EDA1

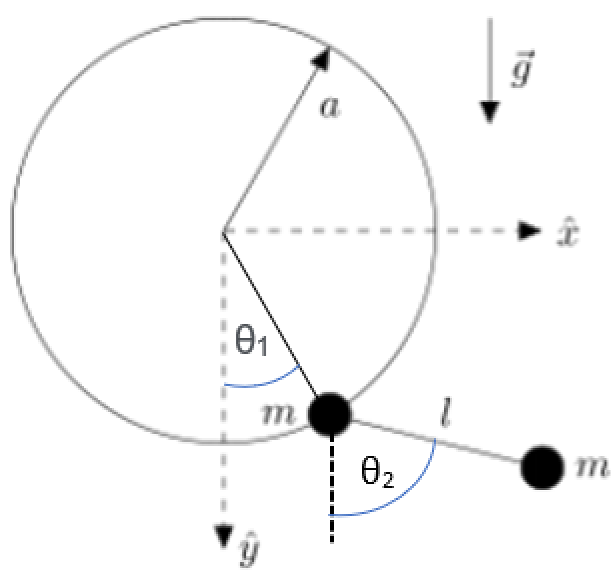
November 9, 2020

1 Simulación numérica | Evaluación de aprendizajes

Néstor Moscato
Mecánica General
Departamento de Ingeniería e Investigación Tecnológica
Universidad Nacional de La Matanza

1.1 Enunciado

Las partículas de masa m se encuentran unidas por una barra rígida de masa despreciable y longitud l. Una de ellas está enhebrada a un anillo de radio a. El movimiento de la barra se realiza siempre en el plano del anillo. El sistema se halla bajo el potencial gravitatorio de la superficie terrestre. El objetivo primario es determinar la fuerza que debe hacer la barra rígida



1. Determine un conjunto de coordenadas generalizadas que describan el estado del sistema y una función para el vínculo de interés recordando de no utilizarla para reducir el número de coordenadas.
2. Construya el correspondiente Lagrangiano.
3. Obtenga con las ecuaciones de Euler-Lagrange la fuerza de vínculo.

4. Genera una gráfica de la magnitud de la fuerza de vínculo en función del tiempo.

[1]:
```
# biblioteca de cálculo simbólico
import sympy as sym
import sympy.physics.mechanics as mech
import numpy as np # Biblioteca de cálculo numérico general
from sympy import sin, sqrt
from sympy.integrals import Integral
import matplotlib.pyplot as plt
mech.init_vprinting() # notación con puntos para derivadas temporales
```

2 Variables

[2]:
```
# Defino los parámetros del sistema
m, g, l, a, landa, landas = sym.symbols('m, g, l, a,  ,  _s', positive=True)

# Defino coordenadas generalizadas
t = sym.symbols('t') # tiempo
tita1 = sym.Function(' _1')(t)
tita2 = sym.Function(' _2')(t)
d = sym.Function('d')(t)
```

2.1 Vínculo

Siendo d el vector que define la distancia entre las masas, la condición de vínculo es

f = d - l = 0

3 Energía cinética

Posición de la masa m.

[3]:
```
# Sistema cartesiano
N = sym.physics.vector.ReferenceFrame('N') # marco referencial N en coordenadas␣
 ↪cartesianas

# Defino las posiciones de las masas
r_1 = a*sym.sin(tita1)*(N.x) + a*sym.cos(tita1)* (N.y)
r_2 = r_1 + d*sym.sin(tita2)*(N.x) + d*sym.cos(tita2)*(N.y)
r_1, r_2
```

[3]:

$$\left(a \sin\left(_1\right)\hat{\mathbf{n}}_{\mathbf{x}} + a\cos\left(_1\right)\hat{\mathbf{n}}_{\mathbf{y}},\ \ (a\sin\left(_1\right) + d\sin\left(_2\right))\hat{\mathbf{n}}_{\mathbf{x}} + (a\cos\left(_1\right) + d\cos\left(_2\right))\hat{\mathbf{n}}_{\mathbf{y}}\right)$$

[4]:
```
# Defino la velocidad
v_1 = r_1.diff(t,N) # r derivada respecto del tiempo, en el marco N
v_2 = r_2.diff(t,N)
v_1, v_2
```

[4]: $\left(a\cos\left(_{1}\right)\dot{}_{1}\hat{\mathbf{n}}_{\mathbf{x}} - a\sin\left(_{1}\right)\dot{}_{1}\hat{\mathbf{n}}_{\mathbf{y}},\ \ (a\cos\left(_{1}\right)\dot{}_{1} + d\cos\left(_{2}\right)\dot{}_{2} + \sin\left(_{2}\right)\dot{d})\hat{\mathbf{n}}_{\mathbf{x}} + (-a\sin\left(_{1}\right)\dot{}_{1} - d\sin\left(_{2}\right)\dot{}_{2} + \cos\left(_{2}\right)\dot{d})\hat{\mathbf{n}}_{\mathbf{y}}\right)$

[5]:
```
# Realizo el cuadrado de la velocidad
v1_cu = sym.physics.vector.dot(v_1, v_1)
v2_cu = sym.physics.vector.dot(v_2, v_2)
```

[6]:
```
# Energía cinética
T = 0.5* m* (v1_cu + v2_cu)
cin=sym.symbols('T')
sym.Eq(cin, T.simplify())
```

[6]: $T = 0.5m\left(2a^{2\dot{}2}_{\ 1} + 2ad\cos\left(_{1} - _{2}\right)\dot{}_{1}\dot{}_{2} - 2a\sin\left(_{1} - _{2}\right)\dot{d}_{1} + d^{2\dot{}2}_{\ 2} + \dot{d}^{2}\right)$

4 Energía potencial

[7]:
```
# Energía potencial
V_1 = - m*g*a*sym.cos(tita1)
V_2 = - m*g*(a*sym.cos(tita1)+d*sym.cos(tita2))
V = V_1 + V_2

pot = sym.symbols('V')
sym.Eq(pot, V.simplify())
```

[7]: $V = -gm\left(2a\cos\left(_{1}\right) + d\cos\left(_{2}\right)\right)$

5 Lagrangiano

[8]:
```
L = T-V
lan = sym.symbols('L')
sym.Eq(lan, L.simplify())
```

[8]: $L = m\left(1.0a^{2\dot{}2}_{\ 1} + 2.0ag\cos\left(_{1}\right) + 1.0ad\cos\left(_{1} - _{2}\right)\dot{}_{1}\dot{}_{2} - 1.0a\sin\left(_{1} - _{2}\right)\dot{d}_{1} + 1.0gd\cos\left(_{2}\right) + 0.5d^{2\dot{}2}_{\ 2} + 0.5\dot{d}^{2}\right)$

[9]:
```
f1 = (a*sym.cos(tita1)-(a*sym.cos(tita1)+d*sym.cos(tita2)))**2 + (a*sym.
↪sin(tita1)-(a*sym.sin(tita1)+d*sym.sin(tita2)))**2 - l**2
sym.Eq(sym.symbols('F'),f1)
```

[9]: $F = -l^{2} + d^{2}\sin^{2}\left(_{2}\right) + d^{2}\cos^{2}\left(_{2}\right)$

[10]:
```
f = l-d
```

6 Ecuaciones de Euler-Lagrange con multiplicadores

Tendremos dos, una para cada una de las coordenadas generalizadas q_i = $_1$, $_2$, d. Primero calculamos el lado izquierdo para cada coordenada

$$\frac{\partial}{\partial q_i}L$$

[11]:
```
ladoizq_tita1 = L.diff(tita1)
li1=sym.symbols('LadoIzq_ _1')
sym.Eq(li1,ladoizq_tita1.simplify())
```

[11]: $LadoIzq_1 = -am\left(2g\sin({}_1) + \left(d\sin({}_1 - {}_2)\dot{}_2 + \cos({}_1 - {}_2)\dot{d}\right)\dot{}_1\right)$

[12]:
```
ladoizq_tita2 = L.diff(tita2)
li2=sym.symbols('Ladoizq_ _2')
sym.Eq(li2,ladoizq_tita2.simplify())
```

[12]: $Ladoizq_2 = 1.0m\left(ad\sin({}_1 - {}_2)\dot{}_1\dot{}_2 + a\cos({}_1 - {}_2)\dot{d}_1 - gd\sin({}_2)\right)$

[13]:
```
ladoizq_d = L.diff(d)
lid=sym.symbols('Ladoizq_d')
sym.Eq(lid,ladoizq_d.simplify())
```

[13]: $Ladoizq_d = m\left(a\cos({}_1 - {}_2)\dot{}_1\dot{}_2 + g\cos({}_2) + d\dot{}_2^2\right)$

Luego calculamos el lado 2 para cada coordenada

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial}{\partial \dot{q}_i}L$$

[14]:
```
lado2_tita1 = L.diff(tita1.diff(t)).diff(t)
ld1=sym.symbols('Lado2_ _1')
sym.Eq(ld1,lado2_tita1.simplify())
```

[14]: $Lado_{21} = am\left(2.0\ddot{a}_1 - 1.0d\sin({}_1 - {}_2)\dot{}_1\dot{}_2 + 1.0d\sin({}_1 - {}_2)\dot{}_2^2 + 1.0d\cos({}_1 - {}_2)\ddot{}_2 - 1.0\sin({}_1 - {}_2)\ddot{d} - 1.0\cos({}_1 - {}_2)\dot{d}_1\right.$

[15]:
```
lado2_tita2 = L.diff(tita2.diff(t)).diff(t)
ld2=sym.symbols('Lado2_ _2')
sym.Eq(ld2,lado2_tita2.simplify())
```

[15]: $Lado_{22} = m\left(-1.0ad\sin({}_1 - {}_2)\dot{}_1^2 + 1.0ad\sin({}_1 - {}_2)\dot{}_1\dot{}_2 + 1.0ad\cos({}_1 - {}_2)\ddot{}_1 + 1.0a\cos({}_1 - {}_2)\dot{d}_1 + 1.0d^2\ddot{}_2 + 2.0d\dot{d}_2\right)$

[16]:
```
lado2_d = L.diff(d.diff(t)).diff(t)
lado2_d.simplify()
```

[16]: $1.0m\left(-a\sin({}_1 - {}_2)\ddot{}_1 - a\cos({}_1 - {}_2)\dot{}_1^2 + a\cos({}_1 - {}_2)\dot{}_1\dot{}_2 + \ddot{d}\right)$

Y escribimos las ecuaciones de Euler-Lagrange con multiplicadores para las coordenadas.

Para ${}_1$:

[17]:
```python
eulerlagrange_tita1 = ladoizq_tita1 - lado2_tita1 - landa*f.diff(tita1)

el_tita1= sym.Eq(eulerlagrange_tita1.simplify(), 0)
el_tita1
```

[17]:
$$am\left(-2.0a\ddot{}_1 - 2.0g\sin\left(_1\right) - 1.0d\sin\left(_1 - _2\right)\dot{}_2^2 - 1.0d\cos\left(_1 - _2\right)\ddot{}_2 + 1.0\sin\left(_1 - _2\right)\ddot{d} - 2.0\cos\left(_1 - _2\right)\dot{d}\dot{}_2\right) = 0$$

Para $_2$:

[18]:
```python
eulerlagrange_tita2 = ladoizq_tita2 - lado2_tita2 - landa*f.diff(tita2)

el_tita2=sym.Eq(eulerlagrange_tita2.simplify(), 0)
el_tita2
```

[18]:
$$m\left(1.0a\sin\left(_1 - _2\right)\dot{}_1^2 - 1.0a\cos\left(_1 - _2\right)\ddot{}_1 - 1.0g\sin\left(_2\right) - 1.0d\ddot{}_2 - 2.0\dot{d}\dot{}_2\right)d = 0$$

Para d:

[19]:
```python
eulerlagrange_d = ladoizq_d - lado2_d - landa*f.diff(d)
el_d=sym.Eq(eulerlagrange_d.simplify(),0)
el_d
```

[19]:
$$1.0am\sin\left(_1 - _2\right)\ddot{}_1 + 1.0am\cos\left(_1 - _2\right)\dot{}_1^2 + 1.0gm\cos\left(_2\right) + 1.0md\dot{}_2^2 - 1.0m\ddot{d} + 1.0 = 0$$

6.1 Fuerza de vínculo

Del sistema de ecuaciones de E-L hay que obtener el valor de .

[20]:
```python
landa1 = sym.solve(eulerlagrange_d,landa)
landa1
```

[20]:
$$\left[m\left(-a\sin\left(_1 - _2\right)\ddot{}_1 - a\cos\left(_1 - _2\right)\dot{}_1^2 - g\cos\left(_2\right) - d\dot{}_2^2 + \ddot{d}\right)\right]$$

Entonces el valor de la fuerza generalizada Q_l es:

[21]:
```python
Ql = ladoizq_d - lado2_d
q=sym.symbols('Q_l')
sym.Eq(q,Ql.simplify())
```

[21]:
$$Q_l = 1.0m\left(a\sin\left(_1 - _2\right)\ddot{}_1 + a\cos\left(_1 - _2\right)\dot{}_1^2 + g\cos\left(_2\right) + d\dot{}_2^2 - \ddot{d}\right)$$

Pero de la ecuación de vínculo sabemos que d = l , y de derivar el vínculo dos veces respecto del tiempo sabemos que $\ddot{}$ = 0, entonces:

[22]:
```python
qr= Ql.subs([(d,l),(f.diff(t,2),0)])
sym.Eq(q,qr.simplify())
```

[22]:
$$Q_l = 1.0m\left(a\sin\left(_1 - _2\right)\ddot{}_1 + a\cos\left(_1 - _2\right)\dot{}_1^2 + g\cos\left(_2\right) + l\dot{}_2^2\right)$$

6.2 Resolución algebráica

Resolvemos las ecuaciones obtenidas, sabiendo que:

[23]:
```
sym.Eq(d.diff(t,2),0)
```

[23]: $\ddot{d} = 0$

[24]:
```
sym.Eq(d.diff(t),0)
```

[24]: $\dot{d} = 0$

[25]:
```
t1= sym.solve(eulerlagrange_tita1, tita1.diff(t,2))
t11 = t1[0].subs([(d,l),(f.diff(t,2),0)])
t11.simplify()
```

[25]: $$\frac{-1.0g\sin\left(_1\right) - 0.5l\sin\left(_1 - _2\right)\dot{}_2^2 - 0.5l\cos\left(_1 - _2\right)\ddot{}_2}{a}$$

[26]:
```
t2= sym.solve(eulerlagrange_tita2, tita2.diff(t,2))
t22 = t2[0].subs([(d,l),(f.diff(t,2),0)])
t22.simplify()
```

[26]: $$\frac{a\sin\left(_1 - _2\right)\dot{}_1^2 - a\cos\left(_1 - _2\right)\ddot{}_1 - g\sin\left(_2\right)}{l}$$

$Theta_1$ punto punto es igual a

[46]:
```
t1s=t11.subs(tita2.diff(t,2),t22)
t1aux=t1s-tita1.diff(t,2) #Como t1s es theta 1 punto punto, pero el sistema lo␣
↪entiende igualado a cero, "paso restando" el theta 1 punto punto
t1pp=sym.solve(t1aux,tita1.diff(t,2))
t1pp[0]
```

[46]: $$\frac{a\sin\left(2.0_1 - 2.0_2\right)\dot{}_1^2 + g\sin\left(_1 - 2.0_2\right) + 3.0g\sin\left(_1\right) + 2.0l\sin\left(_1 - _2\right)\dot{}_2^2}{a\left(\cos\left(2.0_1 - 2.0_2\right) - 3.0\right)}$$

$Theta_2$ punto punto es igual a

[49]:
```
t2s=t22.subs(tita1.diff(t,2),t11)
t2aux=t2s-tita2.diff(t,2) #Como t2s es theta 2 punto punto, pero el sistema lo␣
↪entiende igualado a cero, "paso restando" el theta 2 punto punto
t2pp=sym.solve(t2aux,tita2.diff(t,2))
t2pp[0]
```

[49]: $$\frac{-2.0a\sin\left(_1 - _2\right)\dot{}_1^2 - g\sin\left(2.0_1 - _2\right) + g\sin\left(_2\right) - 0.5l\sin\left(2.0_1 - 2.0_2\right)\dot{}_2^2}{l\left(\cos^2\left(_1 - _2\right) - 2.0\right)}$$

[69]:
```
q_d1=qr.subs(tita1.diff(t,2),t1pp[0])
q_d=q_d1.subs(tita2.diff(t,2),t2pp[0])
q_d.simplify()
```

[69]:

$$1.0m\left(0.333333333333333a\sin\left(_1+2.0_2\right)\sin\left(_2\right)\cos\left(2.0_1\right)^2_1+0.333333333333333a\sin\left(_1\right)\sin\left(2.0_1\right)\cos\left(1.0_2\right)^2_1-0.33\right.$$

Resolución numérica

Damos valores para las variables involucradas

[145]:
```
m_v = 1 #kg
g_v = 9.81 #m/s2
l_v = 20 #m
a_v = 20 #m

valores = {
m : m_v,

g : g_v,
l : l_v,
a : a_v,
}
```

[146]:
```
tita1_v=t1pp[0].subs(valores)
```

[147]:
```
tita2_v=t2pp[0].subs(valores)
```

[148]:
```
tita1_num=sym.lambdify([tita1,tita1.diff(t),tita2,tita2.diff(t)],tita1_v)
tita2_num=sym.lambdify([tita1,tita1.diff(t),tita2,tita2.diff(t)],tita2_v)
```

[149]:
```
def derivaday(t, y):
    dydt = [y[1], tita1_num(y[0], y[1], y[2], y[3]), y[3], tita2_num(y[0],␣
 ↪y[1], y[2], y[3]) ]
    return dydt
```

6.3 Simulación numérica

[150]:
```
t_0=0 #s
t_f =20 #s
t_p = 0.01 #s

t_rango = np.arange(t_0, t_f, t_p)

# condiciones iniciales
theta1_inicial = np.pi/90 # 2 grados
theta2_inicial = np.pi/90 # 2 grados
theta1_p_inicial = 0 # parte del reposo
dtheta2_p_inicial = 0 # partel del reposo
y_inicial = [theta1_inicial, theta1_p_inicial, theta2_inicial,␣
 ↪dtheta2_p_inicial]
```

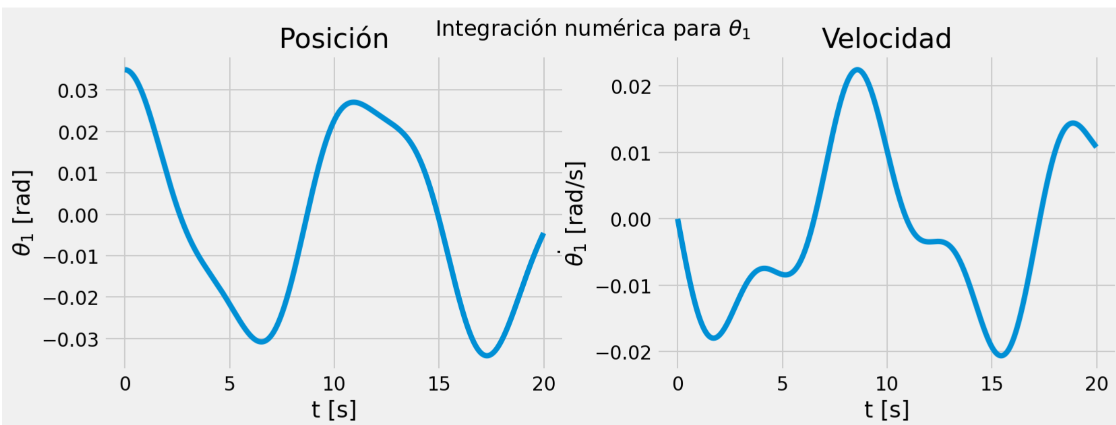
[151]:
```python
from scipy.integrate import solve_ivp
```

[152]:
```python
sol_y = solve_ivp(derivaday, (t_rango[0], t_rango[-1] ), y_inicial, t_eval =␣
↪t_rango)
```

[153]:
```python
plt.style.use('fivethirtyeight')
def graficaFuncion(solucion, nombreCoordenada='q'):
    fig, ax = plt.subplots(nrows= 1, ncols= 2, squeeze=False, figsize=(12, 4))
    fig.suptitle('Integración numérica para $'+ nombreCoordenada + '$',␣
↪fontsize=16)
    ax[0,0].plot(solucion.t, solucion.y[0]) # posición
    ax[0,0].set(xlabel='t [s]', ylabel= '$' + nombreCoordenada+ '$ [rad]',␣
↪title='Posición')
    ax[0,1].plot(solucion.t, solucion.y[1]) # velocidad
    ax[0,1].set(xlabel='t [s]', ylabel='$\dot{' + nombreCoordenada+ '}$ [rad/
↪s]', title='Velocidad')
```

[154]:
```python
graficaFuncion(sol_y, nombreCoordenada = r'\theta_1')
```

[154]:

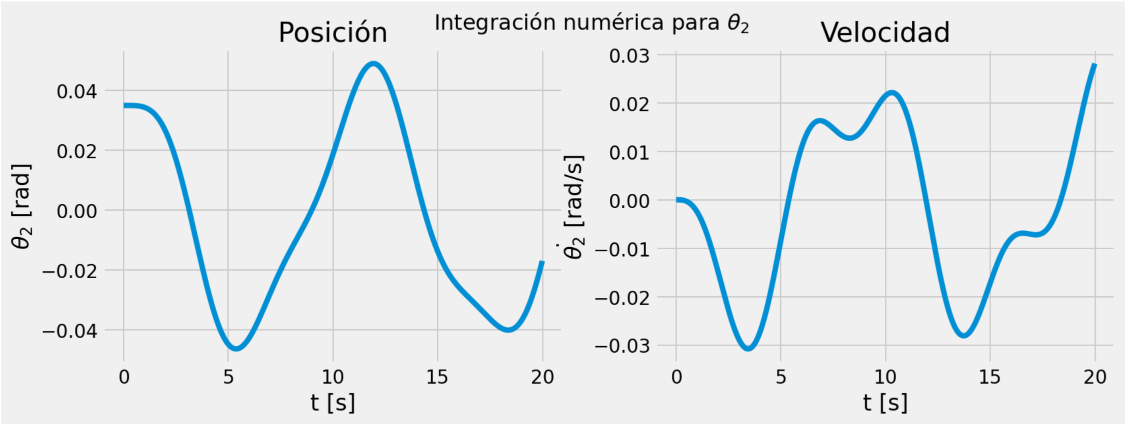


[155]:
```python
solucion = sol_y
nombreCoordenada = r'\theta_2'

fig, ax = plt.subplots(nrows= 1, ncols= 2, squeeze=False, figsize=(12, 4)) #␣
↪dos figuras en la misma fila
fig.suptitle('Integración numérica para $'+ nombreCoordenada + '$', fontsize=16)
ax[0,0].plot(solucion.t, solucion.y[2]) # posición
ax[0,0].set(xlabel='t [s]', ylabel= '$' + nombreCoordenada+ '$ [rad]',␣
↪title='Posición')
ax[0,1].plot(solucion.t, solucion.y[3]) # velocidad
ax[0,1].set(xlabel='t [s]', ylabel='$\dot{' + nombreCoordenada+ '}$ [rad/s]',␣
↪title='Velocidad')
```

[155]: [Text(0.5, 0, 't [s]'),
 Text(0, 0.5, '$\\dot{\\theta_2}$ [rad/s]'),
 Text(0.5, 1.0, 'Velocidad')]

[155]:



6.4 Fuerza de vínculo con datos simulados

[156]:
```
q_dfinal=q_d.subs(valores).simplify()
q_dfinal
```

[156]: $9.81\left(\cos^2\left(_1 - _2\right) - 2.0\right)\left(\cos\left(2.0_1 - 2.0_2\right) - 3.0\right)\cos\left(_2\right) + 1.0\left(\left(20\sin\left(_1\right)\dot{}_1^2 + 20\sin\left(_2\right)\dot{}_2^2\right)\left(\cos^2\left(_1 - _2\right) - 2.0\right)\left(\cos(2.0\right.\right.$

[157]:
```
qd_num = sym.lambdify([tita1, tita1.diff(t), tita2, tita2.diff(t)], q_dfinal)
```

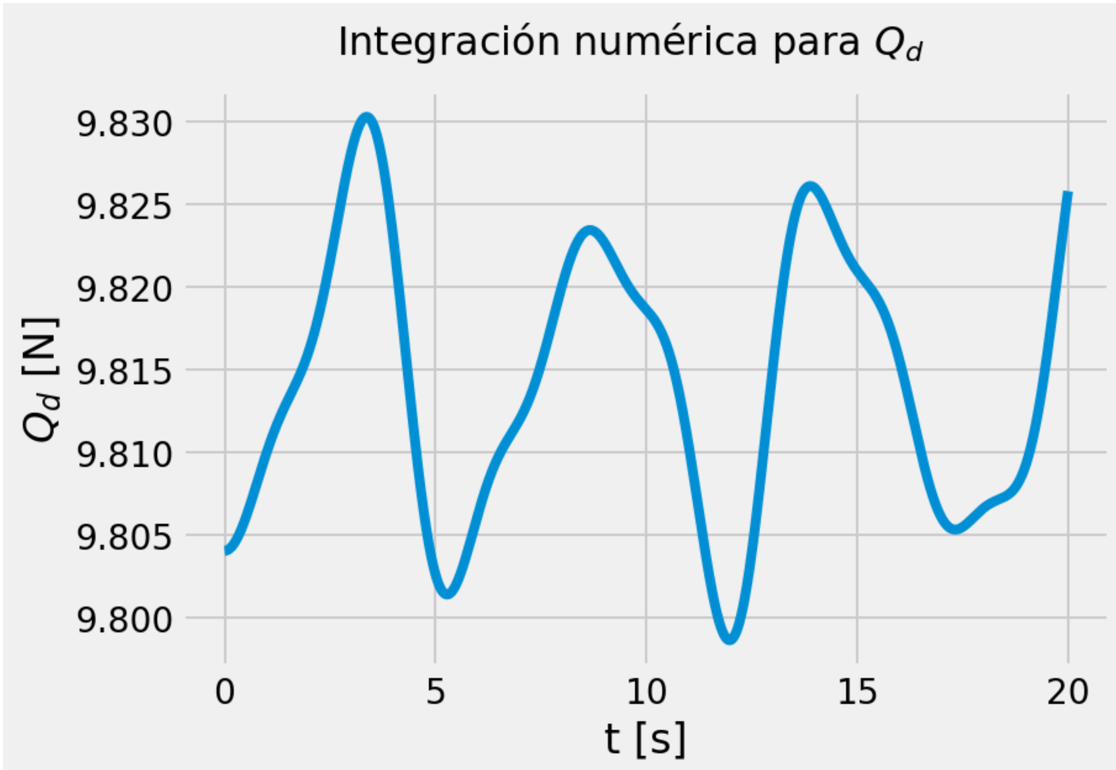
[158]:
```
qd_sim = qd_num(solucion.y[0], solucion.y[1], solucion.y[2], solucion.y[3])
```

[159]:
```
solucion = sol_y
nombreCoordenada = r'Q_d'

fig, ax = plt.subplots(nrows= 1, ncols= 1, squeeze=False, figsize=(6, 4)) # dos␣
↪figuras en la misma fila
fig.suptitle('Integración numérica para $'+ nombreCoordenada + '$', fontsize=16)
# ax[0,0].plot(solucion.t, d_Q_simulado) # posición
ax[0,0].plot(solucion.t, qd_sim) # posición
ax[0,0].set(xlabel='t [s]', ylabel= '$' + nombreCoordenada+ '$ [N]')
```

[159]: [Text(0.5, 0, 't [s]'), Text(0, 0.5, 'Q_d [N]')]

[159]:



6.5 Exploración del comportamiento

Se realizaron distintas combinaciones de los parámetros iniciales en busca de las que resulten en una gráfica más suave de la fuerza de vinculo Q_d, siendo la mostrada una de las mas suaves obtenidas. A continuación se dejan más ejemplos de lo realizado, con distintos parámetros. En los cuadros se muestran los valores que difieren del simulado en esta libreta, el resto de los parámetros tienen valores idénticos.

[ ]:

```

```