6.2 Maximizing Concentration of an Intermediate in a Batch Reactor

In [0]:

```
!pip install -q pyomo
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64

ipopt_executable = '/content/ipopt'
```

Mathematical Model

A material balance for an isothermal stirred batch reactor with a volume $V = 40$ liters and an initial concentration $C_{A,f}$ is given by

$$\begin{aligned} V\frac{dC_A}{dt} & = -Vk_AC_A \\ V\frac{dC_B}{dt} & = Vk_AC_A - Vk_BC_B \end{aligned}$$

Eliminating the common factor V

$$\begin{aligned} \frac{dC_A}{dt} & = -k_AC_A \\ \frac{dC_B}{dt} & = k_AC_A - k_BC_B \end{aligned}$$

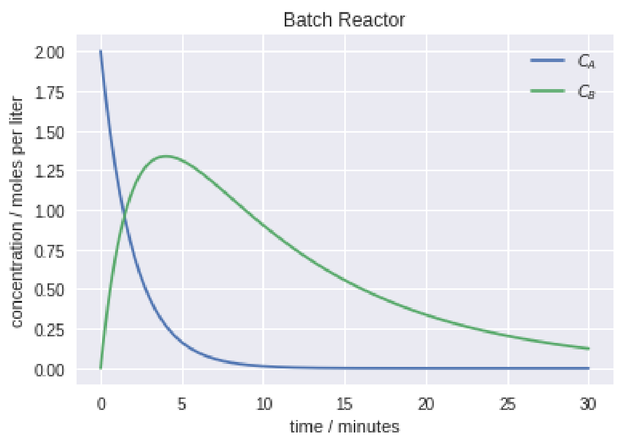
With an initial concentration $C_{A,f}$. A numerical solution to these equations is shown in the following cell.

In [3]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

V = 40      # liters
kA = 0.5    # 1/min
kB = 0.1    # 1/min
CAf = 2.0   # moles/liter

def batch(X, t):
    CA, CB = X
    dCA_dt = -kA*CA
    dCB_dt = kA*CA - kB*CB
    return [dCA_dt, dCB_dt]

t = np.linspace(0,30,200)
soln = odeint(batch, [CAf,0], t)
plt.plot(t, soln)
plt.xlabel('time / minutes')
plt.ylabel('concentration / moles per liter')
plt.title('Batch Reactor')
plt.legend(['$C_A$','$C_B$'])
```

Out[3]:

```
<matplotlib.legend.Legend at 0x7f57786aceb8>
```



Optimization with `scipy.minimize_scalar`

To find the maximum value, we first write a function to compute C_B for any value of time t.

In [0]:

```
def CB(tf):
    soln = odeint(batch, [CAf, 0], [0, tf])
    return soln[-1][1]
```

We gain use `minimize_scalar` to find the value of t that minimizes the negative value of $C_B(t)$.|

In [5]:

```
from scipy.optimize import minimize_scalar
minimize_scalar(lambda t: -CB(t), bracket=[0,50])
```

Out[5]:

```
     fun: -1.3374806339222158
    nfev: 23
     nit: 19
 success: True
       x: 4.023594924340666
```

In [6]:

```
tmax = minimize_scalar(lambda t: -CB(t), bracket=[0,50]).x

print('Concentration c_B has maximum', CB(tmax), 'moles/liter at time', tmax,
'minutes.')
```

```
Concentration c_B has maximum 1.3374806339222158 moles/liter at time 4.023594924340666
minutes.
```

Solution Using Pyomo

The variable to be found is the time t_f corresponding to the maximum concentration of B. For this purpose we introduce a scaled time

$$\tau = \frac{t}{t_f}$$

so that $\tau = 1$ as the desired solution. The problem then reads

$$\max_{t_f} C_B(\tau = 1)$$

subject to

$$\begin{aligned}\frac{dC_A}{d\tau} &= -t_f k_A C_A \\ \frac{dC_B}{d\tau} &= t_f(k_A C_A - k_B C_B)\end{aligned}$$

The solution to this problem is implemented as a solution to the following Pyomo model.

In [8]:

```
from pyomo.environ import *
from pyomo.dae import *

V   = 40     # liters
kA  = 0.5    # 1/min
kB  = 0.1    # 1/min
cAf = 2.0    # moles/liter

m = ConcreteModel()

m.tau = ContinuousSet(bounds=(0, 1))

m.tf = Var(domain=NonNegativeReals)
m.cA = Var(m.tau, domain=NonNegativeReals)
m.cB = Var(m.tau, domain=NonNegativeReals)

m.dcA = DerivativeVar(m.cA)
m.dcB = DerivativeVar(m.cB)

m.odeA = Constraint(m.tau,
    rule=lambda m, tau: m.dcA[tau] == m.tf*(-kA*m.cA[tau]) if tau > 0 else Constraint.Sk
ip)
m.odeB = Constraint(m.tau,
    rule=lambda m, tau: m.dcB[tau] == m.tf*(kA*m.cA[tau] - kB*m.cB[tau]) if tau > 0 else
Constraint.Skip)

m.ic = ConstraintList()
m.ic.add(m.cA[0]  == cAf)
m.ic.add(m.cB[0]  == 0)

m.obj = Objective(expr=m.cB[1], sense=maximize)

TransformationFactory('dae.collocation').apply_to(m)
SolverFactory('ipopt', executable=ipopt_executable).solve(m)
print('Concentration c_B has maximum', m.cB[1](), 'moles/liter at time', m.tf(), 'minut
es.')
```

```
Concentration c_B has maximum 1.3374805810221073 moles/liter at time 4.023594178375687
minutes.
```

In [0]:

```

```