# caseE-linearization

October 4, 2024

# Linearization: Design Study

[ ]:
```python
# %%

import sympy
from sympy import *
from sympy.physics.vector.printing import vlatex
from IPython.display import Math, display

init_printing()

def dotprint(expr):
    display(Math(vlatex(expr)))
```

[ ]:
```python
# %%

t = symbols('t')
# Generalized coordinates
z, theta = symbols(r'z, \theta', cls=Function)
z = z(t)
theta = theta(t)

z_dot = z.diff(t)
theta_dot = theta.diff(t)

z_ddot = z.diff(t,2)
theta_ddot = theta.diff(t,2)

m1, m2, ell, g, F = symbols(r'm_1, m_2, \ell, g, F', real=True)
```

[ ]:
```python
# %%

# Equations of motion in state variable form
f_of_x_and_u = Matrix([
    z_dot,
    theta_dot,
    1/m1*(-m1*g*sin(theta) + m1*z*theta_dot**2),
```

```
    1/(m2*ell**2/3 + m1*z**2)*(ell*F*cos(theta) - 2*m1*z*z_dot*theta_dot -
 ↪m1*g*z*cos(theta) - m2*g*ell/2*cos(theta))
])

dotprint(f_of_x_and_u)
```

$$\left[\begin{matrix}\dot{z}\\ \dot{\theta}\\ \frac{- g m_{1} \sin{\left(\theta \right)} + m_{1} z \dot{\theta}^{2}}{m_{1}}\\ \frac{F \ell \cos{\left(\theta \right)} - \frac{\ell g m_{2} \cos{\left(\theta \right)}}{2} - g m_{1} z \cos{\left(\theta \right)} - 2 m_{1} z \dot{\theta} \dot{z}}{\frac{\ell^{2} m_{2}}{3} + m_{1} z^{2}}\end{matrix}\right]$$

## Deriving Nonlinear State Space Equations

[ ]:
```
# %%

state = Matrix([z, theta, z_dot, theta_dot])
dotprint(state)
```

$$\left[\begin{matrix}z\\ \theta\\ \dot{z}\\ \dot{\theta}\end{matrix}\right]$$

[ ]:
```
# %%

state_deriv = f_of_x_and_u
dotprint(state_deriv)
```

$$\left[\begin{matrix}\dot{z}\\ \dot{\theta}\\ \frac{- g m_{1} \sin{\left(\theta \right)} + m_{1} z \dot{\theta}^{2}}{m_{1}}\\ \frac{F \ell \cos{\left(\theta \right)} - \frac{\ell g m_{2} \cos{\left(\theta \right)}}{2} - g m_{1} z \cos{\left(\theta \right)} - 2 m_{1} z \dot{\theta} \dot{z}}{\frac{\ell^{2} m_{2}}{3} + m_{1} z^{2}}\end{matrix}\right]$$

## Linearization

First, we need to find an equilibrium point. We can do this by setting all derivatives to zero and solving.

[ ]:
```
# %%

equilibrium_equation = state_deriv.subs({z_dot: 0, theta_dot: 0, theta: 0})

eq_solve_dict = solve(equilibrium_equation, (z, F), simplify=True, dict=True)[0]
dotprint(eq_solve_dict)

# Define symbols
m1, m2, ell, g, F, z, ze = symbols('m_1 m_2 ell g F z ze')
```

```
# Original equation (solution from eq_solve_dict)
# z = (ell * (2F - g*m2)) / (2*g*m1)
equation = Eq(z, (ell * (2*F - g*m2)) / (2*g*m1))
```

$$\left\{z : \frac{\ell\left(2F - gm_2\right)}{2gm_1}\right\}$$

[ ]:
```
# %%

# Solve for F
u_eq = solve(equation, F)[0]
print("u_eq = ")
dotprint(u_eq.subs(z,ze))
```

u_eq =

$$\frac{gm_2}{2} + \frac{gm_1 ze}{\ell}$$

---

The solution for $u_{eq}$ (or $\mathbf{F}$) comes from solving the equation above. The equilibrium point is the value of $\mathbf{F}$ at which the system's derivatives (velocities and accelerations) are zero, meaning the system is at rest.

Here, $z_e$ is the equilibrium position. This expression shows that the equilibrium force depends on both the gravitational forces acting on the masses $m_1$ and $m_2$, as well as the equilibrium position $z_e$ scaled by the length $\ell$. We can see that at equilibrium, $z$ can be any value (which we'll call $z_e$), and $F$ depends on this $z_e$.

[ ]:
```
# %%

z_e = symbols('z_e')
u_eq = m1*g/ell*z_e + m2*g/2
theta_eq = 0
z_dot_eq = 0
theta_dot_eq = 0
```

#### Define A, B Jacobians

We can use Sympy's `jacobian` function to find the jacobians of `f(x,u)`.

First we find $A = \frac{\partial f}{\partial x}$:

[ ]:
```
# %%

A = f_of_x_and_u.jacobian(state)
dotprint(A)
```

$$\left[\begin{matrix}0 & 0 & 1 & \\ 0 & 0 & 0 & \\ \dot{\theta}^{2} & - g \cos{\left(\theta \right)} & 0 & \\ - \frac{2 m_{1} \left(F \ell \cos{\left(\theta \right)} - \frac{\ell g m_{2} \cos{\left(\theta \right)}}{2} - g m_{1} z \cos{\left(\theta \right)} - 2 m_{1} z \dot{\theta} \dot{z}\right) z}{\left(\frac{\ell^{2} m_{2}}{3} + m_{1} z^{2}\right)^{2}} + \frac{- g m_{1} \cos{\left(\theta \right)} - 2 m_{1} \dot{\theta} \dot{z}}{\frac{\ell^{2} m_{2}}{3} + m_{1} z^{2}} & \frac{- F \ell \sin{\left(\theta \right)} + \frac{\ell g m_{2} \sin{\left(\theta \right)}}{2} + g m_{1} z \sin{\left(\theta \right)}}{\frac{\ell^{2} m_{2}}{3} + m_{1} z^{2}} & - \frac{2 m_{1} z \dot{\theta}}{\frac{\ell^{2} m_{2}}{3} + m_{1} z^{2}} & - \end{matrix}\right.$$

[ ]:
```python
# %%

A_subs = {
    z: z_e,
    theta: theta_eq,
    z_dot: z_dot_eq,
    theta_dot: theta_dot_eq,
    F: u_eq
}

A_eq = A.subs(A_subs)
dotprint(A_eq)
```

$$\left[\begin{matrix}0 & 0 & 1 & 0\\0 & 0 & 0 & 1\\0 & - g & 0 & 0\\- \frac{g m_{1}}{\frac{\ell^{2} m_{2}}{3} + m_{1} z^{2}} - \frac{2 m_{1} \left(F \ell - \frac{\ell g m_{2}}{2} - g m_{1} z\right) z}{\left(\frac{\ell^{2} m_{2}}{3} + m_{1} z^{2}\right)^{2}} & 0 & 0 & 0\end{matrix}\right]$$

Now we do a similar process to find $B = \frac{\partial f}{\partial u}$

[ ]:
```python
# %%

B = f_of_x_and_u.jacobian(Matrix([F]))
dotprint(B)
```

$$\left[\begin{matrix}0\\0\\0\\0\end{matrix}\right]$$

[ ]:
```python
# %%

B_subs = {
    z: z_e,
    theta: theta_eq,
    z_dot: z_dot_eq,
    theta_dot: theta_dot_eq,
    F: u_eq
}

B_eq = B.subs(B_subs)
```

```
dotprint(B_eq)
```

$$\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$$

### Transfer Function

We can also transform this to a transfer function if we define C and D matrices

[ ]:
```
# %%

C = Matrix([[0, 1, 0, 0], [0, 0, 0, 1.0]])
D = Matrix([[0], [0]])

s = symbols('s')
transfer_func = simplify(C * (s*eye(4) - A_eq).inv() * B_eq + D)
dotprint(transfer_func)
```

$$\begin{bmatrix}0\\0\end{bmatrix}$$

### Simplifying Assumption

Now setting the $m_1 g$ term equal to zero as described in the problem:

[ ]:
```
# %%

A_simplified = A_eq.subs(m1*g, 0)
B_simplified = B_eq.subs(m1*g, 0)

transfer_func_simplified = simplify(C * (s*eye(4) - A_simplified).inv() *
  B_simplified + D)
dotprint(transfer_func_simplified)
```

$$\begin{bmatrix}0\\0\end{bmatrix}$$