, bounds=
  ↳ None,
4 method='highs', callback=None, options=None, $x \backslash \text{theta}=\$ None, integrality=None)
5
```

Identificando las variables

$$c^T = [-3 \quad 1 \quad -11], \quad A_{eq} = [-1 \quad 1 \quad -7], \quad b_{eq} = 13$$

$$A_{ub} = \begin{pmatrix} 3 & -1 & 2 \\ 2 & 0 & 4 \\ 2 & -4 & 1 \end{pmatrix}, \quad b_{ub} = \begin{pmatrix} 0 \\ 3 \\ 3 \end{pmatrix}$$

Se resuelve el problema de optimización, en donde el valor óptimo del problema es $f(x) = 10.0$ y el óptimo se alcanza en $(x, y, z) = (1.5 \ 14.5 \ 0)$

Se prueban además los siguientes métodos disponibles:

```
linprog(method='simplex')
linprog(method='interior-point')
linprog(method='revised simplex')
linprog(method='highs-ipm')
linprog(method='highs-ds')
linprog(method='highs')
```

El método *simplex* y *highs-ipm* alcanzan el óptimo en el mismo punto $x = (1.5 \ 14.5 \ 0)$, por otro lado *revised simplex* alcanza el mismo valor óptimo $f(x) = 10$ pero en el punto $x = (0 \ 18.25 \ 0.75)$ y el método de *interior point* alcanza un valor levemente mayor en el punto $x = (1.03000962 \ 15.67497595 \ 0.23499519)$ (puede ser error numérico)

4. Ejercicio 4 (Optimización no lineal)

Una compañía fabrica cajas (paralelepípedos rectangulares) para transporte. Se requiere fabricar la caja con el mayor volumen posible, tal que la suma de todas sus aristas no superen los 500 cm y cuya base no tenga menos de 3000 cm² de superficie. Resolver utilizando el comando *minimize* de *Scipy*

El problema de optimización asociado, considerando el largo de la base como x , el ancho como y y la altura del paralelepípedo como z , se plantea como

$$\max V(x, y, z) = xyz, \quad \text{s.a.} \quad 4(x + y + z) \leq 500, \quad xy \geq 3000$$

Pues consideramos que el área de la base se corresponde a la cantidad xy , además el paralelepípedo contiene 4 aristas de tamaño x , tamaño y y tamaño z , por lo que la suma es $4(x + y + z)$.

Equivalentemente

$$\min V(x, y, z) = -xyz, \quad \text{s.a.} \quad 4(x + y + z) \leq 500, \quad xy \geq 3000$$

Notemos que $x = (55 \ 55 \ 10)$ es un punto factible

Para *minimize* definimos las restricciones de desigualdad de manera que queden no negativas, y esto es, mayores o iguales a cero. Este toma la estructura

```
1 from scipy.optimize import minimize
2 constraints = [
3     {'type': 'ineq', 'fun': constraint1},
4     {'type': 'ineq', 'fun': constraint2}]
5 result = minimize(objective_function, initial_guess, constraints=constraints, bounds
6     ↪ =((0, None), (0, None), (0, None)))
```

5. Ejercicio 5: (solve - Sympy , fsolve - Scipy)

Encontrar los puntos de intersección (x, y) de las siguientes cónicas:

- $(x - 4)^2 + 3y^2 = 20$
- $x + 8(y + 1)^2 = 10$

a) Utilizando el comando *fsolve* de *Scipy*, que sigue la estructura

```

1 # Método fsolve de Scipy
2 from scipy.optimize import fsolve
3 # Estructura del método
4 scipy.optimize.fsolve(func, x0, args=(), fprime=None, full_output=0, col_deriv=0, xtol
  ↳ =1.49012e-08, maxfev=0, band=None, epsfcn=None, factor=100, diag=None)
5

```

Se obtienen las soluciones $x = -0.46520023110602193$ e $y = 0.14374386506926$, el gráfico del sistema se ve como

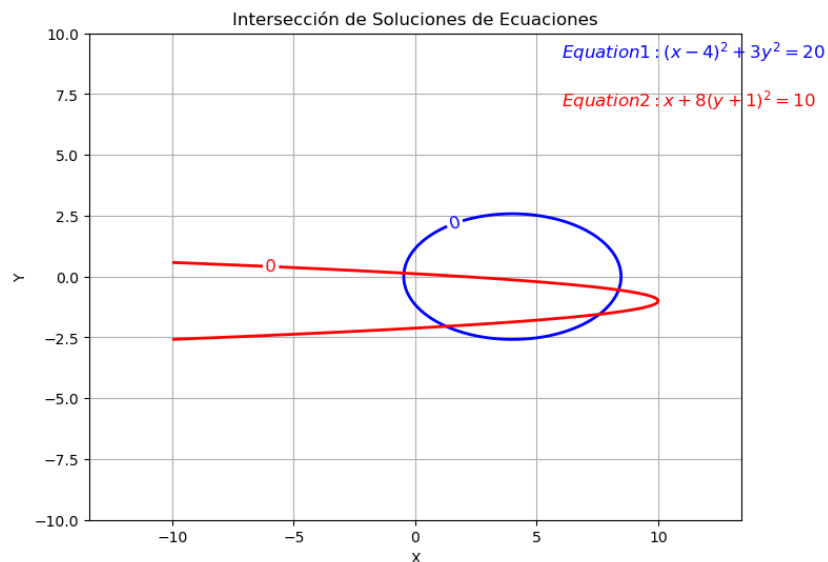


Figura 10: Gráfico del sistema

De donde vemos que las soluciones obtenidas son factibles acorde al gráfico.

b) Mediante el comando *solve* de *Sympy*, se prueba el código

```
1 from sympy import symbols, Eq, solve
2
3 # Definir las variables simbólicas
4 x, y = symbols('x y')
5
6 # Definir las ecuaciones
7 equation1 = Eq((x - 4)**2 + 3*(y**2) - 20, 0)
8 equation2 = Eq(x + 8*(y + 1)**2 - 10, 0)
9
10 # Resolver el sistema de ecuaciones
11 solution = solve((equation1, equation2), (x, y))
12
13
```

Se obtiene una solución no explícita por parte de *Sympy*, sin embargo, haciendo uso del método *nsolve* de *Sympy*

```
1 from sympy import Symbol, nsolve
2 import mpmath
3
4 mpmath.mp.dps = 15
5
6 x = Symbol('x')
7
8 y = Symbol('y')
9
10 f1 = (x - 4)**2 + 3*(y**2) - 20
11
12 f2 = x + 8*(y + 1)**2 - 10
13
14 nsolve((f1, f2), (x, y), (-1, 1))
15
```

Se obtienen las mismas soluciones $x = -0.46520023110602193$ e $y = 0.14374386506926$ que aquellas entregadas por *Scipy*

6. Uso de BOCOP

BOCOP es un programa open-source diseñado para resolver problemas de control óptimo tipo Mayer (M) a tiempo final fijo o libre y con restricciones de control y estado.

$$(M) \begin{cases} \min_{u(\cdot)} J(t_0, y(t_0), t_f, y(t_f)) \text{ (Criterio)} \\ \dot{y}(t) = f(t, y(t), u(t)) \quad \forall t \in [t_0, t_f] \text{ (Dinámica)} \\ \Phi_l \leq \Phi(t_0, y(t_0), t_f, y(t_f)) \leq \Phi_u \text{ (Condición de borde)} \\ y_l \leq y(t) \leq y_u \quad u_l \leq u(t) \leq u_u \quad \forall t \in [t_0, t_f] \text{ (Cotas)} \\ g_l \leq g(y(t), u(t)) \leq g_u \quad \forall t \in [t_0, t_f] \end{cases}$$

donde $y(\cdot) \in \mathbb{R}^n$ es el estado del sistema y $u(\cdot) \in \mathbb{R}^m$ el control. En particular, note que la función objetivo solo depende del tiempo inicial y final y de los estados iniciales y finales.

6.1. Problema 6: Problema de Goddard

Consideremos en este ejercicio el problema de Goddard, que modela el ascenso de un cohete a través de la atmósfera, y restringimos el estudio a trayectorias verticales (monodimensionales). Las variables de estado son la altitud r , la velocidad v y la masa m del cohete durante el vuelo. El cohete está sujeto a fuerzas de gravedad, aerodinámica y del motor. El tiempo final es libre, y el objetivo es alcanzar una cierta altitud con un consumo mínimo de combustible, es decir con masa final máxima. El control es la fuerza u del motor del cohete.

El problema de control óptimo es el siguiente:

$$(G) \begin{cases} \max_{u(\cdot)} & m(t_f) \\ & \dot{r} = v & r(0) = 1 \\ & \dot{v} = -\frac{1}{r^2} + \frac{1}{m}(T_{\max}u - D(r, v)) & v(0) = 0 \\ \text{sa.} & \dot{m} = -bu & m(0) = 2 \\ & D(r, v) \leq C & r(t_f) = 1.02 \\ & u(t) \in [0, 1.5] & \forall t \in [0, t_f] \end{cases}$$

y la fuerza aerodinámica es $D(r, v) = Av^2e^{-k(r-r_0)}$.

1. Escribir el problema en la forma (M) e identifique las funciones J, f, Φ, g y las cotas $\Phi_l, \Phi_u, g_l, g_u, y_l, y_u, u_l, u_u$.

Criterio: $\min_{u(\cdot)} J(t_0, y(t_0), t_f, y(t_f)) = -m(t_f)$ con $t_0 = 0$

Variables de estado: La variable de estado del problema es

$$y(t) = (r(t), v(t), m(t)) \text{ , } y(0) = (1, 0, 2)$$

Dinámica: La dinámica del problema es

$$\dot{y}(t) = \begin{bmatrix} \dot{r}(t) \\ \dot{v}(t) \\ \dot{m}(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ -\frac{1}{r^2(t)} + \frac{1}{m(t)}(T_{max}u(t) - D(r, v)) \\ -bu(t) \end{bmatrix} = f(t, y(t), u(t))$$

Condición de Borde: Escribiendo las condiciones borde de igualdad como desigualdades, escribimos

$$\Phi(t_0, y(t_0), t_f, y(t_f)) = \begin{bmatrix} r(0) \\ v(0) \\ m(0) \\ r(t_f) \end{bmatrix}, \quad \Phi_l = \Phi_u = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1.02 \end{bmatrix}$$

Restricción mixta: Tenemos solamente la restricción mixta

$$g(y(t), u(t)) = D(r, v), \quad g_u = C, \quad g_l \text{ no acotado}$$

Cotas:

$$y_l, y_u \text{ no acotados}$$

$$u_l = 0 \leq u(t) \leq u_u = 1.5 \quad \forall t \in [0, t_f]$$

2. Haga las modificaciones necesarias para resolver el problema de alcanzar la misma altitud pero con $m(0) \in \{0.5, 0.1, 2.7\}$. Entregue los gráficos de la solución y el control óptimo.

Caso 1: $m = 0.5$

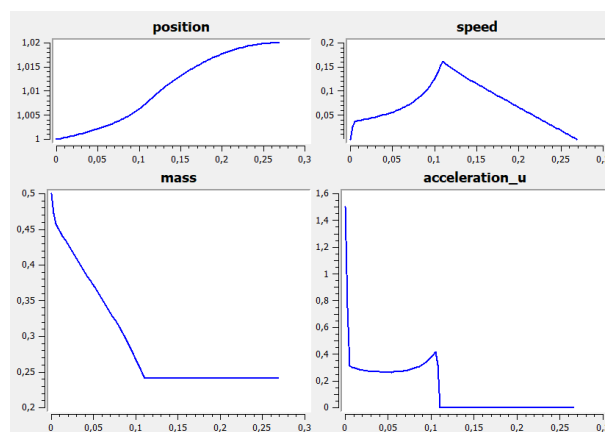


Figura 11: Resolución de BOCOP para $m = 0.5$

Caso 2: $m = 0.1$

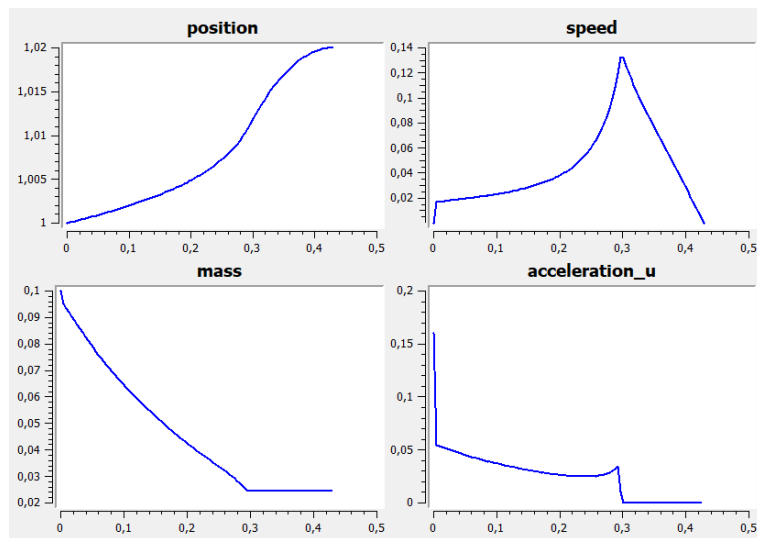


Figura 12: Resolución de BOCOP para $m = 0.1$

Caso 3: $m = 2.7$

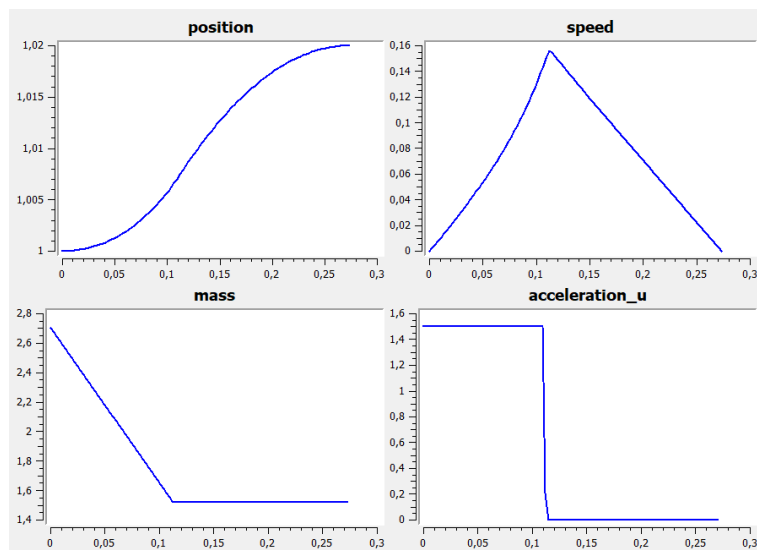


Figura 13: Resolución de BOCOP para $m = 2.7$

3. Considerando (G) original, ahora queremos que el cohete llegue a la altitud deseada con cierta velocidad para asegurarnos que no retornará a la tierra, en particular $v(t_f) = 0,1$. Haga las modificaciones necesarias para resolver este nuevo problema.

Dentro del GUI de BOCOP, añadimos una variable de condición de frontera que nombremos $v(f)$, luego de investigar los archivos del ejemplo de Goddard, se añade en el archivo *boundarycond.tpp* la línea de código

```
1 boundary_conditions[4] = final_state[1];
```

Para indicar que la condición de borde que añadimos se encuentra asociado al estado en tiempo final de la velocidad, que se corresponde al estado 1.

```

13 // dim_constants : number of constants
14 // constants : vector of constants
15
16 // Output :
17 // boundaryconditions : vector of boundary conditions ("Phi" in the example above).
18
19 // The functions of your problem here to be written in C++ code
20 // Remember that the vectors numbering in C++ starts from 0
21 // (ex: the first component of the vector state is state[0])
22
23 // Tdouble variables correspond to values that can change during optimization:
24 // states, controls, algebraic variables and optimization parameters.
25 // Values that remain constant during optimization use standard types (double, int, ...).
26
27 #include "header_boundarycond"
28
29 // INITIAL CONDITIONS FOR GODDARD PROBLEM
30 // g0 = 1, v0 = 0, m0 = 1
31 // MODELED AS 1 <= x0 <= 1, etc
32 boundary_conditions[0] = initial_state[0];
33 boundary_conditions[1] = initial_state[1];
34 boundary_conditions[2] = initial_state[2];
35
36 // FINAL CONDITIONS FOR GODDARD PROBLEM
37 // x1 >= 1.01, MODELED AS 1.01 <= x1
38 boundary_conditions[3] = final_state[0];
39 boundary_conditions[4] = final_state[1];
40
41
42
    
```

Figura 14: Visualización por NotePad++ del archivo boundarycond.tpp

Donde BOCOP entrega la solución factible del problema

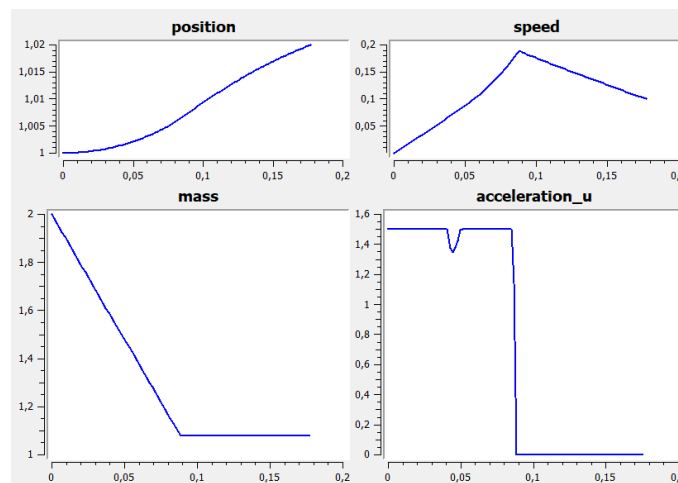
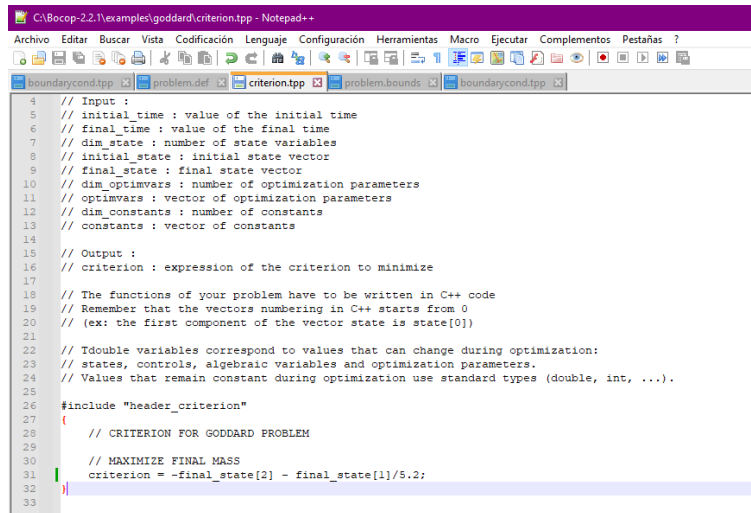


Figura 15: Resolución de BOCOP con $v(t_f) = 0.1$

4. Estamos dispuestos a sacrificar un poco de combustible, a costa de llegar con muchas más velocidad a la altura deseada. Modifique la función objetivo por una de la forma $m(t_f) + v(t_f)/5.2$. Resuelva nuevamente (G) solo con este cambio y muestre la dinámica.

Se resetean las condiciones iniciales con las que viene el problema y procedemos a modificar el archivo *criterion.tpp*, siguiendo la estructura del código se modifica el criterio, y procedemos a resolver el problema.



```

C:\Bocop-2.2\examples\goddard\criterion.tpp - Notepad++
Archivo  Editar  Buscar  Vista  Codificación  Lenguaje  Configuración  Herramientas  Macro  Ejecutar  Complementos  Pestañas  ?

boundarycond.tpp  problem.def  criterion.tpp  problem.bounds  boundarycond.tpp

4  // Input :
5  // initial_time : value of the initial time
6  // final_time : value of the final time
7  // dim_state : number of state variables
8  // initial_state : initial state vector
9  // final_state : final state vector
10 // dim_optimvars : number of optimization parameters
11 // optimvars : vector of optimization parameters
12 // dim_constants : number of constants
13 // constants : vector of constants
14
15 // Output :
16 // criterion : expression of the criterion to minimize
17
18 // The functions of your problem have to be written in C++ code
19 // Remember that the vectors numbering in C++ starts from 0
20 // (ex: the first component of the vector state is state[0])
21
22 // Double variables correspond to values that can change during optimization:
23 // states, controls, algebraic variables and optimization parameters.
24 // Values that remain constant during optimization use standard types (double, int, ...).
25
26 #include "header_criterion"
27
28 // CRITERION FOR GODDARD PROBLEM
29
30 // MAXIMIZE FINAL MASS
31 criterion = -final_state[2] - final_state[1]/5.2;
32
33

```

Figura 16: Visualización por NotePad++ del archivo criterion.tpp

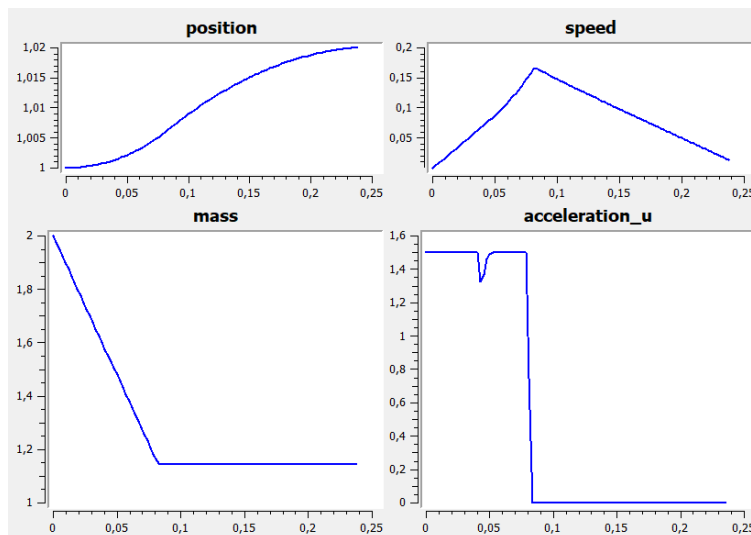


Figura 17: Resolución de BOCOP con la función objetivo de la forma $m(t_f) + v(t_f)/5.2$

6.2. Problema 7

Considere el siguiente problema de control óptimo:

$$(B) \left\{ \begin{array}{lll} \min_{u(\cdot)} & \int_0^{t_f} x(t) + y(t) dt & \\ \ddot{x}(t) = -\frac{\dot{x}(t)}{t} + u(t) & x(0) = 1 & \dot{x}(0) = 1 \\ \text{sa.} & \ddot{y}(t) = \dot{x}(t) + u(t) & y(0) = 1 \quad \dot{y}(0) = 1 \\ u(t) \in [0, 0.5] & \forall t \in [0, t_f] & \dot{y}(t_f) \geq 1.2 \end{array} \right.$$

con $t_f = 1$.

1. Primeramente identifique el sistema que modela este problema y luego introduzca una nueva variable de estado z tal que $z(t_f) = \int_0^{t_f} x(t) + y(t) dt$. ¿Cuál es la dinámica y la condición inicial de z ?

Tenemos un problema de tipo Lagrangeano que deseamos pasar a un problema tipo Mayer.

Definiendo $w = \dot{x}$ y $\Psi = \dot{y}$ se obtiene el sistema

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{w} \\ \dot{\Psi} \end{bmatrix} (t) = \begin{bmatrix} w(t) \\ \Psi(t) \\ -\frac{w(t)}{t} + u(t) \\ w(t) + u(t) \end{bmatrix} = f(t, x, y, w, \Psi) ; \quad \begin{bmatrix} x_0 \\ y_0 \\ w_0 \\ \Psi_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$u(t) \in [0, 0.5] ; \quad \Phi = \Psi(t_f) \geq 1.2$$

Definiendo la variable de estado

$$z(t) = \int_0^t x(s) + y(s) ds ; \quad z(0) = 0$$

Se obtiene la dinámica

$$\dot{\Theta} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{w} \\ \dot{\Psi} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} w(t) \\ \Psi(t) \\ -\frac{w(t)}{t} + u(t) \\ w(t) + u(t) \\ x(t) + y(t) \end{bmatrix} ; \quad \Theta_0 = \begin{bmatrix} x_0 \\ y_0 \\ w_0 \\ \Psi_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Para el problema de minimizar $z(t_f)$

2. Usando esta variable adicional, escriba un problema de Mayer (M) que sea equivalente al problema (B). Identifique las funciones J, f, Φ y g .

Por lo visto, tenemos el problema equivalente de tipo Mayer

$$\min_{u(\cdot)} J(0, \Theta_0, t_f, \Theta(t_f)) = z(t_f)$$

Sujeto a la dinámica

$$\dot{\Theta}(t) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{w} \\ \dot{\Psi} \\ \dot{z} \end{bmatrix} (t) = \begin{bmatrix} w(t) \\ \Psi(t) \\ -\frac{w(t)}{t} + u(t) \\ w(t) + u(t) \\ x(t) + y(t) \end{bmatrix}; \quad \Theta_0 = \begin{bmatrix} x_0 \\ y_0 \\ w_0 \\ \Psi_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$u(t) \in [0, 0.5]; \quad \Phi = \Psi(t_f) \geq 1.2$$

Sin restricciones mixtas $g(\Theta(t), u(t))$ y sin cotas para $\Theta(t)$ y donde las condiciones de borde Φ involucran a $\Theta(0) = \Theta_0$ y $\Psi(t_f) \geq 1.2$

3. Resuelva el problema (B). Grafique los resultados encontrados.

Creando un problema nuevo en BOCOP, y modificando los archivos *criterion.tpp*, *boundarycond.tpp*, *dynamics.tpp* se obtienen las siguientes soluciones para el sistema

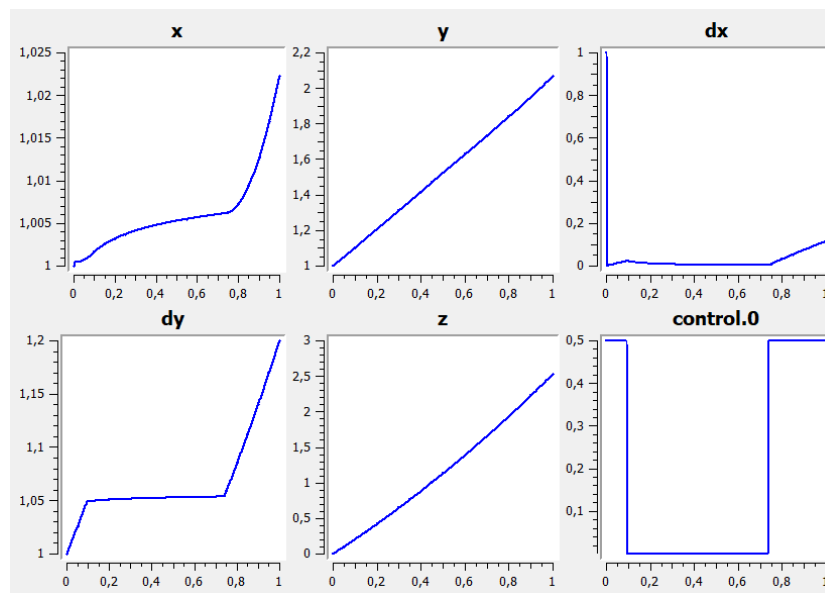


Figura 18: Resolución de BOCOP del problema equivalente tipo Mayer

En *criterion.tpp* añadimos la función objetivo $z(t_f)$

```
1 // HERE : description of the function for the criterion: z(tf)
2 // "criterion" is a function of all variables X[]
3 criterion = final_state[dim_state-1]
4
```

En *dynamics.tpp* añadimos la dinámica del problema

```
1 // HERE : description of the function for the dynamics
2
3 Tdouble x = state[0];
4 Tdouble y = state[1];
5 Tdouble dx = state[2];
6 Tdouble dy = state[3];
7 Tdouble z = state[4];
8
9 state_dynamics[0] = dx;
10 state_dynamics[1] = dy;
11 state_dynamics[2] = -dx/time + control[0];
12 state_dynamics[3] = dx + control[0];
13 state_dynamics[4] = x + y;
14
```

Y en *boundarycond.tpp* añadimos las restricciones de borde

```
1 // HERE : description of the function for the initial and final conditions
2 // Please give a function or a value for each element of boundaryconditions
3 boundary_conditions[0] = initial_state[0];
4 boundary_conditions[1] = initial_state[1];
5 boundary_conditions[2] = initial_state[2];
6 boundary_conditions[3] = initial_state[3];
7 boundary_conditions[4] = initial_state[4];
8 boundary_conditions[5] = final_state[3]; // dy(tf) > 1.2
9
```

Y se inicializan las condiciones iniciales dentro del GUI de BOCOP. Se adjuntan los problemas creados en BOCOP.

4. Considere ahora que en la función objetivo se quiere reemplazar el termino $y(t)$ por $Y(t) = \int_0^t y(s)ds$, con $Y(0) = 0$. Resuelva el problema y entregue sus gráficos.

Considerando las nuevas variables de estado

$$Y(t) = \int_0^t y(s)ds ; z(t) = \int_0^t x(s) + Y(s)ds$$

Se tiene

$$\dot{Y}(t) = y(t) ; \dot{z}(t) = x(t) + Y(t)$$

Obtenemos el problema tipo Mayer

$$\min_{u(\cdot)} J(0, \Theta_0, t_f, \Theta(t_f)) = z(t_f)$$

Sujeto a la dinámica

$$\dot{\Theta}(t) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{w} \\ \dot{\Psi} \\ \dot{Y} \\ \dot{z} \end{bmatrix} (t) = \begin{bmatrix} w(t) \\ \Psi(t) \\ -\frac{w(t)}{t} + u(t) \\ w(t) + u(t) \\ y(t) \\ x(t) + Y(t) \end{bmatrix} ; \Theta_0 = \begin{bmatrix} x_0 \\ y_0 \\ w_0 \\ \Psi_0 \\ Y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$u(t) \in [0, 0.5] ; \Phi = \Psi(t_f) \geq 1.2$$

Sin restricciones mixtas $g(\Theta(t), u(t))$ y sin cotas para $\Theta(t)$ y donde las condiciones de borde Φ involucran a $\Theta(0) = \Theta_0$ y $\Psi(t_f) \geq 1.2$. Añadiendo la nueva variable de estado con las dinámicas

```

1 Tdouble x = state[0];
2 Tdouble y = state[1];
3 Tdouble dx = state[2];
4 Tdouble dy = state[3];
5 Tdouble Y = state[4];
6 Tdouble z = state[5];
7
8 state_dynamics[0] = dx;
9 state_dynamics[1] = dy;
10 state_dynamics[2] = -dx/time + control[0];
11 state_dynamics[3] = dx + control[0];
12 state_dynamics[4] = y;
13 state_dynamics[5] = x + Y;
```

Se obtiene la solución, con un control óptimo tipo Bang-Bang, al igual que en el caso anterior.

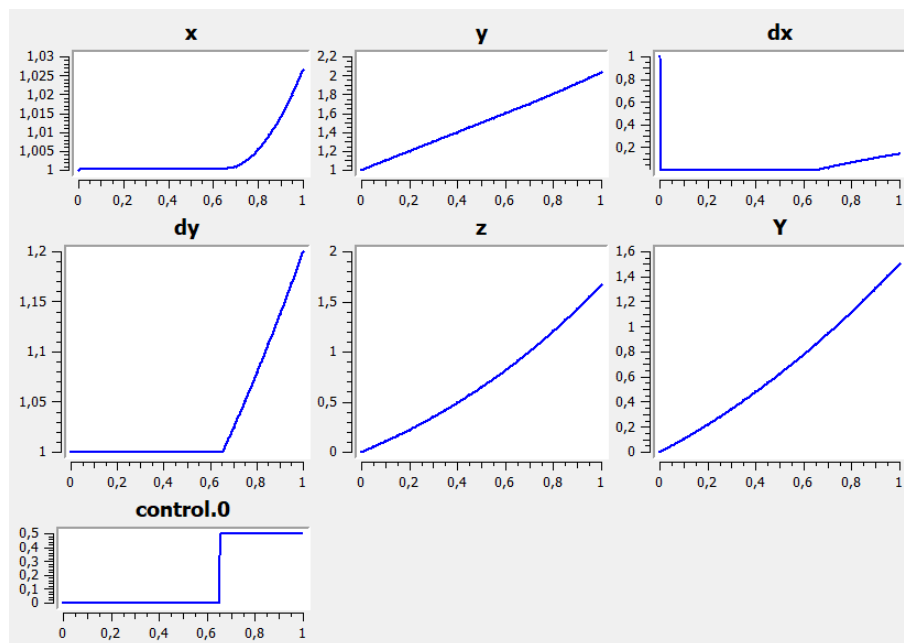


Figura 19: Resolución de BOCOP del problema modificado