CSTR (9.16)

0 = Fa0 – Fa + Vra

Given: k, Fa0, V, v0; Ca?

```
k = 0.23   # 1/hr; Fa0 = 1.0  # mol /hr
v0 = 2.5   # L /hr; V = 10     # L
def func(Ca):
    ra = -k * Ca
    Fa = v0 * Ca
    return Fa0 - Fa + V * ra
guess = 1.0  # mol / L
ans, = fsolve(func, guess)
print('Ca_exit = {0} mol/L'.format(ans))
```

Batch (9.16)

dCa/dt = -kCa

Given: k, Ca0, t; Ca?

```
k = 0.23   # 1/hr; Ca0 = 2.0  # mol / L
def ode(Ca, t):
    dCadt = -k * Ca
    return dCadt
tspan = np.linspace(0, 1)  # hours
sol = odeint(ode, Ca0,  tspan)
```

PFR (9.16)

dFa/dV = ra

Given: ra, Ca0, v0, V ; Faexit?

```
Ca0 = 3.0  # mol / L;v0 = 10.0  # L/min
k = 0.23   #  1/min
def ode(Fa, V):
    Ca = Fa / v0
    return -k * Ca
Vspan = [0, 100] # reactor volume
sol = odeint(ode, Ca0 * v0, Vspan)
Fa_exit = sol[-1, 0]
```

Given: Ca0, v0, ra, conversion; V?

```
Ca0 = 3.0 # mol / L;  v0 = 10.0 # L / min
k = 0.23 # 1 / min;    Faexit = 0.3 * v0

def dFadV(Fa,V):
    Ca = Fa/v0
    dFadV = -k*Ca
    return dFadV

def event1(Fa,V):
    value = Fa-Faexit
    isterminal = True
    direction = 0
    return value, isterminal, direction
Vspan = np.linspace(0,200)
V, F, Te, Ye, Ie = odelay(dFadV, Ca0*v0, Vspan,
[event1])
print('V = {0} L'.format(V[-1]))
```

```
%%%
k = 0.23       # 1/min;   v = 10        # L/min
Ca0 = 3.0      # mol / L;  Fa0 = Ca0 * nu;
 Fa = 0.30 * v

def integrand1(Fa):
    return 1.0 / Fa

def integrand2(V):
    return -k / v

def func(V):
    I1, e1 = quad(integrand1, Fa0, Fa)
    I2, e2 = quad(integrand2, 0, V)
    return I1 - I2

guess = 120 # Liters
sol = fsolve(func, guess)
print('Volume = {0:1.2f}'.format(float(sol)))
```

Changing moles (9.21)

C = F/v, dV/dFa = 1/ra

A == B + C

Given: Fa0, Faexit (conversion), v0, k; Fexit?

```
Fa0 = 0.425            # mol / s; Faexit = 0.2 * Fa0
v0 = 0.00581 # m**3 / s k = 0.072            # 1 / s

def dVdFa(V, Fa):
    xi = (Fa - Fa0) / (-1) # compute reaction extent
    Fb = xi * 1
    Fc = xi * 1
    Ft = Fa + Fb + Fc   # total molar flow
    v = v0 * Ft / Fa0   # volumetric flow

    Ca = Fa / v
    ra = -k * Ca
    return 1.0 / ra
Fspan = np.linspace(Fa0, Fa_exit)
V0 = 0
sol = odeint(dVdFa, V0, Fspan)
print('At a volume of {0:1.2f} m^3 we achieve 80%
conversion of A'.format(sol[-1][0]))
```

```
%%%
Fa0 = 0.425 # mol / s
Ft0 = Fa0
Faexit = 0.2 * Fa0

v0 = 0.0058120733613 # m^3 / s
k = 0.072 # 1 / s

def func(F,V):
    Fa, Fb, Fc = F
    Ft = sum(F)
    v = v0 * Ft/Ft0
    Ca = Fa/v
    ra = -k*Ca
    rb = -ra
    rc = -ra
    dFadV = ra
    dFbdV = rb
    dFcdV = rc
    return [ dFadV, dFbdV, dFcdV]

def event(F,V):
    Fa, Fb, Fc = F
    value = Fa - Faexit
    isterminal = True
    direction = 0
    return value, isterminal, direction
Vspan = np.linspace(0,1)
F0 = [Fa0, 0, 0]
V,sol,TE,YE,IE = odelay(func,F0,Vspan,[event])

Fa = sol[:,0]
Fb = sol[:,1]
Fc = sol[:,2]
plt.plot(V,Fa, 'ko',
V,Fb,'b',
V,Fc,'r--')
plt.xlabel('Volume(m3)')
plt.ylabel('Fi(mol/L)')
plt.show()
```

Pressure drops (9.21)

dFa/dW = ra, dy/dW = -alpha/2y*Ft/Ft0

Given: k, ra, Fa0, alpha, conversion, Fexit?

```
Fa0 = 1.08 # lbmol / h; Fb0 = 0.5 * Fa0
FI0 = Fb0 * 0.8/ 0.2 # flow rate of N2
Fc0 = 0.0; Ft0 = Fa0 + Fb0 + FI0 + Fc0
P0 = 10 # atm; alpha = 0.0166 # 1 / lb_m cat
k = 0.0141 # lb-mol / (atm * lb_m cat * h)

def func(F,W):
    Fa, Fb, Fc, y = F
    Ft = Fa + Fb + Fc + FI0
    P = P0 * y
    Pa = Fa/Ft * P
    Pb = Fb/Ft * P
    ra = -k * Pa**(1.0/3.0)*Pb**(2.0/3.0)
    rb = -0.5 * ra
    rc = -ra
    dFadW = ra
    dFbdW = rb
    dFcdW = rc
    dydW = -alpha/(2*y)*Ft/Ft0
    return [dFadW, dFbdW, dFcdW, dydW]

y0 = 1.0
F0 = [Fa0, Fb0, Fc0, y0]
Wspan = np.linspace(0,50)
sol = odeint(func,F0,Wspan)

plt.subplot(1, 2, 1)
plt.plot(Wspan, sol[:, 0:3])
plt.legend(['A', 'B', 'C'],loc='lower center')
plt.xlabel('Catalyst weight ($lb_m$)')
plt.ylabel('Molar flow (mol/min)')

plt.subplot(1, 2, 2)
plt.plot(Wspan, sol[:,3], 'k--') # plot column 3
plt.xlabel('Catalyst weight ($lb_m$)')
plt.ylabel('$P/P_0$')
plt.legend(['$P/P_0$'],loc='upper right')
plt.tight_layout()
plt.show()
Ft = sum(sol[-1,0:3]) # not included last one
```

Transient CSTR (9.23)

dNa/dt = Fa0 – Fa + Vra

dCa/dt = Fa0/V – Fa/V + ra

Given: Ca0, Cain, v0, V, k, ra = kCa, Cexit?

```
Cain = 0.5 # mol/L; v0 = 1.5 # L/min
V = 2.0 # reactor volume (L); Ca0 = 0
Fa0 = Cain * v0; k = 0.11 # rate constant (1/min)

def dCadt(Ca, t):
    Fa = Ca * v0
    ra = -k * Ca
    return Fa0/V-Fa/V + ra
tspan = np.linspance(0, 20)
sol = odeint(dCadt, Ca0, tspan)
```

Semi-batch (9.28)

A + B(added) == C, dNa/dt = ra*V; dNb/dt = rb*V + v0*Cbfeed

Given: Ca0, V0, reactor volume V, B flows in at v, Cbfeed, r = $kCaCb^2$, C_exit?

```
k = 0.02 # L^2/mol^2/min; Ca0 = 2.0 # mol / L
Cbfeed = 2.0 # mol /L; v0 = 0.1 # L / min;
V0 = 5.0 # L

def func(N,t):
    Na, Nb,V = N
    Ca = Na/V
    Cb = Nb/V
    ra = -k* Ca*Cb**2
    rb = ra
    dNadt = ra*V
    dNbdt = rb * V + v0 *Cbfeed
    dVdt = v0
    return [dNadt,dNbdt,dVdt]

tspan = np.linspace(0,50)
N0 = [Ca0*V0, 0,V0]
sol = odeint(func, N0, tspan)
print sol[-1]
```

Membrane reactor (9.28)

dFa/dv = ra +Ra; Ra = a*ka*(Cas – Ca);

Given: k, kb' = a*kb, -ra = kCa, Fa,Cbs,V, Cexit?

```
k = 0.7 # rate constant 1/min
kb = 0.2 # mass transfer coefficent * a 1/min
Ca0 = 0.2 # mol / L; Fa0 = 10.0 # mol / min
Ft0 = Fa0 # Fb0, Fc0 = 0

v0 = Fa0 / Ca0 # inlet volumetric flow

Cbs = 0.0 # concentration of B outside shell

def func(F,V):
    Fa, Fb,Fc = F
    Ft = Fa + Fb + Fc
    v = Ft / Ft0 * v0
    Ca = Fa/v
    Cb = Fb/v
    ra = - k*Ca
    rb = -ra
    Rb = kb*(Cbs-Cb)
    dFadV = ra
    dFbdV = rb + Rb
    dFcdV = -ra
    return [dFadV, dFbdV, dFcdV]
Vspan = np.linspace(0,400)
F0 = [Fa0,0,0]
sol = odeint(func,F0, Vspan)
Faexit, Fbexit, Fcexit = sol[-1,:]
vexit = sum(sol[-1,:])*v0/Ft0
print('Caexit = {0:0.4f} mol/L, Cbexit = {1:0.4f}
mol/L'.format(Faexit/vexit, Fbexit/vexit))
```

Constant P, mole changing batch

A ==2B

Given k, P, Na0, T;t Ca? V?

```
Na0 = 10.0 #mole;  T = 298    # K;   k = 2.3   #
m3/mol/s;  R = 8.314   # kPa* L / mol/K
P = 101.325  # kPa
V0 = Na0 * R * T / P # initial volume
def func(Na, t):
    Nb = 2.0* (Na0 - Na)  # mole of B
    Nt = Na + Nb          # totoal mole
    V = Nt *T *R /P       # total volume
    Ca = Na / V           # concentration of A
    ra = -k * Ca**2       # rate of A
    return ra*V
tspan = np.linspace(0,2000)
sol = odeint(func, Na0, tspan)
V = (2* Na0 - sol) * T * R /P    # total volume
Ca = sol / V                     # concentration of A
plt.subplot(1,2,1)
plt.plot(tspan,Ca)
plt.xlabel('Time(s)')
plt.ylabel('Concentration Ca (mol/L)')
plt.subplot(1,2,2)
plt.plot(tspan,V)
plt.xlabel('Time(s)')
plt.ylabel('Volume (V)')
plt.tight_layout()
plt.savefig('homework.png')
print('The radio of final volume to the initial volume
is {0:1.2f}'.format(V[-1][0]/V0))
```

For

```
Fc[]
for Ca0 in np.linspace(0, 0.05):
    sol = odeint(cstr, Ca0 * V, tspan)
    plt.plot(tspan, sol/V)
Fc.append(Fcexit)
```

Multiple Reactions

Stoichiometry

A = 2B; B = C

ra = -r1; rb = 2r1-r2

```
import uncertainties as u
from scipy.optimize import fsolve

k1 = 0.09;k2 = 0.2
Ca0 = 2.5
def bat(C,t):
    Ca,Cb = C
    r1 = k1*Ca
    r2 = k2*Cb
    ra = -r1
    rb = 2*r1-r2
    dCadt = ra
    dCbdt = rb
    return[dCadt,dCbdt]
C0 = [Ca0,0]
tspan = np.linspace(0,30)
sol = odeint(bat,C0,tspan)
```

Reversible Reactions:

Given: G, T, R, P0 ; C_eq?

```
import numpy as np
R = 1.987   # cal / mol / K
dG = -730   # cal / mol
T = 1000.0  # K

K = np.exp(-dG / R / T)

Pa0 = 5         # atm
R = 0.082057    # L atm / (mol K)
Ca0 = Pa0 / (R * T)

def func(xi):
    nu = np.array([-1, -1, 1, 1])  # stoichiometric
coefficients
    C0 = np.array([Ca0, Ca0, 0.0, 0.0
    C = C0 + nu * xi
    return K - np.prod(C**nu)

from scipy.optimize import fsolve
guess = 0.05
xi_eq, = fsolve(func, guess)
e500 =  (G_rxn(500.0) * 1000 * u.J /
u.mol ).rescale(u.cal / u.mol)
```

Multiple Reaction Equilibrium:

A + B = C, K1; A +B = D, K2

```
ya0 = 0.5 # initial mole fraction of A
yb0 = 0.5 # initial mole fraction of B
P = 2.5   # initial pressure in atm

def xj(extent):
    'convenience function to calculate mole fractions'
    ext1, ext2 = extent
    ya = (ya0 - ext1 - ext2) / (1.0 - ext1 - ext2)
    yb = (yb0 - ext1 - ext2) / (1.0 - ext1 - ext2)
    yc = (ext1) / (1.0 - ext1 - ext2)
    yd = (ext2) / (1.0 - ext1 - ext2)
    return [ya, yb, yc, yd]

def func(extent):
    'zeros function for fsolve'
    ya, yb, yc, yd = xj(extent)
    eq1 = 108.0 - (yc * P)/(ya * P * yb * P)
    eq2 = 284.0 - (yd * P)/(ya * P * yb * P)
    return [eq1, eq2]

from scipy.optimize import fsolve
guess = [0.1, 0.39]
sol = fsolve(func, guess)
print('The reaction extents are:\n',sol)
print('The mole fractions are: \n',xj(sol))
```

Reversible Reactions Rate laws

k1/k2 = Keq; ra = k1(CaCb-CcCd/Keq)

CSTR: Keq, k1, C0, V,v0; Cexit?

Multiple Reaction in CSTR

r1,r2, k1,k2, T,R, F0,

```
def funcC(C):
    vo = 476.0    # ft^3 / hr
    V = 238.0     # ft^3
    Po = 35.0     # atm
    T = 1500.0    # Rankine
    R = 0.73      # in appropriate units
    CTo = Po / R / T
    Cmo = CTo / 3.0
    Cho = CTo * 2.0 / 3.0
    Cxo = 0.0
    Cmeo = 0.0
    Ctolo = 0.0
    tau = V / vo  #space time
    CM, CH, CX, CMe, CT = C
    k1 = 55.20  # (ft^3/lbmol)^0.5/h
    k2 = 30.20  # (ft^3/lbmol)^0.5/h
    r1m = -k1 * CM * CH**0.5
    r2t = k2 * CX * CH**0.5
```

```
    rM = r1m
    rH = r1m - r2t
    rX = -r1m - r2t
    rMe = -r1m + r2t
    rT = r2t
    return [tau * (-rM) - Cmo + CM,
            tau * (-rH) - Cho + CH,
            tau * (-rX) - Cxo + CX,
            tau * (-rMe) - Cmeo + CMe,
            tau * (-rT) - Ctolo + CT]
initGuesses = [0.002, 0.002, 0.002, 0.002, 0.002]
from scipy.optimize import fsolve
exitC = fsolve(funcC, initGuesses)
species = ['M', 'H', 'X', 'Me', 'T']
for s,C in zip(species, exitC):
    print('{0:^3s}{1:1.5f} lbmol/ft^3'.format(s,C))
```

Multiple Reaction in PFR

M + H = X + Me; X+H = T + Me

```
def dFdV(F, t):
    'PFR moe balances'
    Ft = F.sum()
    v = vo * Ft / Fto
    C = F / v
    CM, CH, CX, CMe, CT = C
    k1 = 55.20
    k2 = 30.20
    r1m = -k1 * CM * CH**0.5
    r2t = k2 * CX * CH**0.5
    rM = r1m
    rH = r1m - r2t
    rX = -r1m - r2t
    rMe = -r1m + r2t
    rT = r2t
    dFMdV = rM
    dFHdV = rH
    dFXdV = rX
    dFMedV = rMe
    dFTdV = rT
    return [ dFMdV, dFHdV, dFXdV, dFMedV,
dFTdV ]
Finit = [Fmo, Fho, Fxo, Fmeo, Ftolo]
Vspan = np.linspace(0.0, 238.0)
sol = odeint(dFdV, Finit, Vspan)
Ft = sol.sum(axis=1)  # sum each row
v = vo * Ft / Fto
FM  = sol[:,0]
FH  = sol[:,1]
FX  = sol[:,2]
FMe = sol[:,3]
FT  = sol[:,4]
F1,F2,F3,F4,F5 = sol.T
```

Linear Regression

$y = X p$

$p = (X^T X)^{-1} X^T y$

e = y – Xp; Summed squared error SSE = e.e

$SST = (y-\bar{y})^2$

R^2 =1-SSE/SST

```
import numpy as np
x = np.array([0, 0.5, 1, 1.5, 2.0, 3.0, 4.0, 6.0, 10])
y = np.array([0, -0.157, -0.315, -0.472, -0.629, -
0.942, -1.255, -1.884, -3.147])
X = np.column_stack([x, x**0])
XTX = np.dot(X.T, X)
XTy = np.dot(X.T, y)
p = np.dot(np.linalg.inv(XTX), XTy)
slope, intercept = p
print('The slope is {0} \nand intercept is
{1}'.format(slope, intercept))
e = y – np.dot(X,p)
SSE = np.dot(e,e)
yb = y – np.mean(y)
SST = np.dot(yb,yb)
R2 = 1-SSE/SST
```

Uncertainty & Interval Needed: Regress:

```
from pycse import regress
alpha = 1-0.95
p,pint,se = regress(X,y,alpha)
```

Rate Law determined:

```
import numpy as np
np.set_printoptions(precision=3
from pycse import deriv, regress
import matplotlib.pyplot as plt
data = np.array(data)
t = data[:, 0]   # column 0
Ca = data[:, 1]  # column 1
dCadt = deriv(t, Ca)
x = np.log(Ca)
y = np.log(-dCadt)
X = np.column_stack([x**0, x])
p, pint, se = regress(X, y, 0.05)
intercept_range = pint[0]
alpha_range = pint[1]
k = np.exp(intercept_range)
```

Polynomial Fit:

```
import numpy as np
```

```
np.set_printoptions(precision=3)
from pycse import regress
import matplotlib.pyplot as plt
data = np.array(data)
t = data[:, 0]
Ca = data[:, 1]
pCa = np.polyfit(t, Ca, 4)
fCa = np.polyval(pCa, t)
dCadt = np.polyval(np.polyder(pCa), t)
# ln(-dCa/dt) = alpha ln(Ca) + ln(k)
x = np.log(Ca)
y = np.log(-dCadt)
X = np.column_stack([x**0, x])
p, pint, se = regress(X, y, 0.05)
intercept_range = pint[0]
alpha_range = pint[1]
k = np.exp(intercept_range)
```

Nonlinear Regression

Curve-fit (No uncertainty)

```
x = np.array([0.5, 0.387, 0.24, 0.136, 0.04, 0.011])
y = np.array([1.255, 1.25, 1.189, 1.124, 0.783,
0.402])
def fit(x,a,b): # y = f(x,a,b)
    return a*x/(x+b)
guess=[1,1];
pars,pcov = curve_fit(fit,x,y,guess)
print pars
xfit = np.linspace(min(x),max(x))
a,b = pars
yfit = fit(xfit,*pars)
```

nlinfit (uncertainty provided)

```
import numpy as np
np.set_printoptions(precision=3)
from pycse import nlinfit

x = np.array([0.5, 0.387, 0.24, 0.136, 0.04, 0.011])
y = np.array([1.255, 1.25, 1.189, 1.124, 0.783,
0.402])
def func(x, a, b):
    return a * x / (b + x)
initial_guess = [1.2, 0.03]
alpha = 0.05
pars, pint, se = nlinfit(func, x, y, initial_guess, alpha)
aint, bint = np.array(pint)
```

Fit with odeint

```
t = np.array([0, 50, 100, 150, 200, 250, 300])
Ca = np.array([0.05, 0.038, 0.0306, 0.0256, 0.0222,
0.0195, 0.0174])
Ca0 = 0.05

def dCadt(Ca,t,k,alpha):
return -k*Ca**alpha
def myfun(t,k,alpha):
Ca = odeint(dCadt,Ca0,t,args=(k,alpha))
return Ca[:,0]
guess =[0.1,2.0]
p,pint,se = nlinfit(myfun,t,Ca,guess,0.05)
k_range,alpha_range = np.array(pint)
```

Uncertainty

Monte Carlo:

```
import numpy as np
from scipy.optimize import fsolve
N = 10000 ;  V =  66000   # L
Fa0 = np.random.normal(5, 0.05, N)
v0 = np.random.normal(10.0, 0.1, N)
k = np.random.normal(3.0, 0.2, N)
SOL = np.empty(k.shape)

for i in range(N):
   def func(Ca):
      ra = -k[i] * Ca**2
      return Fa0[i] - v0[i] * Ca + V * ra
   guess = 0.1 * Fa0[i] / v0[i]
   SOL[i] = fsolve(func, guess)[0]
print('Ca(exit) = {0}+/-{1}'.format(np.mean(SOL),
np.std(SOL)))
```

Wrap

```
import uncertainties as u
from scipy.optimize import fsolve
V = 66000  # reactor volume L^3
Fa0 = u.ufloat(5.0, 0.05)     # mol / h
v0 = u.ufloat(10., 0.1)       # L / h
k = u.ufloat(3.0, 0.2)      # rate constant L/mol/h
def func(Ca, v0, k, Fa0, V):
   Fa = v0 * Ca     # exit molar flow of A
   ra = -k * Ca**2  # rate of reaction of A L/mol/h
   return Fa0 - Fa + V * ra
def Ca_solve(v0, k, Fa0, V):
   guess = 0.1 * Fa0 / v0
   sol = fsolve(func, guess, args=(v0, k, Fa0, V))[0]
   return sol
Ca_exit = u.wrap(Ca_solve)(v0, k, Fa0, V)
print('The exit concentration is {0}'.format(Ca_exit))
Ca_exit.nominal_value
Ca_exit.std_dev
```

Mechanisms

Cost

Total cost = Operating cost [$/V/T] + Feedstocks cost [$/mol]

Given Fy, Costr, Costx, ra, Cx0; Costmin?

Unknowns: v0, V: 0 = Fx0 – Cx*v0-rx*V

```
from scipy.optimize import fsolve
import numpy as np

k = 0.1    # rate constant 1/min
Cx0 = 1.5  # initial concentration
Fy = 90.0  # exit molar flow of Y
def objective(V, v0):
    Fx0 = Cx0 * v0
    Fx = Fx0 - Fy
    Cx = Fx / v0
    rx = -k * Cx
    return Fx0 - Fx + rx * V
v0 = np.linspace(66, 200)
reactor_cost = 0.1  # $/m^3
Xcost = 1.20        # $/kmol
@np.vectorize # elementally operation
def cost(v0):
    V, = fsolve(objective, 10000, args=(v0,))
    CR = reactor_cost * V
    CX = v0 * Cx0 * Xcost
    return (CR + CX) / Fy  # cost in $/mol Y
min(cost(v0))
np.argmin(cost(v0)) # index
print v0[np.argmin(cost(v0))]
from scipy.optimize import fmin
v_opt, = fmin(cost, 120)
```

Profit

Profit = Value – Cost

Given: Cx0, v0, rx, ValueY [$/mol], Cost of reactor operation [$/T/V] PFR; max profit?

Fyexit vs V – profit

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
k = 30.0
CX0 = 2.5 # kmol / m^3
v0 = 12.0 # m^3 / min
FX0 = CX0 * v0
FY0 = 0.0
def dFdV(F, V):
    Fx, Fy = F
    Cx = Fx / v0
    r = k * Cx
    rx = -r
    ry =  r
    dFxdV = rx
    dFydV = ry
    return [dFxdV, dFydV]
Vspan = np.linspace(0, 3)
sol = odeint(dFdV, [FX0, FY0], Vspan)
Fy = sol[:, 1]
V_Y = 1.50 # $/kmol
C_R = 2.50 # $/m^3/min
product_value = Fy * V_Y
operating_cost = Vspan * C_R
profit = product_value - operating_cost
```

fmin

```
def profit(V, sign=1): # the default value of sign is 1
    Vspan = np.linspace(0, V)
    sol = odeint(dFdV, [FX0, FY0], Vspan)
    Fx, Fy = sol[-1] # at exit

    product_value = Fy * V_Y
    operating_cost = V * C_R

    profit = product_value - operating_cost
    return profit * sign

from scipy.optimize import fmin

V_opt, = fmin(profit, 1.5, args=(-1,)) # tuple

print('The maximum profit is ${0:1.2f} at V={1:1.2f}
m^3'.format(profit(V_opt), V_opt))
```

Internal Effectiveness Factors

$D_e < D_{AB}$ (m^2/s)

d^2Ca/dr^2 + 2/r * dCa/dr – k/De*Ca^2 =0

```
De = 0.1        R = 0.5
k = 6.4         Cas = 0.2

def ode(Y,r):
Wa = Y[0]
Ca = Y[1]
if r == 0:
dWadr = 0
else:
dWadr = -2*Wa/r + k/De*Ca
dCadr = Wa
return [dWadr,dCadr]
Wa0 = 0
rspan = np.linspace(0,R,500)
def obj(Ca0):
Y = odeint(ode,[Wa0,Ca0],rspan)
Ca = Y[:,1]
return Ca[-1]-Cas
ans, = fsolve(obj,0.1)
print ans
Y = odeint(ode,[Wa0,ans],rspan)
Ca = Y[:,1]
eta_numerical =
(np.trapz(k*Ca*4*np.pi*(rspan**2),rspan)/np.trapz(k
*Cas*4*np.pi*(rspan**2),rspan))
```

$\Phi = \sqrt{\frac{ka^2}{D_A}}$ Reaction rate/ Diffusion Rate

$$C(r) = \frac{3}{r}\frac{\sinh \Phi r}{\sinh 3\Phi}$$

$$\eta = \frac{1}{\Phi}\left[\frac{1}{\tanh 3\Phi} - \frac{1}{3\Phi}\right]$$

Different Shapes :

Similar to sphere

Different Order:

d^2/dr^2 = -2/r* dc/dr + Φ^2c^n

BVP_nl

```
from pycse import BVP_nl

n = [1.0, 2.0, 3.0]
R = 3.0
r1 = 0
r2 = R
N = 300
Rbar = np.linspace(r1,r2,N)
THI = np.logspace(-2,3,20)

for order in n:
ETA = []
c0 = 0.9
p =4
init = c0 + (1-c0)/R**p *Rbar **p
for thi in THI:
def F(rbar, cbar, dcbardrbar):
return -2.0/rbar*dcbardrbar + thi**2*cbar**order
def BCS(rbar,cbar):
return [(cbar[1]-cbar[0])/(rbar[-1]-rbar[0]),
cbar[-1]-1]
Cbar = BVP_nl(F,Rbar,BCS,init)
eta = 1./9 *np.trapz(Cbar**order*Rbar**2,Rbar)
ETA +=[eta]
plt.loglog(THI,ETA,label ='n={0}'.format(order))
```

Non-isothermal

$$\upsilon = \upsilon_0 \frac{F_t}{F_{t0}}\frac{P_o}{P}\frac{T}{T_0}$$

$$\frac{dlnK}{dT} = \frac{\Delta H}{RT^2}$$

```
dH = -2000.0
K =1.0
R = 8.314

def dlnKdT(lnK,T,dH):
return dH/R/T**2
Tspan1 = np.linspace(298,1000)
lnK1 = odeint (dlnKdT,np.log(K),Tspan1,args =
(dH,))
Tspan2 = np.linspace(298,100)
lnK2 = odeint(dlnKdT,np.log(K),Tspan2,args =
(dH,))

Tspan = np.concatenate([Tspan2[::-1],Tspan1])
lnK = np.concatenate([lnK2[::-1],lnK1])
```

Equilibrium Constant:

$$K(T) = K_1(T)\exp\left[\frac{-\Delta H(T_1)}{R}\left(\frac{1}{T} - \frac{1}{T_1}\right)\right]$$

Rate Constant:

$$k(T) = k(T_0)\exp\left[\frac{-E}{R}\left(\frac{1}{T} - \frac{1}{T_0}\right)\right]$$

Non-isothermal Batch

Adiabatic batch:

$$\frac{dT}{dt} = \frac{-\Delta H_R r V}{\sum N_i C_{p,i}}$$

```
V = 1200.0  #L
T0 = 300.15
Ca0 = Cb0 = 2.0
Cc0 = 0.0
Cpa = Cpb = 20.0
Cpc = 40.0   # cal/mol/K

k0 = 0.01725  # L/mol/min
E = 1500.0     # cal/mol
R = 1.987      # cal/mol/K
dH = -10000.0

def batch(N,t):
Na,Nb,Nc,T=N
Ca = Na/V
Cb = Nb/V
k = k0*np.exp(-E/R*(1.0/T - 1.0/T0))
r = k*Ca*Cb
dNadt = -r*V
dNbdt = -r*V
dNcdt= r*V
dTdt = -dH*r*V/(Na*Cpa+Nb*Cpb+Nc*Cpc)
return [dNadt,dNbdt,dNcdt,dTdt]

N0 =[Ca0*V,Cb0*V,0,T0]
tspan = np.linspace(0,200)
sol = odeint(batch,N0,tspan)
Na,Nb,Nc,T = sol.T
```

With Heat Exchange

$$\frac{dT}{dt} = \frac{-\Delta H_R r V + Ua(T_{coolant} - T)}{\sum N_i C_{p,i}}$$

```
dTdt =(-dH*r*V+Ua*(Tcoolant-
T))/(Na*Cpa+Nb*Cpb+Nc*Cpc)
```

Non-isothermal-CSTR

$$\sum N_j \, C_{p,j} \frac{dT}{dt} = \sum F_{j0} C_{p,j} (T_0 - T) - \Delta H_{rx} r V$$

Steady State

$$0 = \sum F_{j0} C_{p,j} (T_0 - T) - \Delta H_{rx} r V$$

Given conversion, V, Texit?

```
R = 8.314e-3       Hrx = -6.9
Tfeed = 330.0
k1 = 31.1          T1 = 360.0
E = 65.7
Kc1 = 3.03         T2 = 273.15+60

Ca0 = 9300.0
Ft0 = 163000.0
Fa0 = 0.9*Ft0
Fb0 = 0
Fi0 = 0.1*Ft0

Cpa = 0.141
Cpb = 0.141
dCp = Cpb-Cpa

Cpi = 0.161

X = 0.7
Ca = Ca0*(1.0-X)
Cb = Ca0*X

Fa = Fa0*(1-X)

def obj(Y):
    V,T = Y
    Hrxn = Hrx+dCp*(T-Tfeed)
    k = k1*np.exp(-E/R*(1.0/T - 1.0/T1))
    Kc = Kc1*np.exp(-Hrxn/R*(1.0/T-1.0/T2))
    r = k*(Ca-Cb/Kc)
    ra = -r
    # mole balance
    z1 = Fa0-Fa+ra*V
    # energy balance
    z2 = (Fa0*Cpa + Fb0*Cpb +Fi0*Cpi)*(Tfeed-
T)+(-Hrxn*r*V)
    return [z1, z2]
V,Texit = fsolve(obj,[16,360])
```

Multiple Steady States

Given T, Two conversions from balances

```
V = 40.1
v0 = (233.1+2*46.62)
Fa0 = 43.04
Fb0 = 802.0
Fm0 = 71.87

Ca0 = Fa0/v0

A = 16.96e12
E = 32400.0
R = 1.987
```

```
Cpa = 35.0          # BTU/(lbmol*R)
Cpb = 18.0          # BTU/(lbmol*R)
Cpc = 46.0          # BTU/(lbmol*R)
Cpm = 19.5          # BTU/(lbmol*R)
Ha = -66600.0       # BTU/(lbmol)
Hb = -123000.0      # BTU/(lbmol)
Hc = -226000.0      # BTU/(lbmol)
Tr = 527.67         # reference temperature for
enthalpy in R
Hrx_TR = Hc - Hb - Ha
deltaCp = Cpc - Cpa - Cpb

Tfeed = 534.0       # Feed temperature in Rankine

Tspan = np.linspace(Tfeed, Tfeed + 100.0) #
temperature in Rankine

XMB = np.empty(Tspan.shape)
XEB = np.empty(Tspan.shape)

for i,T in enumerate(Tspan):
    k = A*np.exp(-E/(R*T))
    Hrx = Hrx_TR + deltaCp * (T-Tr)
    def MB(X):
        Ca = Ca0*(1.0-X)
        r = k*Ca
        ra = -r
        z = Fa0-v0*Ca +ra*V
        return z
    XMB[i], = fsolve(MB,0.1)
    def EB(X):
        Ca = Ca0*(1.0-X)
        Fa = v0*Ca
        rV = (Fa-Fa0)/(-1)
        z = ((Fa0*Cpa+Fb0*Cpb+Fm0*Cpm)*(Tfeed-
T)+(-Hrx*rV))
        return z
    XEB[i], = fsolve(EB,0.91)
```

Transient Adiabatic CSTR

$$\frac{dT}{dt} = \frac{\sum F_{j0} C_{p,j} (T_0 - T) - \Delta H_{rx} rV}{\sum N_j \, C_{p,j}}$$

```
V = 40.1
v0 = (233.1+2*46.62)
Fa0 = 43.04
Fb0 = 802.8
Fc0 = 0.0
Fm0 = 71.87

Ca0 = Fa0/v0
Cm = Fm0/v0

A = 16.96e12
E = 32400.0
R = 1.987

Cpa = 35.0          # BTU/(lbmol*R)
Cpb = 18.0          # BTU/(lbmol*R)
Cpc = 46.0          # BTU/(lbmol*R)
Cpm = 19.5          # BTU/(lbmol*R)
Ha = -66600.0       # BTU/(lbmol)
Hb = -123000.0      # BTU/(lbmol)
Hc = -226000.0      # BTU/(lbmol)
Tr = 527.67         # reference temperature for
enthalpy in R
Hrx_TR = Hc - Hb - Ha
deltaCp = Cpc - Cpa - Cpb

def cstr(Y,t,Tfeed):
    Na,Nb,Nc,T = Y
    k = A*np.exp(-E/R/T)
    Ca = Na/V
    Cb = Nb/V
    Cc = Nc/V

    r = k*Ca
    ra = -r
    rb = -r
    rc = r

    Fa = Ca*v0
    Fb = Cb*v0
    Fc = Cc*v0

    dNadt = Fa0-Fa+ra*V
    dNbdt = Fb0-Fb+rb*V
    dNcdt = Fc0-Fc+rc*V

    Hrx = Hrx_TR+deltaCp*(T-Tr)

    nCp = V*(Ca*Cpa+Cb*Cpb+Cc*Cpc+Cm*Cpm)
    dTdt = ((Fa0*Cpa+Fb0*Cpb+Fm0*Cpm)*(Tfeed-
T)-Hrx*r*V)/nCp

    return [dNadt,dNbdt,dNcdt,dTdt]

Y0 = [0,V*3.45,0,530]
tspan = np.linspace(0,25,500)
for Tfeed in np.linspace(525,535,20):
    sol = odeint(cstr,Y0,tspan,args = (Tfeed,))
    #X = (Ca0-sol[:,0]/V)/Ca0
    X = (Fa0-sol[:,0])/Fa0
    T = sol[:,-1]
```

Stability

Heat Removal:

$$R(T) = \sum F_{j0} C_{p,j} (T_0 - T)$$

Heat Generated:

$$G(T) = -\Delta H_{rx} r V$$

$$r = kC_A$$

$$0 = C_{A0} - C_A - kC_A\tau$$

$$C_A = \frac{C_{A0}}{1 + k(T)\tau}$$

$$G(T) = -\Delta H_{rx} r V = \frac{-\Delta H_{rx} k(T) C_{A0} V}{1 + k(T)\tau}$$

Non-isothermal PFR

$$\frac{dT}{dV} = \frac{-\Delta H_{Rx}(T) r}{\sum F_i \, C_{p,i}}$$

With constant T heat exchanger:

$$\frac{dT}{dV} = \frac{-\Delta H_{Rx}(T) r + Ua(T_a - T)}{\sum F_i \, C_{p,i}}$$

Shell and tube concurrent heat exchanger:

$$Q = Ua(T_{shell} - T)$$

$$\frac{dT}{dV} = \frac{-\Delta H_{Rx}(T) r + Q}{\sum F_i \, C_{p,i}}$$

$$\frac{dT_{shell}}{dV} = \frac{-Q}{mC_{pcoolant}}$$

Countercurrent:

$$\frac{dT_{shell}}{dV} = \frac{Q}{mC_{pcoolant}}$$

Reversible Reaction:

$$X = \frac{K}{1 + K}$$

$$X = \frac{C_p(T_{feed} - T)}{\Delta H}$$