

1 Lecture 0: Intro to Reaction Engineering

1. Reaction engineering

Understanding, modeling, designing, using, controlling, analyzing, improving anything in which chemical reactions happen.

1. Reaction engineering applications

(a) Traditional

- i. Industrial chemical/petroleum processes
- ii. Fine chemical/pharmaceutical processes
- iii. Emerging, eg biorefinery, shale gas, <http://cistar.us>

(b) Energy storage, batteries, fuel cells

(c) Environmental systems

- i. Atmosphere, lake, bioreactor (water purification), catalytic convertor

(d) Biological systems

- i. Cell, organ, body

(e) Laboratory reactors - interrogate, quantify

(f) Research - improved materials (catalysts), improved processes, understand limitations

- i. Sabatier plot, <https://doi.org/10.1038/nchem.121>

2. Course structure

(a) Quantifying chemical reactions

- i. Stoichiometry
- ii. Thermodynamics - heat flow, direction, equilibrium
- iii. Kinetics - rates, mechanisms

(b) Physical/chemical interactions

- i. Transport, mixing, diffusion resistance, ...

(c) Chemical reactors

- i. Ideal 0 and 1-dimensional
- ii. Non-ideal
- iii. Non-isothermal
- iv. Non-steady state
- v. Multiphase

(d) Chemical processes (beyond us)

(e) Markets (beyond us)

2 Stoichiometry and reactions

1. Substances
2. Amounts
 - (a) mass, moles, volumes
 - (b) flow rates
3. compositions
 - (a) amount/total amount
4. Reactions and stoichiometric coefficients
 - (a) Advancements $n_j = \sum_i \nu_{ij} \xi_i$
 - (b) Limiting reagents

3 Chemical thermodynamics and equilibria

1. Chemical reactions $\sum_j \nu_j A_j = 0$
2. Thermodynamic potential differences
 - (a) Standard states
 - (b) Formation reactions
 - (c) Reaction enthalpy $\Delta H^\circ(T) = \sum_j \nu_j H_j^\circ(T) = \sum_j \nu_j H_{f,j}^\circ(T)$
 - (d) Reaction entropy $\Delta S^\circ(T) = \sum_j \nu_j S_j^\circ(T)$
3. Equilibrium-closed system
 - (a) Free energy vs reaction advancement, $G(\xi, T) = \sum_j n_j \mu_j = \sum_j (n_{j0} + \nu_j \xi) (\mu_j^\circ(T) + RT \ln a(\xi, T))$
 - (b) Equilibrium $(\partial G / \partial \xi)_{T,P} = 0$
 - (c) Equilibrium constants and algebraic solutions
 - (d) Multiple reactions
4. Le'Chatelier principle - system at equilibrium responds to oppose any perturbation
 - (a) Pressure, composition
 - (b) Temperature: Gibbs-Helmholtz and van't Hoff
5. Equilibrium-open system
 - (a) Reaction phase diagrams, see <http://pubs.acs.org/doi/abs/10.1021/jacs.6b02651> for an example
 - (b) Electrochemical reactions
6. The molecular interpretation
7. Non-ideal activities
8. Surface adsorption
 - (a) Langmuir

4 Empirical kinetics

1. rates: number per unit time per unit something
2. reactor mass balance
3. rate expressions, Functions of T , P , composition C_i
4. rate orders
5. apparent orders
6. integrated rate expressions
7. temperature and Arrhenius expression, $k = Ae^{-E_a/k_B T}$
 - (a) Arrhenius plot, $\ln k$ vs $1/T$

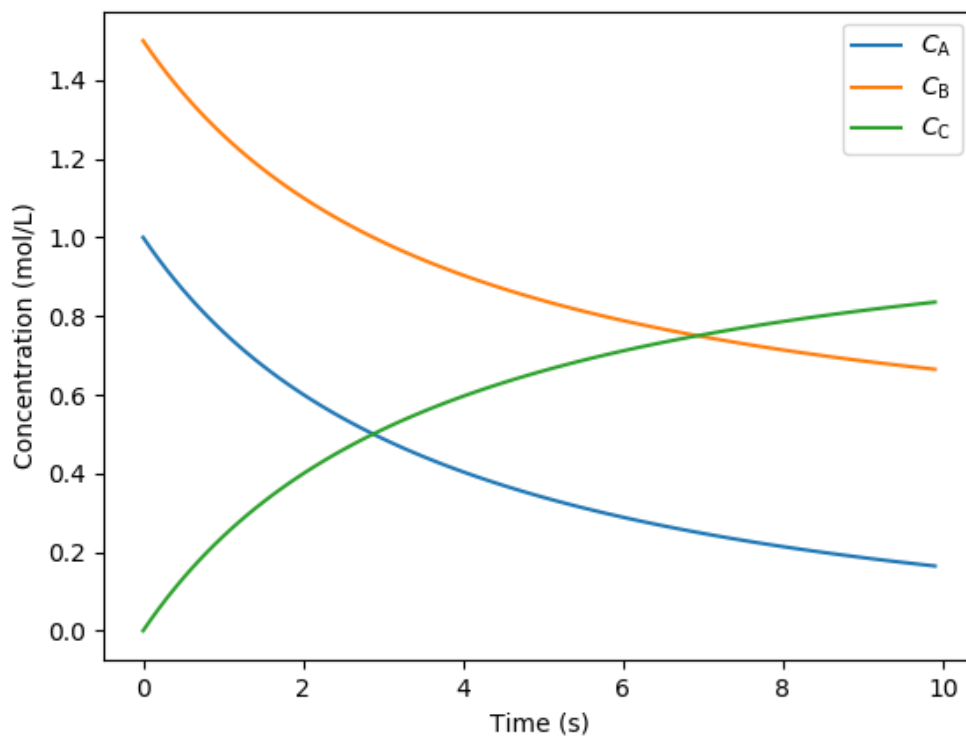
Table 1: Basic kinetic rate laws

	differential rate	integrated rate	half-life
First order	$r = kC_A$	$C_A = C_{A0}e^{-k\tau}$	$\ln 2/k$
Second order	$r = kC_A^2$	$1/C_A = 1/C_{A0} + k\tau$	$1/kC_{A0}$

```

1 import numpy as np           #this lets up handles arrays of data
2 import matplotlib.pyplot as plt
3 from scipy.integrate import odeint, solve_ivp
4
5 def dCdt(C,t,k):
6     dC_Adt = -k*C[0]*C[1]    # A + B -> C;  r = k CA CB
7     dC_Bdt = -k*C[0]*C[1]
8     dC_Cdt =  k*C[0]*C[1]
9     dCdt = [dC_Adt,dC_Bdt,dC_Cdt]
10    return dCdt
11
12 # initialize concentrations
13 C_0 = [1., 1.5, 0.]
14
15 # initialize k's
16 k = 0.2
17
18 # Range of time to solve over
19 t = np.arange(0,10,0.1)
20 t_span = (0., 10.)
21
22 p = (k,) # turn parameters into a tuple
23 # Solve two ODEs with odeint
24 #C = solve_ivp(dCdt,t_span,C_0,p,method='LSODA')
25 C = odeint(dCdt,C_0,t,p)
26
27 C_A = C.transpose()[0] # Get C_A from C
28 C_B = C.transpose()[1] # Get C_B from C
29 C_C = C.transpose()[2]
30 plt.figure()
31 plt.plot(t,C_A,'-',label=r'$C_{\rm A}$')
32 plt.plot(t,C_B,'-',label=r'$C_{\rm B}$')
33 plt.plot(t,C_C,'-',label=r'$C_{\rm C}$')
34 plt.xlabel('Time (s)')
35 plt.ylabel('Concentration (mol/L)')
36 plt.legend()
37 plt.savefig('./conc.png')

```



5 Analyzing reactor data

1. Differential methods
 - (a) Measuring rates
2. Integral methods
3. Half-lives

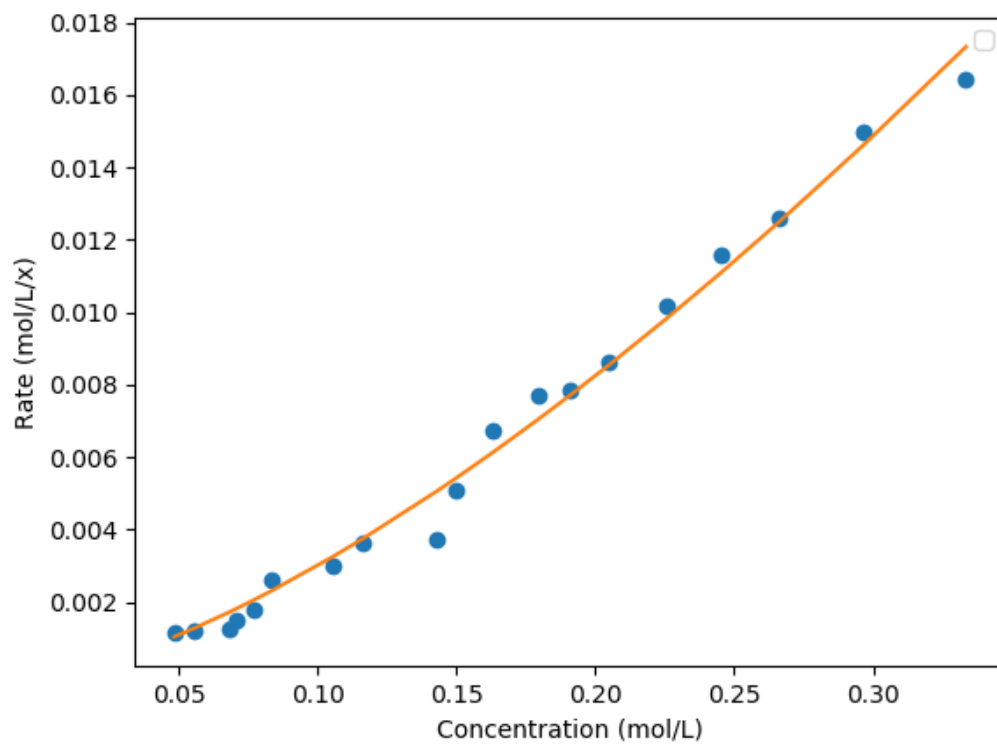
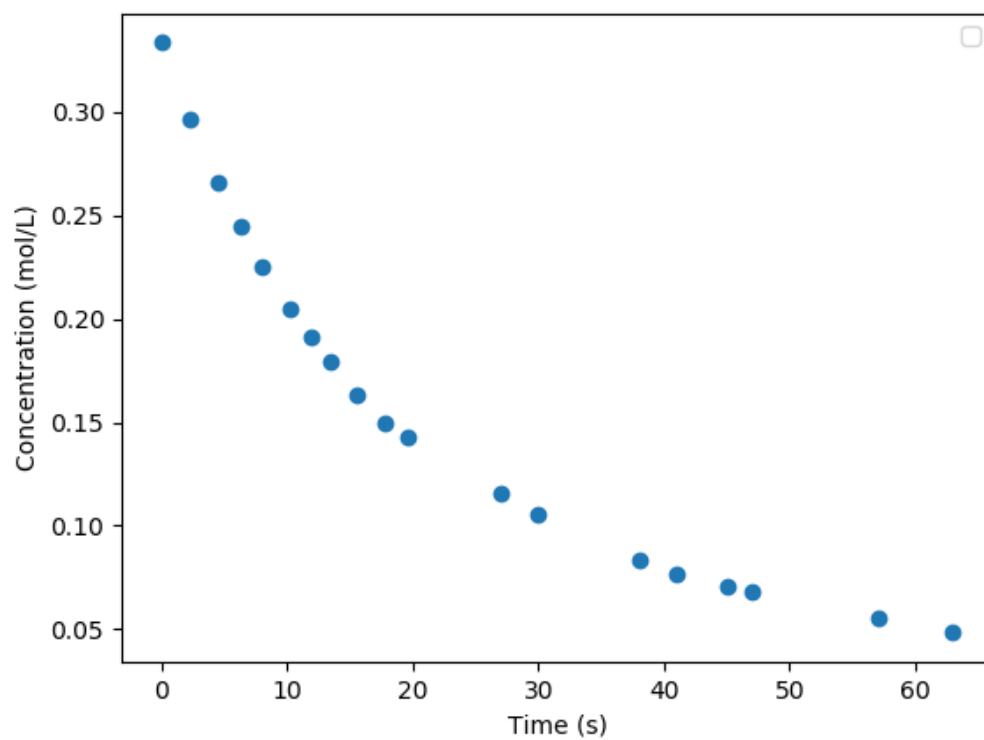
```

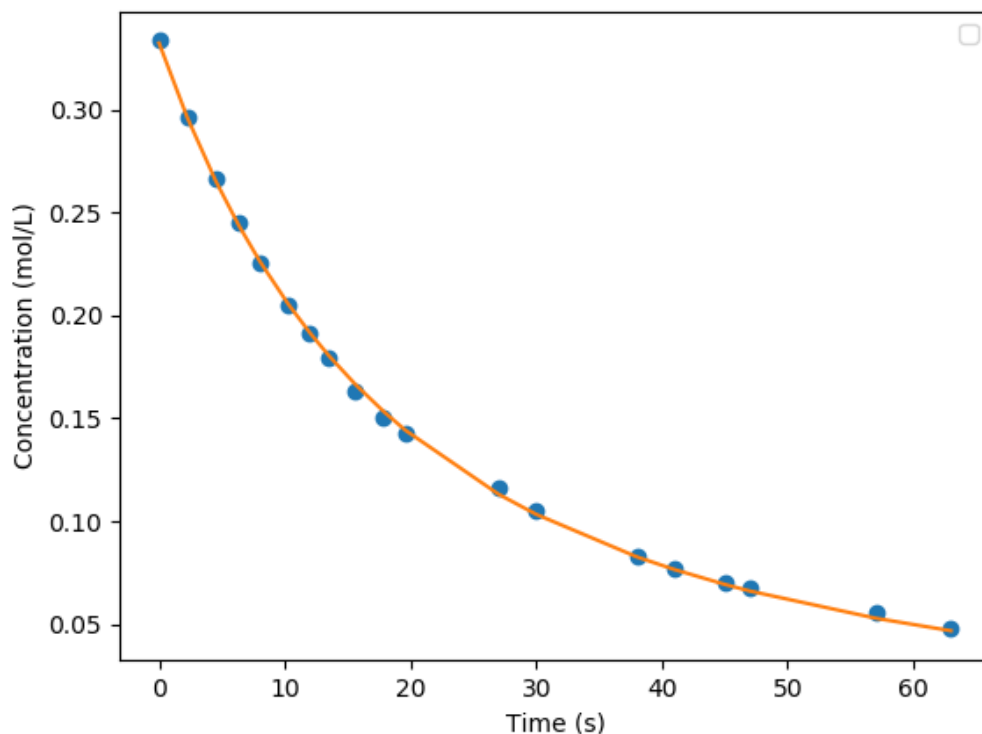
1  import numpy as np                #this lets up handles arrays of data
2  import matplotlib.pyplot as plt
3  from scipy.optimize import curve_fit
4
5  def differential(x, k, alpha):
6      return k*x**alpha
7
8  def integral(t, a, b):
9      return (2*a/(2+a*b*t))**2
10
11  t = np.array([0.00, 2.25, 4.50, 6.33, 8.00, 10.25, 12.00, 13.50, 15.60, 17.85, 19.60, 27.00, 30.00, 38.00, 41.00, 45.00, 47.00,
12
13  C_Br2 = np.array([0.3335, 0.2965, 0.2660, 0.2450, 0.2255, 0.2050, 0.1910, 0.1794, 0.1632, 0.1500, 0.1429, 0.1160, 0.1053, 0.0830,
14
15  plt.figure()
16  plt.plot(t, C_Br2, 'o')
17  plt.xlabel('Time (s)')
18  plt.ylabel('Concentration (mol/L)')
19  plt.legend()
20  plt.savefig('./xylene-conc.png')

```

```
21
22 delta_t = np.ediff1d(t)          # finite difference between adjacent points
23 delta_C = np.ediff1d(C_Br2)
24
25 grad_t = np.gradient(t)          # second order approximation to gradient, allowing for unequal step size
26 grad_C = np.gradient(C_Br2)
27 rate = -np.divide(grad_C, grad_t)
28
29 plt.figure()
30 plt.plot(C_Br2, rate, 'o')
31 plt.xlabel('Concentration (mol/L)')
32 plt.ylabel('Rate (mol/L/x)')
33 plt.legend()
34
35 popt, pcov = curve_fit(differential, C_Br2, rate )
36
37 print('k = {0:f}, alpha={1:f}'.format(popt[0], popt[1]))
38
39 model = differential(C_Br2, popt[0], popt[1])
40 plt.plot(C_Br2, model, '-')
41
42 plt.savefig('./xylene-rate.png')
43
44 difference_array = np. subtract(rate, model)
45 squared_array = np. square(difference_array)
46 mse = squared_array. mean()
47 print(mse)
48
49 # Suggests order of 1.5
50 popt1, pcov1 = curve_fit(integral, t, C_Br2)
51 print('k = {0:f}'.format(popt[1]))
52
53 model1 = integral(t, popt1[0], popt1[1])
54
55 plt.figure()
56 plt.plot(t, C_Br2, 'o')
57 plt.plot(t, model1, '-')
58 plt.xlabel('Time (s)')
59 plt.ylabel('Concentration (mol/L)')
60 plt.legend()
61 plt.savefig('./xylene-int-model.png')
```

```
k = 0.085277, alpha=1.450860
2.4942019231742367e-07
k = 1.450860
```

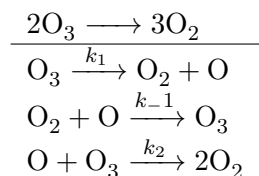




6 Molecular chemical kinetics

6.0.1 Reaction mechanisms

1. Elementary steps and molecularity
2. Ozone decomposition, rate second-order at high P_{O_2} , first-order at low P_{O_2}



3. Detailed balance and microscopic reversibility
4. Equilibrium requirement $K_{eq}(T) = k_f(T)/k_r(T)$
5. Reversibility $r_{net} = r_f(1 - \beta)$, $\beta = Q/K_c = \exp(-\Delta G(T, c_j)/RT)$
6. Collision theory
 - (a) $A + B \rightarrow \text{products}$
 - (b) rate proportional to A/B collision frequency z_{AB} weighted by fraction of collisions with energy $> E_a$

$$r = kC_A C_B, k = \left(\frac{8k_B T}{\pi \mu} \right)^{1/2} \sigma_{AB} N_{av} e^{-E_a/k_B T}$$

- (c) upper bound on real rates

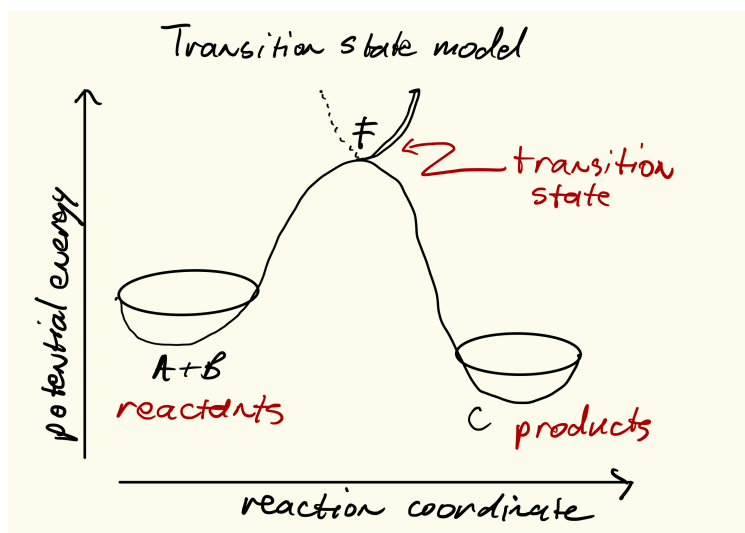
6.0.2 Transition state theory (TST)

1. Assumptions

- (a) Existence of reaction coordinate (PES)
- (b) Existence of dividing surface
- (c) Equilibrium between reactants and “transition state”
- (d) Harmonic approximation for transition state

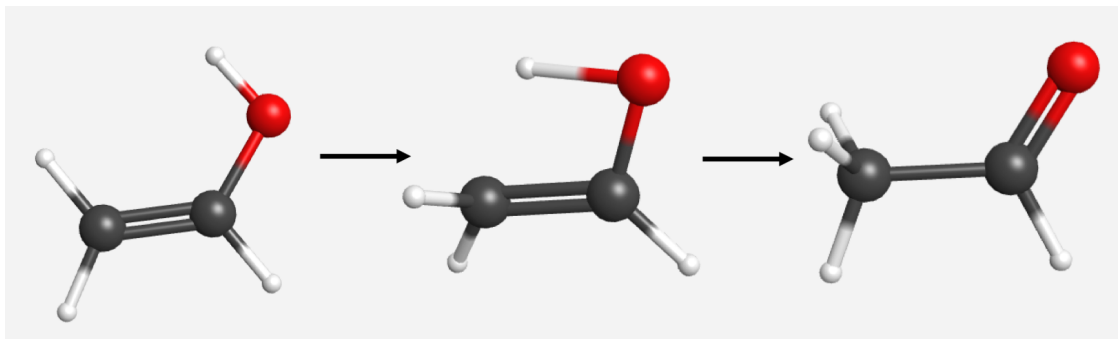
2. rate proportional to concentration of “activated complex” over reactants times crossing frequency

$$\begin{aligned}
 r &= k C_A C_B \\
 &= k^\ddagger C_{AB}^\ddagger \\
 &= \nu^\ddagger K^\ddagger C_A C_B \\
 &= \nu^\ddagger \frac{k_B T}{h \nu^\ddagger} \bar{K}^\ddagger(T) C_A C_B \\
 &= \frac{k_B T}{h} \frac{q^\ddagger(T)}{q_A(T) q_B(T)} e^{-\Delta E(0)/k_B T} C_A C_B
 \end{aligned}$$



6.0.3 Locating transition states computationally

- 1. Reactants/products are minima on potential energy surface
- 2. Transition state is first order saddle point. Unique point on pathway from reactant to product valley
- 3. vinyl alcohol to acetaldehyde example
- 4. <https://www.webmo.net>



6.0.4 Thermodynamic connection

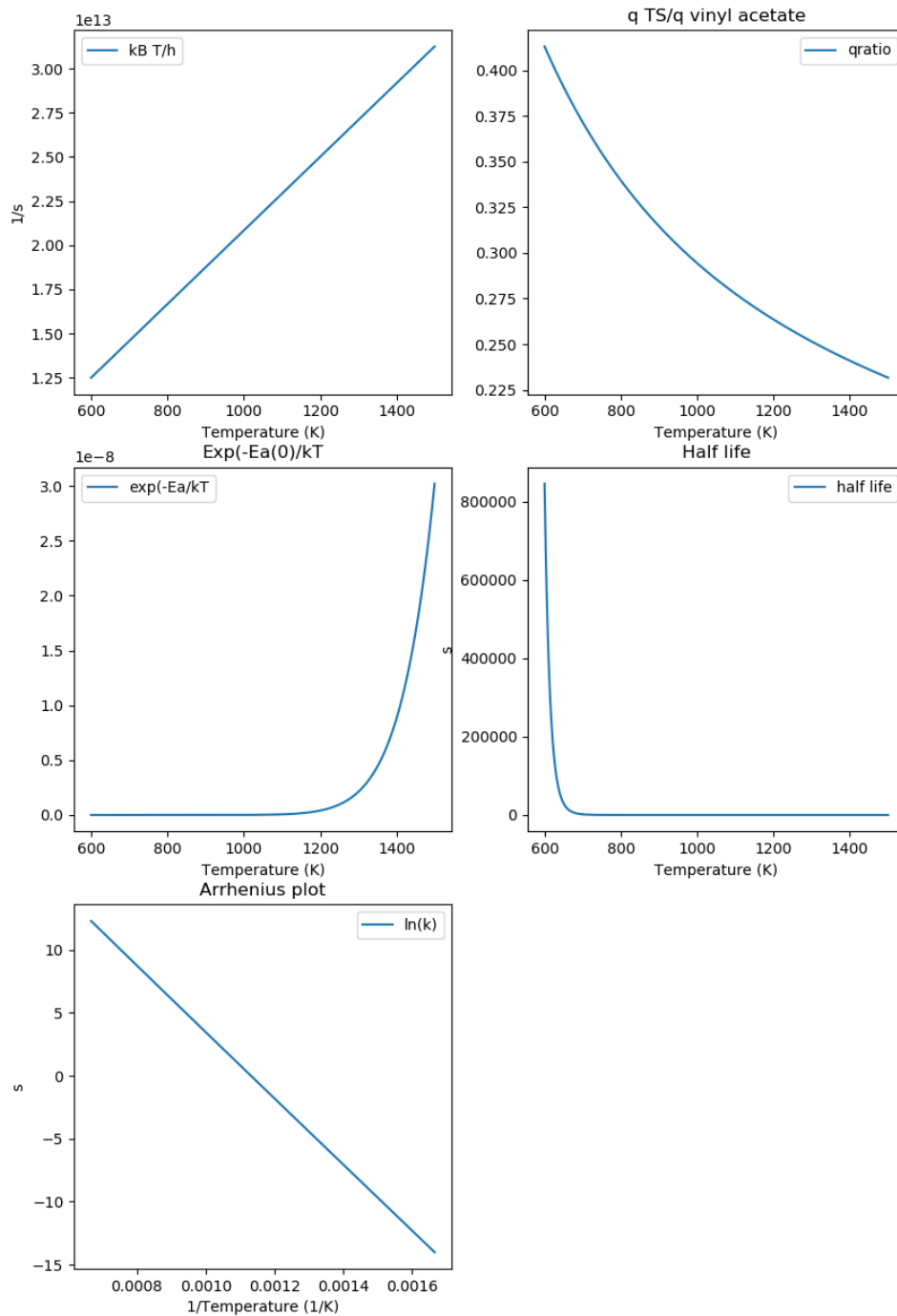
1. Relate activated complex equilibrium constant to activation free energy (isochoric standard state)

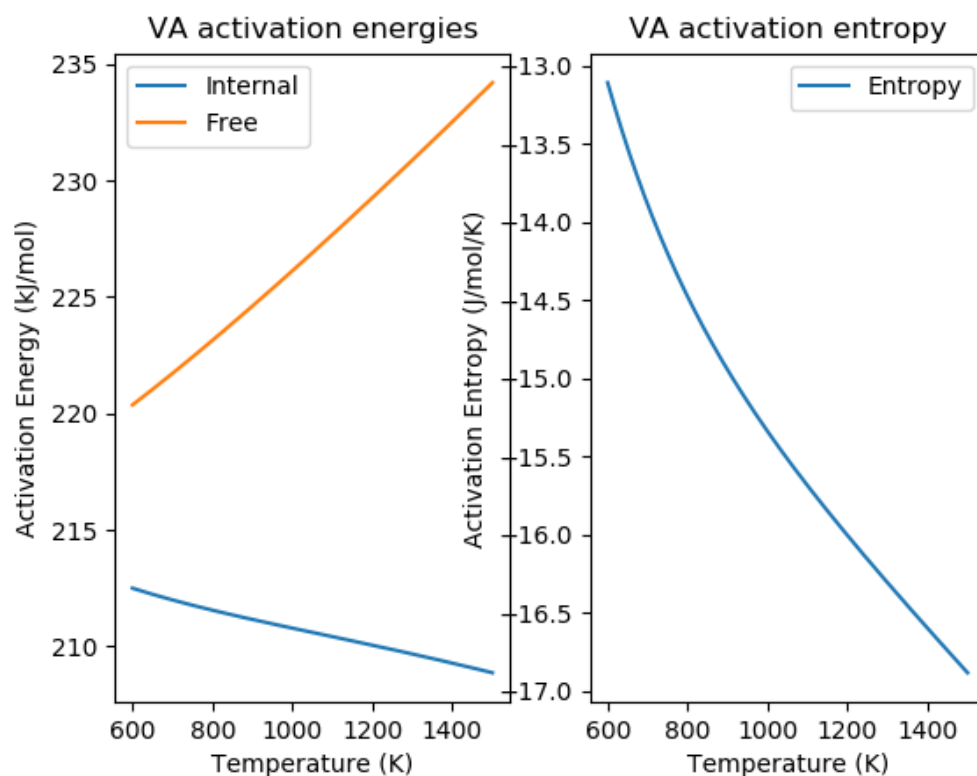
$$\bar{K}^\ddagger(T) = e^{-\Delta A^{\circ\ddagger}(T)/kT} = e^{-\Delta U^{\circ\ddagger}(T)/k_B T} e^{\Delta S^{\circ\ddagger}(T)/k_B}$$

2. Compare to Arrhenius expression

$$E_a = \Delta U^{\circ\ddagger}(T) + kT, A = \frac{k_B T}{h} e^1 e^{\Delta S^{\circ\ddagger}(T)/k_B}$$

Vinyl alcohol to TS 216 kJ/mol
 Delta Uddagger (1000 K) = 211 kJ/mol
 Delta Addagger (1000 K) = 226 kJ/mol
 Delta Sddagger (1000 K) = -1480 J/mol K





6.0.5 Bimolecular reaction

1. Diels-Alder example

```

1 import numpy as np
2
3 kB = 8.61733e-5      # eV /K
4 h = 4.13566766e-15   # eV s
5 eVtokJ = 96.485332
6 Nav = 6.022e23       # Avogadro's number
7 RO = kB * eVtokJ * 1000.    # gas constant in J/mol K
8
9 A = 9.2e6 # liter/mole/second
10 Ea = 99. # kJ/mole
11
12 T = 500.    # K
13
14
15 # 1 M standard state
16 deltaUdd = Ea - RO * T /1000    # kJ /mol
17
18 SS = 1.0 # mol/liter
19 deltaSdd = RO * ( np.log(A/(1./SS)) - np.log(kB * T / h) - 1.)
20
21 deltaAdd = deltaUdd - T * deltaSdd/1000.
22
23 print('1 M standard state, 500 K:')
24 print('Delta Udd ={:4.0f} kJ/mol    Delta Sdd = {:4.0f} J/mol K    Delta Add = {:4.0f} kJ/mol'.format(deltaUdd,deltaSdd,deltaAdd))
25
26 # 1 bar standard state
27 PO = 1.0e5    # 1 bar = 10^5 Pa = 10^5 J/m^3
28 PO = PO / 1e3    # J/l

```

```

29 SS = P0/(R0 * T)    # J/mol / J/l = mol/liter
30 print('1 bar = {:.3f} mol/liter standard state, 500 K'.format(SS))
31
32 deltaHdd = Ea - 2* R0 * T /1000    # kJ /mol
33 deltaSdd = R0 * ( np.log(A/(1./SS)) - np.log(kB * T / h) - 2.)
34
35 deltaGdd = deltaHdd - T * deltaSdd/1000.
36 print('Delta Hdd ={:4.0f} kJ/mol    Delta Sdd = {:4.0f} J/mol K    Delta Gdd = {:4.0f} kJ/mol'.format(deltaHdd,deltaSdd,deltaGdd))

```

1 M standard state, 500 K:

Delta Udd = 95 kJ/mol Delta Sdd = -124 J/mol K Delta Add = 157 kJ/mol

1 bar = 0.024 mol/liter standard state, 500 K

Delta Hdd = 91 kJ/mol Delta Sdd = -164 J/mol K Delta Gdd = 172 kJ/mol

6.0.6 Correlations across reactions

1. early vs late transition states
2. Brønsted-Evans-Polyani relationship

$$E_a = \alpha \Delta H + \beta$$

3. linear free energy relationships between similar reactions (substituent effects)

$$\ln(k_1/k'_1) \propto \ln(K_1/K'_1)$$

4. compensation effect linear correlation across catalysts for the same reaction

$$\Delta H^{\circ\dagger} \propto \Delta S^{\circ\dagger}$$

7 Mechanisms

1. Lindemann-Hinshelwood model for first order reactions
2. Quasi-steady state approximation

$$r_{\text{intermediate}} \approx 0$$

```

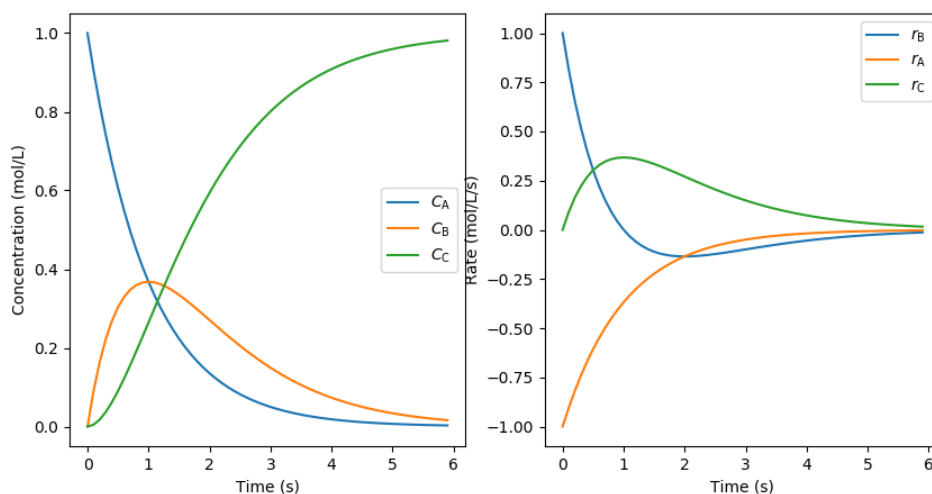
1 import numpy as np                #this lets up handles arrays of data
2 import matplotlib.pyplot as plt
3 from scipy.integrate import odeint, solve_ivp
4
5 def dCdt(C,t,k):
6     dC_Adt = -k[0]*C[0]           # A -> B -> C; r = k CA CB
7     dC_Bdt = k[0]*C[0]-k[1]*C[1]
8     dC_Cdt = k[1]*C[1]
9     dCdt = [dC_Adt,dC_Bdt,dC_Cdt]
10    return dCdt
11
12 # initialize concentrations
13 C_0 = [1., 0, 0.]
14
15 # initialize k's
16 k = [1,1]
17
18 # Range of time to solve over
19 t = np.arange(0,6,0.1)

```

```

20 t_span = (0., 10.)
21
22 p = (k,) # turn parameters into a tuple
23 # Solve two ODEs with odeint
24 #C = solve_ivp(dCdt,t_span,C_0,p,method='LSODA')
25 C = odeint(dCdt,C_0,t,p)
26
27 # would be nice to back compute rates
28 #print(C)
29 #R = dCdt(C,t,k)
30 #rate=np.array(R)
31
32 #print(R)
33 #rate_B = rate.transpose()[1]
34
35 #plt.figure()
36 #plt.plot(t,rate_B)
37 #
38
39 C_A = C.transpose()[0] # Get C_A from C
40 C_B = C.transpose()[1] # Get C_B from C
41 C_C = C.transpose()[2]
42 plt.figure(figsize=(10,5))
43 plt.subplot(1,2,1)
44
45 plt.plot(t,C_A,'-',label=r'$C_{\rm A}$')
46 plt.plot(t,C_B,'-',label=r'$C_{\rm B}$')
47 plt.plot(t,C_C,'-',label=r'$C_{\rm C}$')
48 plt.xlabel('Time (s)')
49 plt.ylabel('Concentration (mol/L)')
50 plt.legend()
51 #plt.savefig('./Images/PSSA.png')
52
53 plt.subplot(1,2,2)
54 plt.plot(t,k[0]*C_A-k[1]*C_B,label=r'$r_{\rm B}$')
55 plt.plot(t,-k[0]*C_A,label=r'$r_{\rm A}$')
56 plt.plot(t,k[1]*C_B,label=r'$r_{\rm C}$')
57 plt.xlabel('Time (s)')
58 plt.ylabel('Rate (mol/L/s)')
59 plt.legend()
60 plt.savefig('./Images/PSSA.png')

```



1. free energy surfaces

2. simplifying reaction networks

```

1  import numpy as np                #this lets up handles arrays of data
2  import matplotlib.pyplot as plt
3  from scipy.integrate import odeint, solve_ivp
4  from scipy.interpolate import BPoly
5
6  kB = 8.61733e-5                  # eV /K
7  h = 4.13566766e-15              # eV s
8
9  def dCdt(C,t,k):
10     dC_Adt = -k[0]*C[0]+k[1]*C[1]      # A -> B -> C;  r = k CA CB
11     dC_Bdt =  k[0]*C[0]-k[1]*C[1] -k[2]*C[1]+k[3]*C[2]
12     dC_Cdt =  k[2]*C[1]-k[3]*C[2]
13     dCdt = [dC_Adt,dC_Bdt,dC_Cdt]
14     return dCdt
15
16 # initialize concentrations
17 C_0 = [1., 0, 0.]
18
19 RO = 8.314
20 T = 500.
21
22 # initialize k's
23 deltaGtot = -100.0
24 deltaG1 = -50.0
25 deltaG2 = deltaGtot - deltaG1
26 deltaG1d = 100.
27 deltaG2d = 125.
28
29 deltaGn1d = deltaG1d-deltaG1
30 deltaGn2d = deltaG2d-deltaG2
31
32 p1 = BPoly.from_derivatives([0, 1, 2, 3, 4], [[0, 0], [deltaG1d, 0], [deltaG1, 0], [deltaG1+deltaG2d,0], [deltaGtot,0]])
33
34 x=np.linspace(0,4,100)
35 y=p1(x)
36
37 plt.figure(figsize=(12,5))
38 plt.subplot(1,3,1)
39 plt.plot(x,y)
40 plt.ylabel('Free energy')
41 plt.xlabel('Reaction coordinate')
42
43 k = (kB*T/h)*np.exp(-np.array([deltaG1d,deltaGn1d,deltaG2d,deltaGn2d])*1000./(RO*T))
44 print(deltaG1d,deltaGn1d,deltaG2d,deltaGn2d)
45 print(k)
46
47 # Range of time to solve over
48 t = np.arange(0,6,0.1)
49 # t_span = (0., 10.)
50
51 p = (k,) # turn parameters into a tuple
52 # Solve two ODEs with odeint
53 #C = solve_ivp(dCdt,t_span,C_0,p,method='LSODA')
54 C = odeint(dCdt,C_0,t,p)
55
56 # would be nice to back compute rates
57 #print(C)
58 #R = dCdt(C,t,k)
59 #rate=np.array(R)
60
61 #print(R)
62 #rate_B = rate.transpose()[1]
63
64 #plt.figure()
65 #plt.plot(t,rate_B)

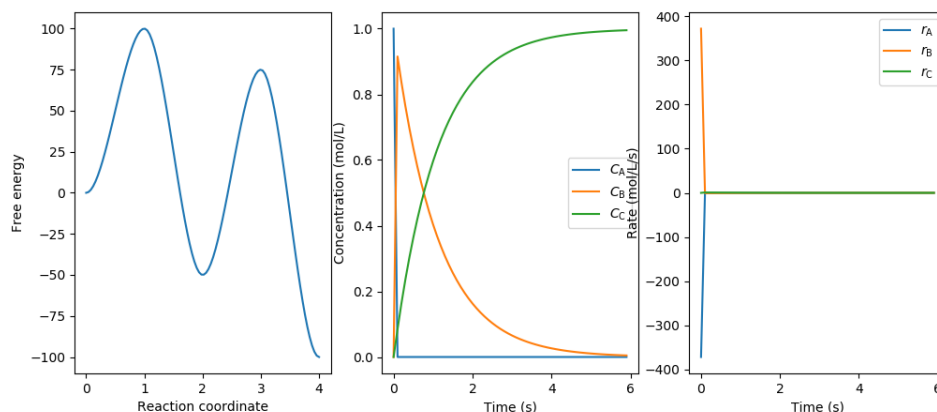
```

```

66 #
67
68 C_A = C.transpose()[0] # Get C_A from C
69 C_B = C.transpose()[1] # Get C_B from C
70 C_C = C.transpose()[2]
71
72 plt.subplot(1,3,2)
73 plt.plot(t,C_A,'-',label=r'$C_{\rm A}$')
74 plt.plot(t,C_B,'-',label=r'$C_{\rm B}$')
75 plt.plot(t,C_C,'-',label=r'$C_{\rm C}$')
76 plt.xlabel('Time (s)')
77 plt.ylabel('Concentration (mol/L)')
78 plt.legend()
79 # plt.savefig('./Images/ABC-conc.png')
80
81 plt.subplot(1,3,3)
82 plt.plot(t,-k[0]*C_A+k[1]*C_B,label=r'$r_{\rm A}$')
83 plt.plot(t,k[0]*C_A-k[1]*C_B-k[2]*C_B + k[3]*C_C,label=r'$r_{\rm B}$')
84 plt.plot(t,k[2]*C_B-k[3]*C_C,label=r'$r_{\rm C}$')
85 plt.xlabel('Time (s)')
86 plt.ylabel('Rate (mol/L/s)')
87 plt.legend()
88 plt.savefig('./Images/ABC-rxn.png')

```

100.0 150.0 125.0 175.0
 [3.71956184e+02 2.22248647e-03 9.09212619e-01 5.43266340e-06]



1. Pre-equilibrium
2. Selectivity
3. Rate control

7.0.1 Application: gas-phase reactions

1. Ethane pyrolysis, $C_2H_6 \longrightarrow C_2H_4 + H_2$, [doi:10.1021/jp206503d](https://doi.org/10.1021/jp206503d)

8 Heterogeneous reactions

1. adsorption, L-H
2. TPD

3. catalysis
4. Sabatier analysis

8.0.1 Heterogeneous reactions and catalysis

1. molecule-surface collisions
2. surface reactions
3. Ammonia oxidation, $\text{NH}_3 + \text{O}_2 \longrightarrow \text{NO} + \text{N}_2$, [doi:10.1021/acscatal.8b04251](https://doi.org/10.1021/acscatal.8b04251)

9 Liquid-phase reactions

9.0.1 Diffusion-controlled reactions

1. Intermediate complex
2. Steady-state approximation
3. Diffusion-controlled limit ($k_D = 4\pi(r_A + r_B)D_{AB}$)
4. Reaction-controlled limit ($k_{app} = (k_D/k_{-D})k_r$)

Table 2: Equilibrium and Rate Constants**Equilibrium Constants** $a A + b B \rightleftharpoons c C + d D$

$$K_{eq}(T) = e^{\Delta S^\circ(T)/k_B} e^{-\Delta H^\circ(T)/k_B T}$$

$$K_c(T) = \left(\frac{1}{c^\circ} \right)^{\nu_c + \nu_d - \nu_a - \nu_b} \frac{(q_c/V)^{\nu_c} (q_d/V)^{\nu_d}}{(q_a/V)^{\nu_a} (q_b/V)^{\nu_b}} e^{-\Delta E(0)\beta}$$

$$K_p(T) = \left(\frac{k_B T}{P^\circ} \right)^{\nu_c + \nu_d - \nu_a - \nu_b} \frac{(q_c/V)^{\nu_c} (q_d/V)^{\nu_d}}{(q_a/V)^{\nu_a} (q_b/V)^{\nu_b}} e^{-\Delta E(0)\beta}$$

Unimolecular Reaction $[A] \rightleftharpoons [A]^\ddagger \rightarrow C$

$$k(T) = \nu^\ddagger \bar{K}^\ddagger = \frac{k_B T}{h} \frac{\bar{q}_\ddagger(T)/V}{q_A(T)/V} e^{-\Delta E^\ddagger(0)\beta}$$

$$E_a = \Delta H^{\circ\ddagger} + k_B T \quad A = e^1 \frac{k_B T}{h} e^{\Delta S^{\circ\ddagger}}$$

Bimolecular Reaction $A + B \rightleftharpoons [AB]^\ddagger \rightarrow C$

$$k(T) = \nu^\ddagger \bar{K}^\ddagger = \frac{k_B T}{h} \frac{q_\ddagger(T)/V}{(q_A(T)/V)(q_B(T)/V)} \left(\frac{1}{c^\circ} \right)^{-1} e^{-\Delta E^\ddagger(0)\beta}$$

$$E_a = \Delta H^{\circ\ddagger} + 2k_B T \quad A = e^2 \frac{k_B T}{h} e^{\Delta S^{\circ\ddagger}}$$