6.1 Unconstrained Scalar Optimization

Application: Maximizing Production of a Reaction Intermediate

A desired product B is produced as intermediate in a series reaction

$$A \xrightarrow{k_A} B \xrightarrow{k_B} C$$

where A is a raw material and C is a undesired by-product. The reaction operates at temperature where the rate constants are $k_A = 0.5\ \text{min}^{-1}$ and $k_A = 0.1\ \text{min}^{-1}$. The raw material is available as a solution with concenration $C_{A,f} = 2.0\ \text{moles}/\text{liter}$.

A 100 liter tank is avialable to run the reaction. Below we will answer the following questions:

1. If the goal is obtain the maximum possible concentration of B, and the tank is operated as a continuous stirred tank reactor, what should be the flowrate?
2. What is the production rate of B at maximum concentration?

Mathematical Model for a Continuous Stirred Tank Reactor

The reaction dynamics for an isothermal continuous stirred tank reactor with a volume $V = 40$ liters and feed concentration $C_{A,f}$ are modeled as

$$\begin{aligned} V\frac{dC_A}{dt} &= q(C_{A,f} - C_A) - Vk_AC_A \\ V\frac{dC_B}{dt} &= -qC_B + Vk_AC_A - Vk_BC_B \end{aligned}$$

At steady-state the material balances become

$$\begin{aligned} 0 &= q(C_{A,f} - \bar{C}_A) - Vk_A\bar{C}_A \\ 0 &= -q\bar{C}_B + Vk_A\bar{C}_A - Vk_B\bar{C}_B \end{aligned}$$

which can be solved for C_A

$$\bar{C}_A = \frac{qC_{A,f}}{q + Vk_A}$$

and then for C_B

$$\bar{C}_B = \frac{qVk_AC_{A,f}}{(q + Vk_A)(q + Vk_B)}$$

The numerator is first-order in flowrate q, and the denominator is quadratic. This is consistent with an intermediate value of q corresponding to a maximum concentration $\bar{C}_B$.

The next cell plots $\bar{C}_B$ as a function of flowrate q.

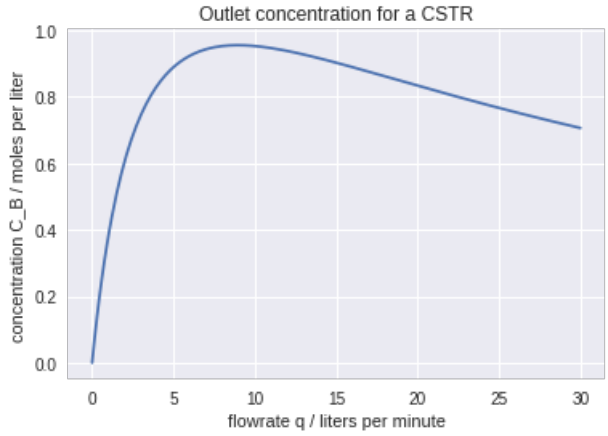
In [9]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

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

def cstr(q):
    return q*V*kA*CAf/(q + V*kB)/(q + V*kA)

q = np.linspace(0,30,200)
plt.plot(q, cstr(q))
plt.xlabel('flowrate q / liters per minute')
plt.ylabel('concentration C_B / moles per liter')
plt.title('Outlet concentration for a CSTR')
plt.show()
```



We see that, for the parameters given, there is an optimal flowrate somewhere between 5 and 10 liters per minute.

Analytical Solution using Calculus

As it happens, this problem has an interesting analytical solution that can be found by hand, and which can be used to check the accuracy of numerical solutions. Setting the first derivative of $\bar{C}_B$ to zero,

$$\left.\frac{d\bar{C}_B}{dq}\right|_{q^*} = \frac{Vk_AC_{A,f}}{(q^*+Vk_A)(q^*+Vk_B)} - \frac{q^*Vk_AC_{A,f}}{(q^*+Vk_A)^2(q^*+Vk_B)} - \frac{q^*Vk_AC_{A,f}}{(q^*+Vk_A)(q^*+Vk_B)^2} = 0$$

Clearing out the non-negative common factors yields

$$1 - \frac{q^*}{(q^*+Vk_A)} - \frac{q^*}{(q^*+Vk_B)} = 0$$

and multiplying by the non-negative denominators produces

$$q^{*2} + q^*V(k_A+k_B) + V^2k_Ak_B - q^*(q^*+Vk_B) - q^*(q^*+Vk_A) = 0$$

Expanding these expressions followed by arithmetic cancelations gives the final result

$$q^* = V\sqrt{k_Ak_B}$$

which shows the optimal dilution rate, $\frac{q^*}{V}$, is equal the geomtric mean of the rate constants.

In [10]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

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

qmax = V*np.sqrt(kA*kB)
CBmax = cstr(qmax)
print('Flowrate at maximum CB = ', qmax, 'liters per minute.')
print('Maximum CB =', CBmax, 'moles per liter.')
print('Productivity = ', qmax*CBmax, 'moles per minute.')
```

```
Flowrate at maximum CB =  8.94427190999916 liters per minute.
Maximum CB = 0.9549150281252629 moles per liter.
Productivity =  8.541019662496845 moles per minute.
```

Numerical Solution with Pyomo

This problem can also be solved using Pyomo to create a model instance. First we make sure that Pyomo and ipopt are installed, then we proceed with the model specification and solution.

In [0]:

```
!pip install -q pyomo
!wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip" && unzip -o -q ipopt-li
nux64
ipopt_executable = '/content/ipopt'
```

In [12]:

```
from pyomo.environ import *

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

# create a model instance
m = ConcreteModel()

# create the decision variable
m.q = Var(domain=NonNegativeReals)

# create the objective
m.CBmax = Objective(expr=m.q*V*kA*CAf/(m.q + V*kB)/(m.q + V*kA), sense=maximize)

# solve using the nonlinear solver ipopt
SolverFactory('ipopt', executable=ipopt_executable).solve(m)

# print solution
print('Flowrate at maximum CB = ', m.q(), 'liters per minute.')
print('Maximum CB =', m.CBmax(), 'moles per liter.')
print('Productivity = ', m.q()*m.CBmax(), 'moles per minute.')
```

```
Flowrate at maximum CB =  8.944271964904415 liters per minute.
Maximum CB = 0.9549150281252627 moles per liter.
Productivity =  8.541019714926698 moles per minute.
```

One advantage of using Pyomo for solving problems like these is that you can reduce the amount of algebra needed to prepare the problem for numerical solution. This not only minimizes your work, but also reduces possible sources of error in your solution.

In this example, the steady-state equations are

$$\begin{aligned} 0 &= q(C_{A,f} - \bar{C}_A) - Vk_A\bar{C}_A \\ 0 &= -q\bar{C}_B + Vk_A\bar{C}_A - Vk_B\bar{C}_B \end{aligned}$$

with unknowns C_B and C_A. The modeling strategy is to introduce variables for the flowrate q and these unknowns, and introduce the steady state equations as constraints.

In [13]:

```
from pyomo.environ import *

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

# create a model instance
m = ConcreteModel()

# create the decision variable
m.q  = Var(domain=NonNegativeReals)
m.CA = Var(domain=NonNegativeReals)
m.CB = Var(domain=NonNegativeReals)

# equations as constraints
m.eqn = ConstraintList()
m.eqn.add(0 == m.q*(CAf - m.CA) - V*kA*m.CA)
m.eqn.add(0 == -m.q*m.CB + V*kA*m.CA - V*kB*m.CB)

# create the objective
m.CBmax = Objective(expr=m.CB, sense=maximize)

# solve using the nonlinear solver ipopt
SolverFactory('ipopt', executable=ipopt_executable).solve(m)

# print solution
print('Flowrate at maximum CB = ', m.q(), 'liters per minute.')
print('Maximum CB =', m.CBmax(), 'moles per liter.')
print('Productivity = ', m.q()*m.CBmax(), 'moles per minute.')
```

```
Flowrate at maximum CB =  8.944272002876573 liters per minute.
Maximum CB = 0.9549150281377385 moles per liter.
Productivity =  8.541019751298471 moles per minute.
```

In [0]:

```

```