Memorandum for a PBR Performance Investigation

Date: 25th April 2021

To: Dr. Vikramaditya G. Yadav

From: Muhammad Usman Siddiqui (46205233)

Subject: CHBE 355 Final

Introduction

The performance of a pilot-scale PBR enveloped by a heat exchanger is investigated. The reactor is used to undertake a gaseous condensation reaction (1). In the investigation, the kinetics of the reaction are determined. The Thiele modulus and the effectiveness of the catalyst used are found. And the conversion, pressure drop, and exit temperature of the coolant from the PBR are calculated. This memo summarizes the findings of the investigation, and the methodology used to obtain those findings. Based on the findings, recommendations are made to increase the conversion in the reactor. Python has been used in all of the analysis and the code used is provided in Appendix A.

$$A + B \rightarrow C \tag{1}$$

Results and Discussion

Determining the Kinetics of the reaction

Experimental data providing the concentration of the reactants in a sixty minute time interval are obtained from from a highly pressurized, fixed volume batch reactor operating at the same temperature as the PBR. The reactants are gaseous so the volume inside the reactor can change but a high pressure ensures that the volume remains constant inside. At a high pressure, the rate of reaction is also higher and the phase does not change.

Linear regression is used and the experimental data are fitted linearly to find the order of the reaction and the reaction rate constant. For each reactant (A and B) multiple plots are generated corresponding to the different order of reactions. The Concentration (C) vs time, ln (C) vs time, and the 1/C vs time graphs are plotted. All the plots are presented in Appendix A. The ln (C) vs time plot represents a first order reaction. This plot gives a straight line for both the reactants which means the reaction is first order in A and in B. The overall rate constant is determined by taking the slope of each plot, dividing it by 10 mol/m^3 (the initial concentration) and taking their average. The rate constant is calculated to be 0.0079 m^3/mol.min, and the reaction is second order overall. The rate expression obtained is shown below (2).

$$-rA = k[CA][CB] \tag{2}$$

Where: CA = Concentration of A

CB = Concentration of B

k = 0.0079 m^3/mol.min

Analysis on The Catalyst

A mass transport analysis is conducted to find the thiele modulus, and internal and external effectiveness factors of the catalyst. A boundary value problem is formulated to find these parameters. The differential equation (3) describing the second order reaction in the spherical catalyst is simplified to (4) and is coded. The boundary conditions used are shown in Figure 1. The differential equation is solved numerically using the shooting method in which the initial value of d**C**/dr is guessed and the initial value of dJ/dr where J = d**C**/dr is specified as zero; then the equation is resolved with the correct initial value of d**C**/dr.

$$\frac{D_A}{r^2} \times \frac{d}{dr}\left(r^2 \frac{dC_A}{dr}\right) = k_{ov} C_A^{\,2} \qquad (3)$$

$$\frac{d^2 \Psi}{dr^2} = \Psi^2 \times \frac{\Phi^2}{R^3} \qquad (4)$$

where D_A = effective diffusivity $\qquad k_{ov} = 0.0079 \; \frac{m^3}{mol\,min}$

r = radius

$\frac{\Psi}{r} = \frac{C_A}{C_{A\,surf}}$

Φ^2 = Thiele modulus,

R = maximum radius

$$\text{At } r = R,\ \Psi = R;\ \text{At } r = 0,\ \left(\frac{dC_A}{dr} = 0\right),\ \frac{d\Psi}{dr} = 0$$

Figure 1: Boundary conditions for the shooting method

The expression for Thiele modulus is obtained by using its definition that it is a ratio of the reaction rate to the diffusion rate and the fact that it is dimensionless. It's expression obtained is shown below (5). The expression for the internal effectiveness factor ***η*** derived for a second order reaction is also shown below (6).

$$\Phi^2 = \frac{k_{ov} \times R^2 \times C_{Asurf}}{D_A} \quad (5)$$

wher C_{Asurf} = Concentration of A at surface.

$$\eta = \frac{3\,|W_A|_{r=R}|}{k_{ov}\, C_{Asurf}^{2}\, R} \quad (6)$$

where $|W_A|_{r=R}| = \frac{dC}{dr} \times D_A$

The d**C**/dr value obtained from the shooting method at r =R is used to find ***η*** as shown in (3). The value of the concentration of A at the surface of the catalyst (CAsurf) is not known and is guessed between 0 and 3. Different values of CAsurf are tried, and the ***η*** and Thiele modulus values for each CAsurf are calculated. The relevant results obtained are shown in Table 1. From these a logical value of CAsurf is chosen which gives us our desired ***η***.

CAsurface (M)	η	Thiele Modulus
0.009	0.821	1.074
0.012	0.412	1.499
0.024	0.104	2.99

Table 1: CAsurface, η, and thiele modulus values

The value for ***η*** is chosen to be 0.412 and the value of Thiele modulus is chosen to be 1.499 at a CAsurf of 0.012 M. Smaller CAsurf values give ridiculously large ***η*** and larger CAsurf give small ***η*** values. A high thiele modulus results in a lot of catalyst being unused and a small value results in more unreacted reactants. Thus, a combination that gives a reasonable ***η***, and a moderate value of the Thiele modulus is required . By method of elimination resulting in elimination of all impractical points CAsurf is 0.012 M. At CAsurf = 0.012 M, a nice balanced between ***η*** and the thiele modulus value is obtained.

The overall effectiveness factor (**Ω**) of the catalyst is calculated using the expression (7). It is calculated to be 0.412. **Ω** is used to find the rate constant used to solve the PBR.

$$\Omega = \frac{\eta k_{conv}}{k_{conv} + \eta k_{ov} (1 - \phi_{cat}) \left(\frac{R}{3}\right) C_{Asurf}} \quad (7)$$

$$\text{where } k_{conv} = \frac{D_A}{\text{thickness of boundary layer}}$$

Solving the PBR

Four differential equations modelling the PBR are obtained and solved using Python. The Differential Equations (8) -(11) are shown below. The value of k used in the rate expression is the product of k overall and **Ω** (k = kov * **Ω**).

$$\frac{dy}{dW} = \frac{-\alpha}{2y}(1 + \varepsilon X_A) \quad (8)$$

$$\text{where } \varepsilon = -0.5$$

$$\frac{dX_A}{dW} = \frac{k C_A C_B}{F_{Ao}} \quad (9)$$

$$\frac{dT_c}{dW} = \frac{U\alpha(T - T_c)}{\dot{m}_{cool} \; C_{pcool}} \quad (10)$$

$$\frac{dT}{dW} = \frac{-U\alpha'(T - T_c) - (-r_A)\Delta H_{rxn}}{\Sigma F_i C_{pi}} \quad (11)$$

Where: y = pressure drop

XA = Conversion of A

Tc = Temperature of coolant

T = Temperature of reactor

The differential equations are solved analytically, and the variation of parameters along the catalyst bed are plotted and displayed in Figures 2 and 3.

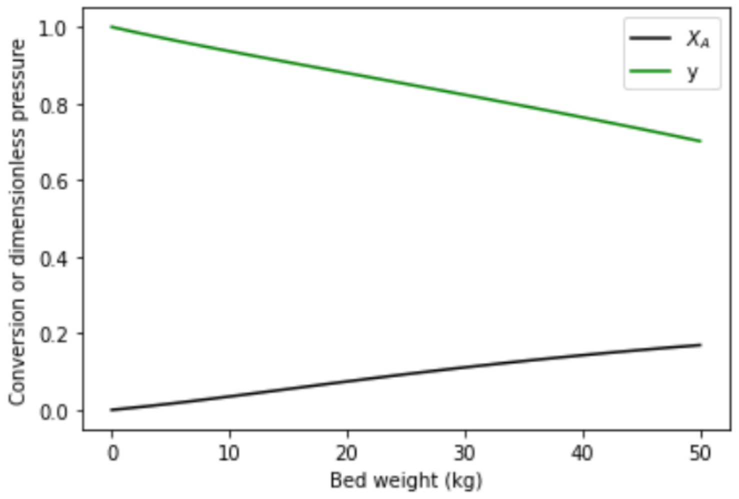


Figure 2: Pressure drop and conversion vs bed weight

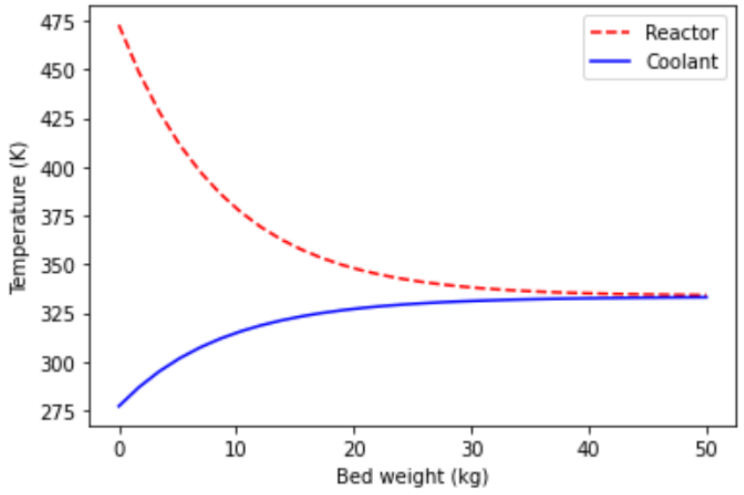


Figure 3: Temperature of reactor and coolant vs bed weight.

It is determined that the Conversion of A in the system is 0.17, the pressure drop in the reactor is 29.81 %, and outlet temperature of the coolant is 333.02 K.

Recommendations

Increasing the reactor temperature by 50 K increases the conversion of A to 0.19 which is an increase of 11.8%. Increasing the temperature further by 50 K only causes the conversion to increase to 0.2 which is 5.3%. Considering the cost and risks associated with increasing temperature, an increase of only 50 K is recommended. Increasing the inlet concentration of reactant A has a big impact on the conversion. Increasing the concentration of A by 3 M causes the conversion to increase to 0.26 which is an increase of 52.9 %. It is recommended to increase the operating temperature of the reactor by 50 K and increase the inlet concentration of A by 3 M; both of these combined increase the conversion by 59%.

Statement of Ethics

I declare that all the contents of this report are produced by me. I collaborated with Alyeldin Helmy, Fortune Komolafe, and Cindy Lam to obtain some of the differential equations stated above, and discussed general problem solving techniques with them.

Appendix A:
Python Code

In [24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

In [43]:
```
# Importing the Excel data

data1 = pd.read_excel(r'Kinetic data.xls', sheet_name='Experiment 1' )
data2 = pd.read_excel(r'Kinetic data.xls', sheet_name= 'Experiment 2' )
```

In [ ]:
```
# Finding the order of reaction in A
```

In [68]:
```
time1 = data1['Time (min)'].values
concA = data1['CA (M)'].values
```

In [9]:
```
time1
```

Out[9]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60], dtype=int64)
```

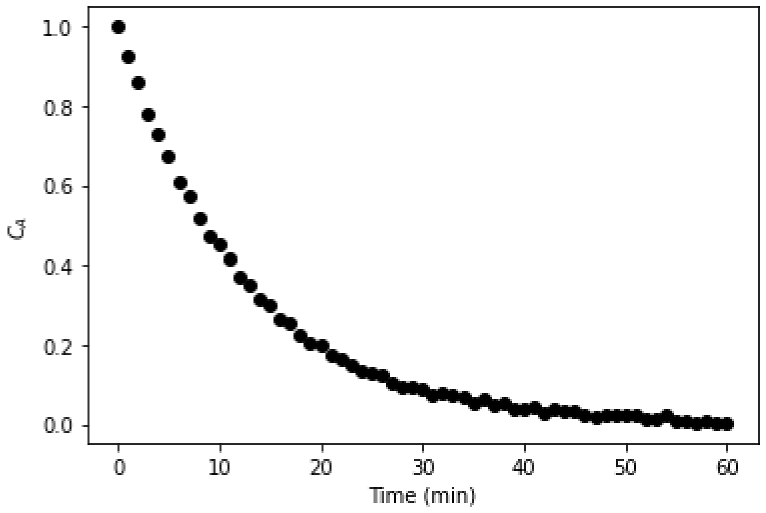
In [69]:
```
time2 = data2['Time (min)'].values
concB = data2['CB (M)'].values
```

In [27]:
```
concB
```

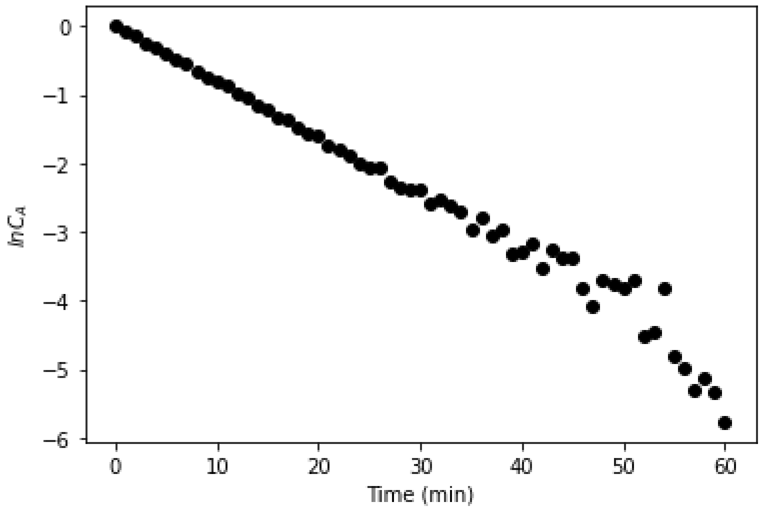
Out[27]:
```
array([1.        , 0.92952973, 0.86244848, 0.78636541, 0.72565755,
       0.67296879, 0.62048022, 0.58376741, 0.53865066, 0.48744322,
       0.4512177 , 0.4184111 , 0.39685989, 0.36454591, 0.3307865 ,
       0.31330113, 0.27804602, 0.25772376, 0.24740911, 0.22068821,
       0.21423081, 0.18618045, 0.18292585, 0.15972038, 0.15956841,
       0.13461478, 0.12489025, 0.12542682, 0.11661382, 0.10084443,
       0.09914995, 0.0821979 , 0.08598414, 0.06905906, 0.07316583,
       0.05812204, 0.06432998, 0.05202096, 0.05847434, 0.04412466,
       0.05018139, 0.04625111, 0.03268502, 0.03333504, 0.02674997,
       0.02798006, 0.02631608, 0.02132571, 0.02108028, 0.02831794,
       0.01568882, 0.01678503, 0.02548466, 0.02061271, 0.01967683,
       0.01971783, 0.02046597, 0.0188523 , 0.01701709, 0.01537514,
       0.01425987])
```

In [165]:
```
# Defining the different order plots

ln_concA = np.log(concA)  #First Order
rec_concA = 1/concA  #Second Order
rec_conc2A = 1/(concA**2)  # Third order
```

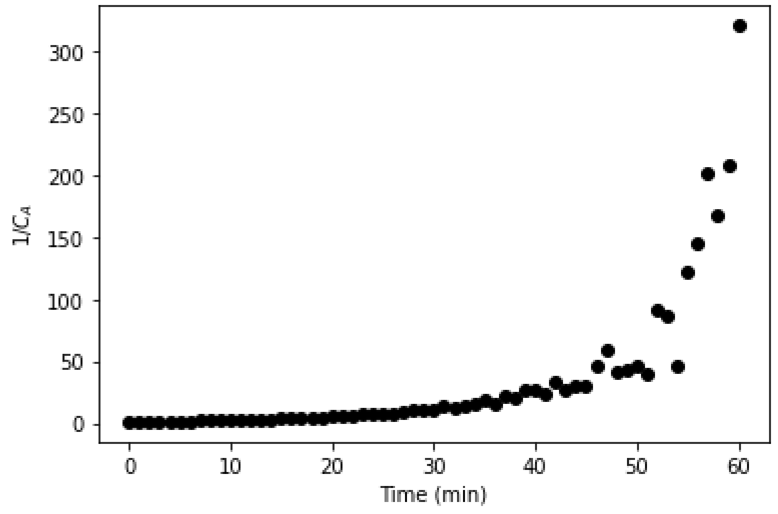
In [166]:
```python
#Zeroth Order

plt.plot(time1, concA, "ko")
plt.xlabel("Time (min)")
plt.ylabel("$C_A$")
plt.show()
```



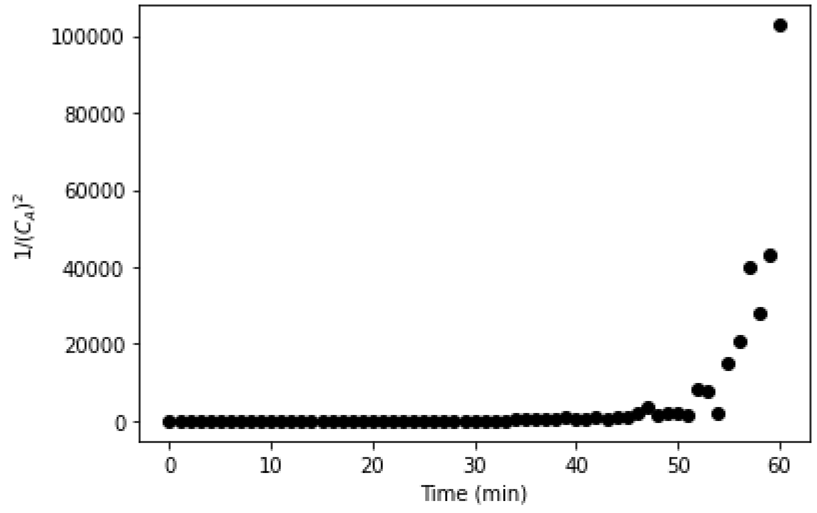
In [72]:
```python
# First Order

plt.plot(time1, ln_concA, "ko")
plt.xlabel("Time (min)")
plt.ylabel("$lnC_A$")
plt.show()
```



In [163]:
```
# Second Order

plt.plot(time1, rec_concA, "ko")
plt.xlabel("Time (min)")
plt.ylabel("$1/C_A$")
plt.show()
```



In [164]:
```
#Third Order

plt.plot(time1, rec_conc2A, "ko")
plt.xlabel("Time (min)")
plt.ylabel("$1/(C_A)^2 $")
plt.show()
```



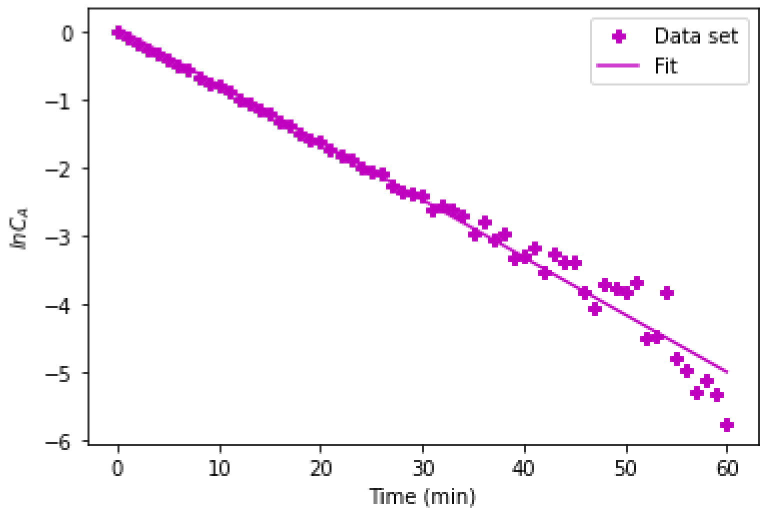
In [135]:
```python
# The first order curve looks linear so we will fit that
# The reaction is first order in A
```

In [150]:
```python
# Data Fitting

from sklearn.linear_model import LinearRegression

model = LinearRegression()
trans_time1 = time1.reshape((-1,1))  #transpose
model.fit(trans_time1,ln_concA)  #Linear regression
```

Out[150]:
```
LinearRegression()
```

In [151]:
```python
intercept1 = model.intercept_
slope1 = model.coef_
R_2 = model.score(trans_time1, ln_concA) #R^2
```

In [162]:
```python
#Plotting the Fit

trendline = model.predict(trans_time1)
plt.plot(time1, ln_concA, "mP", label ='Data set')
plt.plot(time1, trendline, "m", label ='Fit')
plt.xlabel("Time (min)")
plt.ylabel("$lnC_A$")
plt.legend()
plt.show()
```

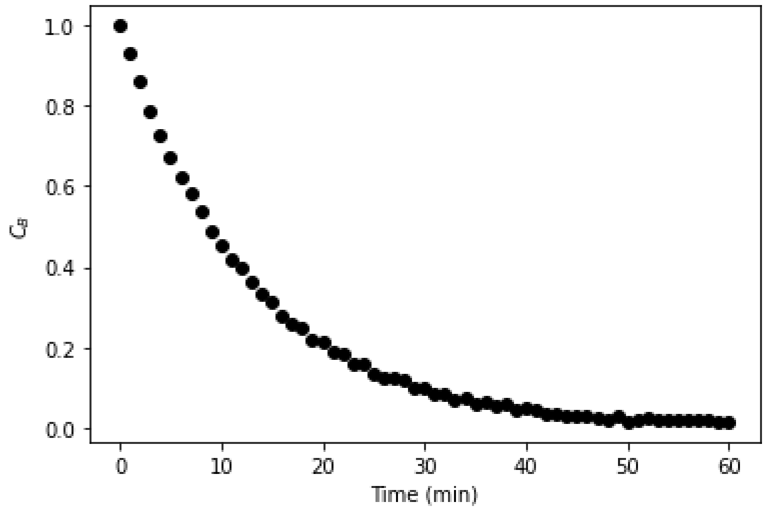


In [90]:
```python
# Finding the order of reaction in B
```

In [154]:
```python
# Defining the different order plots

ln_concB = np.log(concB)  #First Order
rec_concB = 1/concB  #Second Order
rec_conc2B = 1/(concB**2)  # Third order
```

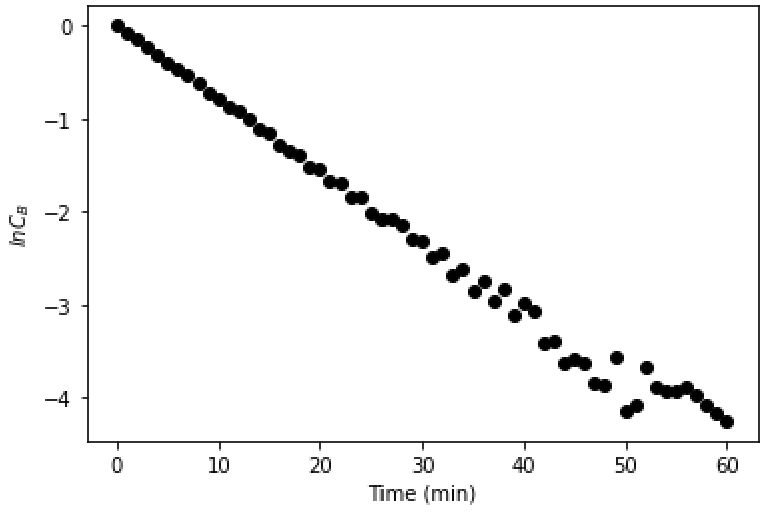
In [155]:

```
#Zeroth Order

plt.plot(time2, concB, "ko")
plt.xlabel("Time (min)")
plt.ylabel("$C_B$")
plt.show()
```

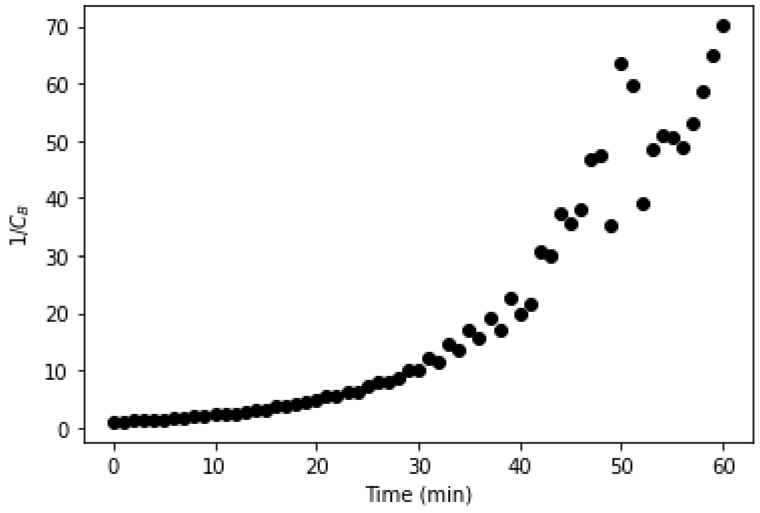


In [156]:

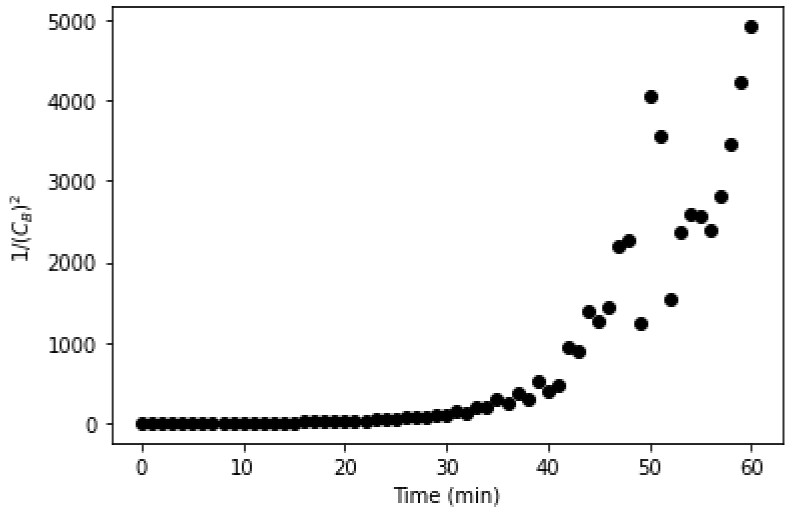
```
# First Order

plt.plot(time2, ln_concB, "ko")
plt.xlabel("Time (min)")
plt.ylabel("$lnC_B$")
plt.show()
```



In [161]:
```python
# Second Order

plt.plot(time2, rec_concB, "ko")
plt.xlabel("Time (min)")
plt.ylabel("$1/C_B$")
plt.show()
```



In [158]:
```python
#Third Order

plt.plot(time2, rec_conc2B, "ko")
plt.xlabel("Time (min)")
plt.ylabel("$1/(C_B)^2$")
plt.show()
```



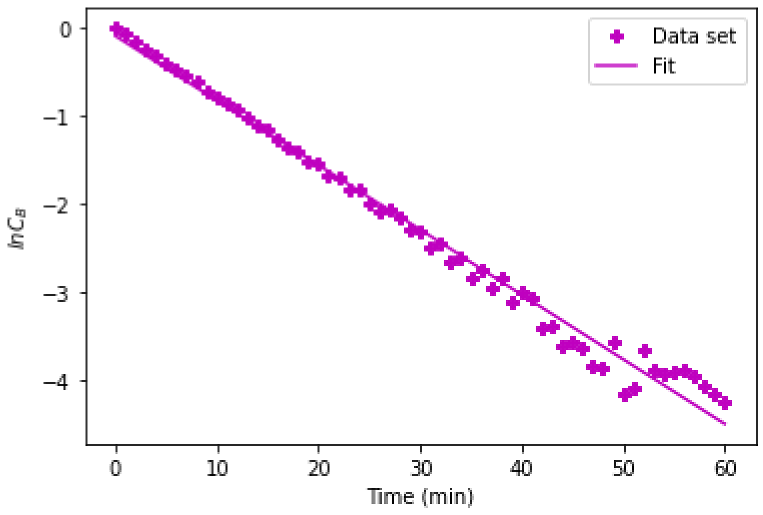
In [125]:
```
# First order curve is linear so we fit that
# The reaction is first order in B
```

In [132]:
```
# Fitting the Data for B

model2 = LinearRegression()
trans_time2 = time2.reshape((-1,1))  #transpose
model2.fit(trans_time2,ln_concB)  #Linear regression
```

Out[132]:
```
LinearRegression()
```

In [131]:
```
intercept2 = model2.intercept_
slope2 = model2.coef_
R_2B = model2.score(trans_time2, ln_concB) #R^2
```

In [159]:
```
#Plotting the Fit

trendlineB = model2.predict(trans_time2)
plt.plot(time2, ln_concB, "mP", label ='Data set')
plt.plot(time2, trendlineB, "m", label ='Fit')
plt.xlabel("Time (min)")
plt.ylabel("$lnC_B$")
plt.legend()
plt.show()
```



In [185]:
```
# The actual rate constant

k = -(slope1/10 + slope2/10)/2  # (m^3/mol.min)
print('The value of k is', k)
```

```
The value of k is [0.00790161]
```

In [ ]:

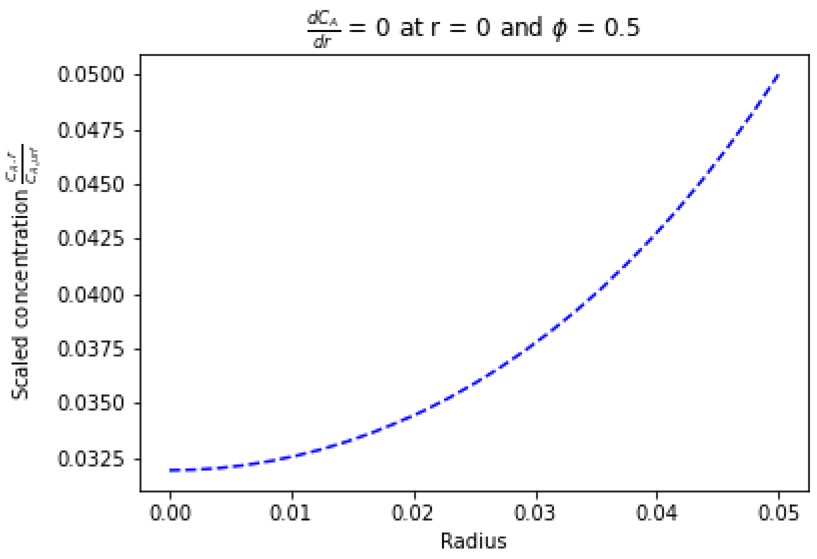
In [190]:
```python
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fmin
import matplotlib.pyplot as plt
```

In [300]:
```python
R = 0.05 #m
Deff = 1.6e-7
Cas = 0.0121
CAb = 3 # M
r = np.linspace(0,R,25)
ko = 0.0079
kc = Deff/1e-4
rho = 6 #kg/L

PHI = (ko*R**2*Cas/(Deff))
```

In [301]:
```python
def bvp(S, r):
    # Ca = (k*CA - S[1])/k
    dCdr = S[1]
    dJdr = (PHI/R**3)*S[0]**2
    dSdr = [dCdr, dJdr]
    return dSdr


#Shooting method for case of dCdr = 0 at r = 0

def shooting2(C_guess):
    S0 = [C_guess, 0]
    S2 = odeint(bvp, S0, r)
    C_numerical2 = S2[:,0]
    k = np.size(r)-1
    sol2 = abs(C_numerical2[k] - R)
    return sol2

#Running the optimization
C_ini_correct = fmin(shooting2, 0, xtol=1e-8)
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 28
         Function evaluations: 56
```

In [302]:
```
#Resolving the ODE with the correct initial value of G
S_ini2 = [C_ini_correct,0,]
S_correct2 = odeint(bvp, S_ini2, r)
C_correct2 = S_correct2[:,0]

#Plotting the result
plt.plot(r, C_correct2, "b--", label = 'Numerical solution')
plt.xlabel("Radius")
plt.ylabel("Scaled concentration " r'$\frac{C_A.r}{C_{A_surf}}$')
plt.title(r'$\frac{dC_A}{dr}$' " = 0 at r = 0 and " r'$\phi$' " = 0.5" )
plt.show()
```



In [264]:
```
C_correct2
```

Out[264]:
```
array([0.03191951, 0.03194594, 0.03202531, 0.03215788, 0.0323441 ,
       0.03258459, 0.03288016, 0.03323181, 0.03364075, 0.0341084 ,
       0.03463641, 0.03522665, 0.03588127, 0.03660268, 0.03739361,
       0.03825708, 0.03919648, 0.0402156 , 0.04131863, 0.04251024,
       0.04379562, 0.04518052, 0.04667135, 0.0482752 , 0.04999999])
```

In [295]:
```
CAs = np.linspace(0.0001, 0.05, 30)

eta = (3*C_correct2[24]*Deff)/(ko * CAs**2 * 0.05)
```

```
In [305]:  CAs

Out[305]: array([0.0001    , 0.00182069, 0.00354138, 0.00526207, 0.00698276,
                 0.00870345, 0.01042414, 0.01214483, 0.01386552, 0.01558621,
                 0.0173069 , 0.01902759, 0.02074828, 0.02246897, 0.02418966,
                 0.02591034, 0.02763103, 0.02935172, 0.03107241, 0.0327931 ,
                 0.03451379, 0.03623448, 0.03795517, 0.03967586, 0.04139655,
                 0.04311724, 0.04483793, 0.04655862, 0.04827931, 0.05      ])
```

```
In [304]:  eta

Out[304]: array([6.07595159e+02, 1.83291555e+00, 4.84472816e-01, 2.19432643e-01,
                 1.24612109e-01, 8.02105694e-02, 5.59157309e-02, 4.11937755e-02,
                 3.16040074e-02, 2.50111368e-02, 2.02850419e-02, 1.67821233e-02,
                 1.41140053e-02, 1.20350569e-02, 1.03837674e-02, 9.05040302e-03,
                 7.95829466e-03, 7.05256456e-03, 6.29309548e-03, 5.65001065e-03,
                 5.10068991e-03, 4.62775301e-03, 4.21766778e-03, 3.85977120e-03,
                 3.54556925e-03, 3.26822818e-03, 3.02219982e-03, 2.80294188e-03,
                 2.60670683e-03, 2.43038064e-03])
```

```
In [303]:  thiele = (ko*R**2*CAs)/Deff
           thiele

Out[303]: array([0.01234375, 0.22474138, 0.43713901, 0.64953664, 0.86193427,
                 1.0743319 , 1.28672953, 1.49912716, 1.71152478, 1.92392241,
                 2.13632004, 2.34871767, 2.5611153 , 2.77351293, 2.98591056,
                 3.19830819, 3.41070582, 3.62310345, 3.83550108, 4.04789871,
                 4.26029634, 4.47269397, 4.68509159, 4.89748922, 5.10988685,
                 5.32228448, 5.53468211, 5.74707974, 5.95947737, 6.171875  ])
```

```
In [293]:  CAsurface = 0.015586
           eta = 0.4119
           thiele = 1.499
```

```
In [294]:  Omega = (eta * kc)/(kc + eta*ko * (1 - thiele**0.5)* (R/3) * CAsurface)
           Omega

Out[294]: 0.41190488179240864
```

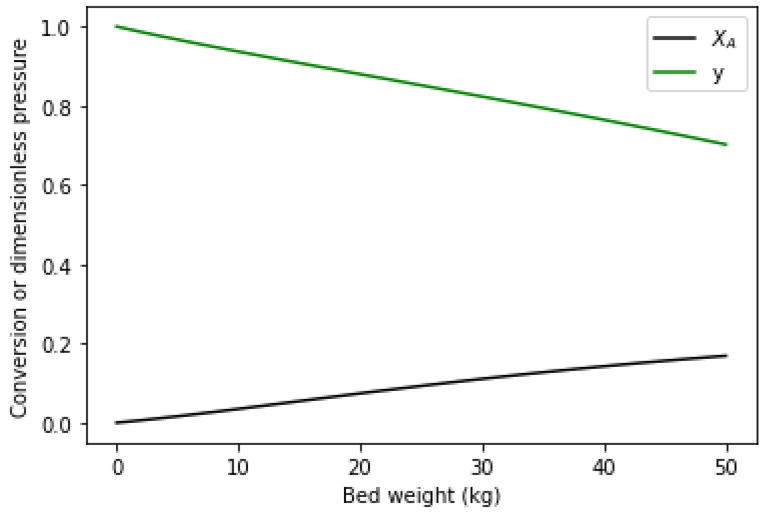
In [62]:
```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

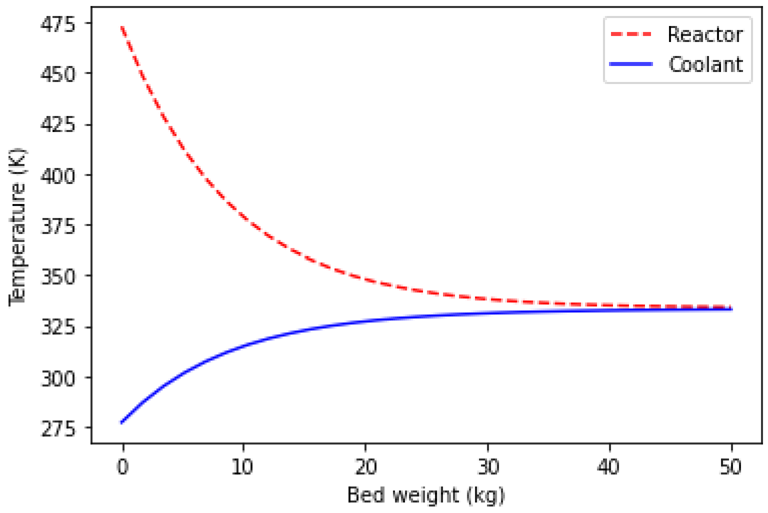
In [63]:
```python
# Defining parameters
# A + B > C
# Since inlet flow is equimolar CA = CB
# PBR is isothermal so (T/To) = 1

#Stoichiometry
a = 1
b = 1
c = 1
delta = c - a - b

# Inlet conditions
yAo = 0.5 #pure A in inlet
CAo = 3 # mol m^-3
vo = 10/3 # L/s
thetaB = 1
eps = (yAo*delta)/a
Tc = 277 #K
Omega = 0.412


#Reaction rate data
k_o = 0.0079 #(m^3/mol.min)
k = k_o * Omega


#Thermodynamic data
Hrxno = -100000 #J/mol
CpA = 2000 #J/mol.K
CpB = 3000 #J/mol.K
CpC = 2400 #J/mol.K
deltaCp = (CpC-((b*CpB) + (a*CpA)))/a

To = 473 #K

Tref = 298 #K

#Heat exchange and coolant data
U = 40/3 #W/sqm.K
a = 300 #surface area per unit volume
mc = 5/3 #coolant mass flow rate (kg/s)
CpCool = 75000 #J/m^3.K

#Ergun parameters
alpha = 0.014 #1/kg
```

In [64]:

```
#Define the functions
# S = [XA, T, y, Tc]

def pbr(S, W):
    CA = (CAo * S[2] * To * (1-S[0]))/(S[1]*(1+eps*S[0]))
    CB = CA
    Hrxn = Hrxno
    dXAdW = (k*CA*CB)/(CAo*vo)
    dTdW = -(U*a*(S[1]-S[3])+ k*CA*CB*Hrxn)/(CAo*vo*((1-S[0])*(CpA +CpB)+S[0]
    dydW = -(alpha*S[1]*(1 + eps*S[0]))/(2*S[2]*To)
    dTcdW = (U*a*(S[1]-S[3]))/(mc*CpCool)
    dSdW =  [dXAdW, dTdW, dydW, dTcdW]
    return dSdW
```

In [65]:
```python
#Solving the system of ODEs
W = np.linspace(0, 50, 30)
S0 = [0,To,1,Tc]
S = odeint(pbr, S0, W)

#Extracting the solutions
XA = S[:,0]
T = S[:,1]
y = S[:,2]
Tc = S[:,3]

#Preparing the plots

plt.plot(W, XA, "k", label = '$X_{A}$')
plt.plot(W, y, "g", label = 'y')
plt.xlabel("Bed weight (kg)")
plt.ylabel("Conversion or dimensionless pressure")
plt.legend()
plt.show()

plt.plot(W, T, "r--", label = "Reactor")
plt.plot(W, Tc, "b", label = "Coolant")
plt.xlabel("Bed weight (kg)")
plt.ylabel("Temperature (K)")
plt.legend()
plt.show()

print('The exit temperature of the reacting stream is',np.round(T[len(T)-1],2
print('The exit conversion is',np.round(XA[len(XA)-1],2))
print('The pressure drop is',np.round((1-y[len(y)-1])*100,2),'%')
Tcout = Tc[len(Tc)-1]
```





```
The exit temperature of the reacting stream is 334.13 K
The exit conversion is 0.17
The pressure drop is 29.81 %
```

In [66]:
```
Tcout
```

Out[66]:
```
333.02167858455203
```

In [ ]: