ELG7113 Machine Learning for Adaptive and Intelligent Control Systems

Student: Derek Boase

Std Num: 300043860

e-mail: dboas065@uottawa.ca

assignment GitHub: git@github.com:derekboase/Adaptive_Control_Code.git

In [1]:
```
import matplotlib.pyplot as plt
import control as co
import numpy as np
import sympy as sp

from sympy.abc import e, p, alpha, gamma, omega, theta, zeta
from scipy.integrate import odeint
from numpy import cos, sin, pi
```

Question 1: MIT Rule

In [2]:
```
y, ym, ydot, yddot, u, uc = sp.symbols('y,y_m,\dot{y},\ddot{y},u,u_c')
theta_1, theta_2 = sp.symbols('theta_1,theta_2')
a,b,am,bm = sp.symbols('a,b,a_m,b_m')

model = sp.Eq(yddot, -a*ydot + b*u)
u_eq = theta_1*(y - uc) - theta_2*ydot
model_u_sub = sp.Eq(sp.solve(sp.expand(model.subs(u, u_eq)), b*theta_1*uc)[0], b*theta_1*uc)
model_p = sp.Eq((-p**2 - p*(a+b*theta_2) + b*theta_1)*y, b*theta_1*uc)


Gm = omega**2/(p**2 + 2*zeta*omega*p + omega**2)
ym_eq = sp.Eq(ym, Gm*uc)
```

Q1.1: Derivation of MIT Rule Adaptation Laws

The model of the system is given by,

$\ddot{y} = -\dot{y}a + bu$

where y is the output and u is the input. The adjustment law of the controller has two parameters, θ_1 and θ_2, such that the input to the system is given as,

$u = -\dot{y}\theta_2 + \theta_1\left(-u_c + y\right)$

Substituting the input equation into the model equation gives,

$-\ddot{y} - \dot{y}a - \dot{y}b\theta_2 + b\theta_1 y = b\theta_1 u_c$

To solve for $y(t)$ the differential operator is used such that $p = \frac{d}{dt}$. With this substitution, the model is then given as,

$y\left(b\theta_1 - p^2 - p\left(a + b\theta_2\right)\right) = b\theta_1 u_c$

In [3]:
```
model_p = sp.Eq(y, sp.collect(sp.solve(model_p, y)[0], p))
```

This gives,

$$y = -\frac{b\theta_1 u_c}{-b\theta_1 + p^2 + p\left(a + b\theta_2\right)}$$

To find the values of θ_1 and θ_2 for perfect model following, we consider the given model,

$$y_m = \frac{\omega^2 u_c}{\omega^2 + 2\omega p\zeta + p^2}$$

By comparing the numerators of the model and the reference model, we find θ_1 to be,

In [4]:
```
num, denum = sp.fraction(model_p.rhs)
num_m, denum_m = sp.fraction(ym_eq.rhs)
theta_1_MF = sp.Eq(theta_1, sp.solve(num - num_m, theta_1)[0])
theta_2_MF = sp.Eq(theta_2, sp.solve((denum - denum_m).subs(theta_1, theta_1_MF.rhs), theta_2)[0])
```

Then for perfect model following the values of the parameters should be,

$$\theta_1 = -\frac{\omega^2}{b}$$

and

$$\theta_2 = \frac{-a + 2\omega\zeta}{b}$$

Next we find the sensitivity of the error function, $\frac{\partial e(\theta_1, \theta_2)}{\partial \theta_i}$, for $i \in \{1, 2\}$. One note to make is that since y_m doesn't depend on the parameters the sensitivities are given by the derivation of the output, $\frac{\partial y(\theta_1, \theta_2)}{\partial \theta_i}$

In [5]:
```
sens_1, sens_2 = sp.symbols('de/dt_1, de/dt_2')
sensitivity_1 = sp.diff(model_p.rhs, theta_1)
sensitivity_1 = sp.Eq(sens_1, sp.simplify(sensitivity_1))
sensitivity_2 = sp.diff(model_p.rhs, theta_2)
sensitivity_2 = sp.Eq(sens_2, sp.simplify(sensitivity_2))
```

The evaluation of these derivatives yeilds,

$$de/dt_1 = -\frac{bpu_c\left(a + b\theta_2 + p\right)}{\left(-b\theta_1 + p^2 + p\left(a + b\theta_2\right)\right)^2}$$

$$de/dt_2 = \frac{b^2 p\theta_1 u_c}{\left(-b\theta_1 + p^2 + p\left(a + b\theta_2\right)\right)^2}$$

In [6]:
```
sensitivity_1 = sensitivity_1.subs(theta_1, theta_1_MF.rhs).subs(theta_2, theta_2_MF.rhs)
sensitivity_2 = sensitivity_2.subs(theta_1, theta_1_MF.rhs).subs(theta_2, theta_2_MF.rhs)
```

These are dependent on the estimated parameters and thus require substitution of the perfect model following parameters,

$$de/dt_1 = -\frac{bpu_c(2\omega\zeta + p)}{\left(\omega^2 + 2\omega p\zeta + p^2\right)^2}$$

$$de/dt_2 = -\frac{b\omega^2 pu_c}{\left(\omega^2 + 2\omega p\zeta + p^2\right)^2}$$

In [7]:
```
uc_sub = sp.solve(ym_eq, uc)[0]
sensitivity_1 = sensitivity_1.subs(theta_1, theta_1_MF.rhs).subs(theta_2, theta_2_MF.rhs).subs(uc, uc_sub)
sensitivity_2 = sensitivity_2.subs(theta_1, theta_1_MF.rhs).subs(theta_2, theta_2_MF.rhs).subs(uc, uc_sub)
```

By rearranging the equation for the reference model to solve for u_c we get,

$$u_c = \frac{y_m\left(\omega^2 + 2\omega p\zeta + p^2\right)}{\omega^2}$$

Substituting this into the sensitvity equations,

$$de/dt_1 = -\frac{bpy_m(2\omega\zeta + p)}{\omega^2\left(\omega^2 + 2\omega p\zeta + p^2\right)}$$

$$de/dt_2 = -\frac{bpy_m}{\omega^2 + 2\omega p\zeta + p^2}$$

Then,

$$\dot{\theta}_i = -\gamma' e\frac{\partial e}{\partial \theta_i}$$

Then the adaptation law's are given as,

$$\dot{\theta}_1 = \gamma' e\frac{bpy_m(2\omega\zeta + p)}{\omega^2\left(\omega^2 + 2\omega p\zeta + p^2\right)}$$

$$\dot{\theta}_2 = \gamma' e\frac{bpy_m}{\omega^2 + 2\omega p\zeta + p^2}$$

By defining $\gamma = \gamma' b$ we get

$$\dot{\theta}_1 = \gamma e\frac{py_m(2\omega\zeta + p)}{\omega^2\left(\omega^2 + 2\omega p\zeta + p^2\right)}$$

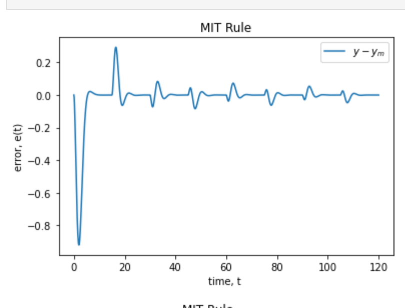
$$\dot{\theta}_2 = \gamma e\frac{py_m}{\omega^2 + 2\omega p\zeta + p^2}$$

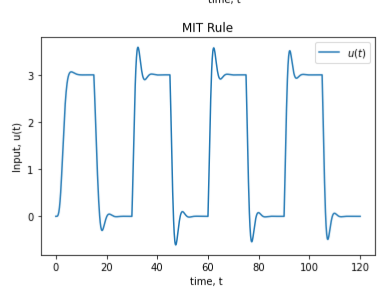
Q1.2: Implementation of MIT Rule Adaptation Laws

In [8]:
```
t = np.linspace(0, 100, 1001)
K = 2.0*1.5*0.6
C = 1.5**2
def reference_signal(time=t):
    _uc = np.array([1])
    for _t in time[1:]:
        rat = 2.0*pi*_t/30.0
        _uc = np.concatenate((_uc, np.array([sin(rat)>= 0])), axis=0)
    return _uc
uc=reference_signal()
```

In [9]:
```
t = np.linspace(0, 120, 1001)
K = 2.0*1.5*0.6
C = 1.5**2
def reference_signal(time=t):
    _uc = np.array([1])
    for _t in time[1:]:
        rat = 2.0*pi*_t/30.0
        _uc = np.concatenate((_uc, np.array([sin(rat)>= 0])), axis=0)
    return _uc
uc=reference_signal()
```

In [10]:
```
def parameters(init, t):
    global C, K  # C = omega^2, K = 2*omega*zeta
    gamma = 5

    # Initial conditions
    ym, ym_prime = init[0], init[1]
    y, y_prime = init[2], init[3]
    theta1, theta1_p, theta1_pp = init[4], init[5], init[6]
    theta2, theta2_p, theta2_pp = init[7], init[8], init[9]
    u = init[10]

    uc = float(sin(2.0*pi*t/30.0) >= 0)

    # ym calculation
    dym_dt = ym_prime
    d2ym_dt2 = -K*dym_dt - C*ym + C*uc

    # y calculation
    dy_dt = y_prime
    d2y_dt2 = -(3 + theta2)*dy_dt + theta1*(y - uc)
    e = dy_dt-dym_dt   # WORKING

    # Theta1 calculation
    M = gamma*e*(K*dym_dt + d2ym_dt2)
    dth1_dt = theta1_p
    d2th1_dt2 = theta1_pp
    d3th1_dt3 = -K*d2th1_dt2 - C*dth1_dt + M/C

    # Theta2 calculation
    N = gamma*e*dym_dt
    dth2_dt = theta2_p
    d2th2_dt2 = theta2_pp
    d3th2_dt3 = -K*d2th2_dt2 - C*dth2_dt + N

    # Calculation of u
    u = theta1*(y - uc) - theta2*dy_dt

    return [dym_dt, d2ym_dt2, dy_dt, d2y_dt2, dth1_dt, d2th1_dt2, d3th1_dt3, dth2_dt, d2th2_dt2, d3th2_dt3, u]

init = np.zeros(11,)
params_sol = odeint(parameters, init, t)
# plt.plot(t, params_sol[:, 0])
# plt.plot(t, params_sol[:, 2], '--')
plt.plot(t, params_sol[:,2] - params_sol[:,0])
plt.legend(['$y - y_m$'])
plt.ylabel('error, e(t)')
plt.xlabel('time, t ')
plt.title('MIT Rule')
plt.show()

plt.plot(t, params_sol[:,10])
plt.legend(['$u(t)$'])
plt.ylabel('Input, u(t)')
plt.xlabel('time, t ')
plt.title('MIT Rule')
plt.show()
```



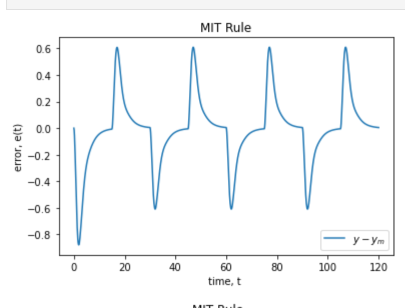


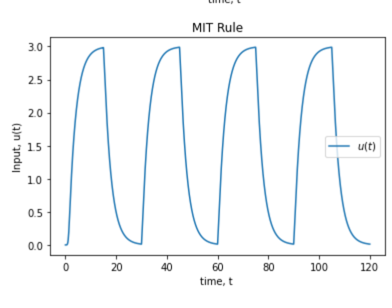
Q1.2: Implementation of normalized-MIT Rule Adaptation Laws

Note that for the normalized MIT rule the adaptation rate becomes,

$$\dot{\theta}_i = \frac{\gamma\Psi_i e}{\alpha + \Psi_i^T\Psi_i}$$

In [11]:
```
t = np.linspace(0, 120, 1001)

def parameters(init, t):
    global C, K
    alpha = 0.5
    gamma = 3.0

    # Initial conditions
    ym, ym_prime = init[0], init[1]
    y, y_prime = init[2], init[3]
    sens1, sens1_p, sens1_pp = init[4], init[5], init[6]
    sens2, sens2_p, sens2_pp = init[7], init[8], init[9]
    theta1, theta2 = init[10], init[11]
    u = init[12]

    uc = float(sin(2.0*pi*t/30.0) >= 0)

    # ym calculation
    dym_dt = ym_prime
    d2ym_dt2 = -K*dym_dt - C*ym + C*uc

    # y calculation
    dy_dt = y_prime
    d2y_dt2 = -(3 + theta2)*dy_dt + theta1*(y - uc)
    e = dy_dt-dym_dt

    # Sensitivity1 calculation
    M = -(K*dym_dt + d2ym_dt2)
    ds1_dt = sens1_p
    d2s1_dt2 = sens1_pp
    d3s1_dt3 = K*d2s1_dt2 + C*ds1_dt - M/C

    # Sensitivity2 calculation
    N = -dym_dt
    ds2_dt = sens2_p
    d2s2_dt2 = sens2_pp
    d3s2_dt3 = K*d2s2_dt2 + C*ds2_dt - N

    dth1_dt = gamma*sens1*e/(alpha + sens1**2)
    dth2_dt = gamma*sens2*e/(alpha + sens2**2)

    # Calculation of u
    u = theta1*(y - uc) - theta2*dy_dt

    return [dym_dt, d2ym_dt2, dy_dt, d2y_dt2, ds1_dt, d2s1_dt2, d3s1_dt3, ds2_dt, d2s2_dt2, d3s2_dt3, dth1_dt,

init = np.zeros(13,)
params_sol = odeint(parameters, init, t)
plt.plot(t, params_sol[:,2] - params_sol[:,0])
plt.legend(['$y - y_m$'])
plt.ylabel('error, e(t)')
plt.xlabel('time, t ')
plt.title('MIT Rule')
plt.show()

plt.plot(t, params_sol[:,-1])
plt.legend(['$u(t)$'])
plt.ylabel('Input, u(t)')
plt.xlabel('time, t ')
plt.title('MIT Rule')
plt.show()
```





Question 2: Lyapunov Method

In [12]:
```
import matplotlib.pyplot as plt
import control as co
import numpy as np
import sympy as sp

from sympy.abc import a, b, e, p, s, alpha, beta, gamma, omega, theta, zeta
from scipy.integrate import odeint
from numpy import cos, sin, pi
```

Q2.1: Lyapunov Validataion and Adaptation Rules

In [13]:
```
Ve_eq = sp.Symbol('V(e)')
dVe_sym = sp.Symbol('dV/dt')
t = sp.Symbol('t')
theta_1 = sp.Function('theta_1')(t)
theta_2 = sp.Function('theta_2')(t)
ym = sp.Function('y_m')(t)
y = sp.Function('y')(t)
uc = sp.Function('u_c')(t)
u = sp.Function('u')(t)
e = sp.Function('e')(t)


G_p = b/p
Gm_s = beta/(s+alpha)
u = theta_1*uc - theta_2*y
V_e = 1/2*(e**2 + 1/(b*gamma)*(alpha-b*theta_2)**2 + 1/(b*gamma)*(beta-b*theta_1)**2)
dVe_eq = sp.Eq(dVe_sym, sp.nsimplify(sp.diff(V_e, t)))

# Model following parameters
theta_1_MF = sp.solve(beta - b*theta_1, theta_1)[0]
theta_2_MF = sp.solve(alpha - b*theta_2, theta_2)[0]

e_t_dot = sp.simplify(sp.expand(b*(-theta_2*y + theta_1*uc) - (-alpha*ym + beta*uc)))
e_t_dot_subs = e_t_dot.subs(ym, y) - alpha*e
e_t_seperated = sp.collect(sp.collect(e_t_dot_subs, uc), y)
dVe_simp = dVe_eq.subs([(sp.diff(e, t), e_t_dot_subs), (theta_1, theta_1_MF), (theta_2, theta_2_MF)])
dVe_adapt = dVe_eq.subs(sp.diff(e, t), e_t_seperated)
```

To verify that the given function is a Lyapunov fucntion, we must show that,

1. The function, $V(e)$ is positive definite
2. The derivative of the function $\frac{dV(e)}{dt}$ is negative semi-definite
3. The function is zero at the equilibrium point $V(e = 0) = 0$

These conditions imply that the "energy" of the function is positive non-zero away from the equilibrium point and that as the parameters converge to the true values, the energy decreases. The last restriction ensures that the energy at the stability point is 0

These restrictions are valid for quadratic, asymptotically stable Lyapunov functions which it is suspected this is.

To satisfy the first criteria, consider the equaion for the proposed Lyapunov function,

$$V(e) = 0.5e^2(t) + \frac{0.5\left(\alpha - b\theta_2(t)\right)^2}{b\gamma} + \frac{0.5\left(-b\theta_1(t) + \beta\right)^2}{b\gamma}$$

where $b, \gamma > 0$. Given that the squared terms will always evaluate to positive values and that the denominator is always positive, then for any e the function is positive, satisfying the first stipulation is statisfied.

To investigate the second requirement, we take the derivative of the proposed Lyapunov function, noting that e, θ_1 and θ_2 are functions of time. Then the derivative is,

$$dV/dt = e(t)\frac{d}{dt}e(t) - \frac{\left(\alpha - b\theta_2(t)\right)\frac{d}{dt}\theta_2(t)}{\gamma} - \frac{\left(-b\theta_1(t) + \beta\right)\frac{d}{dt}\theta_1(t)}{\gamma}$$

To evaluate this derivative we need a closed form expression for the $\frac{d}{dt}e(t)$. For this, consider

$\dot{e}(t) = \dot{y}(t) - \dot{y}_m(t)$

From the given model functions, we get that

$\dot{y}(t) = -b\theta_2(t)y(t) + b\theta_1(t)u_c(t)$

$\dot{y}_m(t) = -\alpha y_m(t) + \beta u_c(t)$

Then,

$\frac{d}{dt}y(t) - \frac{d}{dt}y_m(t) = \alpha y_m(t) + b\theta_1(t)u_c(t) - b\theta_2(t)y(t) - \beta u_c(t)$

By adding and subtracting $\alpha y(t)$ and notine that $\alpha e(t) = \alpha\left(y(t) - y_m(t)\right)$ we get,

$\frac{d}{dt}e(t) = -\alpha e(t) + \alpha y(t) + b\theta_1(t)u_c(t) - b\theta_2(t)y(t) - \beta u_c(t)$

By substituting this result into the equation for the time derivative of $V(e)$ and substituting the model following parameters for $\theta_1 = \frac{\beta}{b}$ and $\theta_2 = \frac{\alpha}{b}$ we get,

$dV/dt = -\alpha e^2(t)$

Clearly, given the fact that $\alpha > 0$ and that $e^2(t) > 0\ \forall\ t$ then the second criteria is satisfied.

To meet the third criteria when θ_1 and θ_2 convervge to the proper values and the function is at steady state, (i.e. $e = 0$) then the fucntion is trivially 0. The third criteria is therefore met.

To find the adaptation laws of the coefficients we consider the equation,

$$dV/dt = \left(-\alpha e(t) + \left(\alpha - b\theta_2(t)\right)y(t) + \left(b\theta_1(t) - \beta\right)u_c(t)\right)e(t) - \frac{\left(\alpha - b\theta_2(t)\right)\frac{d}{dt}\theta_2(t)}{\gamma} - \frac{\left(-b\theta_1(t) + \beta\right)\frac{d}{dt}\theta_1(t)}{\gamma}$$

Combining the terms for θ_1 and it's derivatives gives,

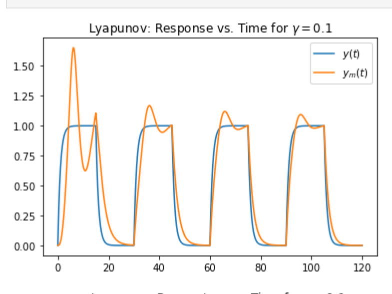
$\frac{d}{dt}\theta_1(t) = -e\gamma u_c(t)$

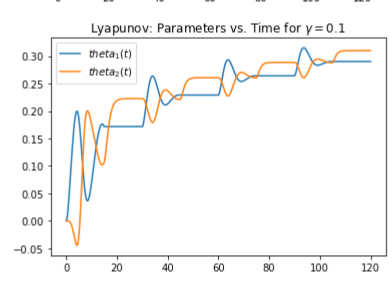
Likewise for θ_2,

$\frac{d}{dt}\theta_2(t) = e\gamma y(t)$

Q2.2: Implementation of Lypunov Methods with $\gamma = 0.1$

In [14]:
```
t = np.linspace(0, 120, 1001)

def lyapunov(init, t, gamma):
    b = 2
    alpha, beta = 1, 1

    # Initial conditions
    ym = init[0]
    y = init[1]
    theta1, theta2 = init[2], init[3]

    uc = float(sin(2.0*pi*t/30.0) >= 0)

    # ym calculation
    dym_dt = -alpha*ym + beta*uc

    # y calculation
    dy_dt = b*(-theta2*y + theta1*uc)
    e = y-ym

    # Theta1 calculation
    dth1_dt = -e*gamma*uc

    # Theta1 calculation
    dth2_dt = e*gamma*y

    return [dym_dt, dy_dt, dth1_dt, dth2_dt]

init = np.zeros(4,)
params_sol = odeint(lyapunov, init, t, args=(0.1,))

plt.plot(t, params_sol[:, 0])
plt.plot(t, params_sol[:, 1])
plt.title('Lyapunov: Response vs. Time for $\gamma = 0.1$')
plt.legend(['$y(t)$', '$y_{m}(t)$'])
plt.show()

plt.plot(t, params_sol[:, 2])
plt.plot(t, params_sol[:, 3])
plt.title('Lyapunov: Parameters vs. Time for $\gamma = 0.1$')
plt.legend(['$theta_1(t)$', '$theta_{2}(t)$'])
plt.show()
```





Q2.2: Implementation of Lypunov Methods with $\gamma = 0.5$

In [15]:
```
params_sol = odeint(lyapunov, init, t, args=(0.5,))

plt.plot(t, params_sol[:, 0])
plt.plot(t, params_sol[:, 1])
plt.title('Lyapunov: Response vs. Time for $\gamma = 0.5$')
plt.legend(['$y(t)$', '$y_{m}(t)$'])
plt.show()

plt.plot(t, params_sol[:, 2])
plt.plot(t, params_sol[:, 3])
plt.title('Lyapunov: Parameters vs. Time for $\gamma = 0.5$')
plt.legend(['$theta_1(t)$', '$theta_{2}(t)$'])
plt.show()
```