Practical Kinetics

Practice 1:

Kinetic Resolution

Objectives:

1. Timecourse simulation of first-order kinetic resolution

2. Predict yield as a function of selectivity factor

Practice Assignment 1

This exercise reviews the material in Exercise 0.

You should be familar with:

- import statements
- list comprehensions
- mathematical operations (in particular, `math.exp`)
- plotting

Consider a first-order kinetic resolution:

$$SM_R \xrightarrow{k_R} PDT_R$$

$$SM_S \xrightarrow{k_S} PDT_S$$

Suppose that k_R = 1.0 and k_S = 0.1. For a **racemate**, compute the timecourse of this reaction from 0 to 100 seconds. Calculate the starting material and product *ee* as a function of conversion. Assume the total starting concentration is 1.0.

Practice Assignment 1

$$SM_R \xrightarrow{k_R} PDT_R$$

$$SM_S \xrightarrow{k_S} PDT_S$$

$$[A] = [A]_0 \exp\left(-k_1 t\right)$$

In the next slides, I will give some fill-in-the-blank code. Try not to look at the answers until you've given it a good try.

Strategy:

Make four lists, `SM_R`, `PDT_R`, `SM_S`, and `PDT_S` to hold the concentrations as a function of another list called `time`. Calculate the concentrations using the first-order integrated rate law.

Then generate three more lists, `conversion`, `ee_SM`, and `ee_PDT`. For each point in time, look at the starting material and product concentrations to calculate these quantities (using list comprehensions).

Instead of plotting concentration vs. time, we'll plot starting material *ee* vs. conversion.

Code Outline 1

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from math import exp
%matplotlib inline

# rate constants
k_R    = ___
k_S    = 0.1

# starting concentrations
SM_R_0 = ___
SM_S_0 = ___
total_concentration = ___ + ___

# this fills the interval [1E-3, 1E2] with 50 points logarithmically
# this helps even out the points on the graphs we will make
time = np.logspace(-3,2,50)

# use the range command to get the indices of time: [0, 1, ..., len(time-1)]
indices = ___

# use exp(-k*t) in a list comprehension
SM_R  = [ ___ ]
PDT_R = [ ___ ]
SM_S  = [ ___ ]
PDT_S = [ ___ ]
```

Code Outline 1

(continued)

```
conversion = [ ___ for i in indices ]
ee_SM      = [ ___ for i in indices ]
ee_PDT     = [ ___ for i in indices ]

# plot
plt.plot(___, ___, "r.", label="ee (SM)")
plt.plot(___, ___, "b.", label="ee (PDT)")
plt.xlabel("conversion")
plt.ylabel("enantiomeric excess")

# this puts the legend in the best place
plt.legend(loc="best")
```

Solutions to Assignment 1

Here is my answer:

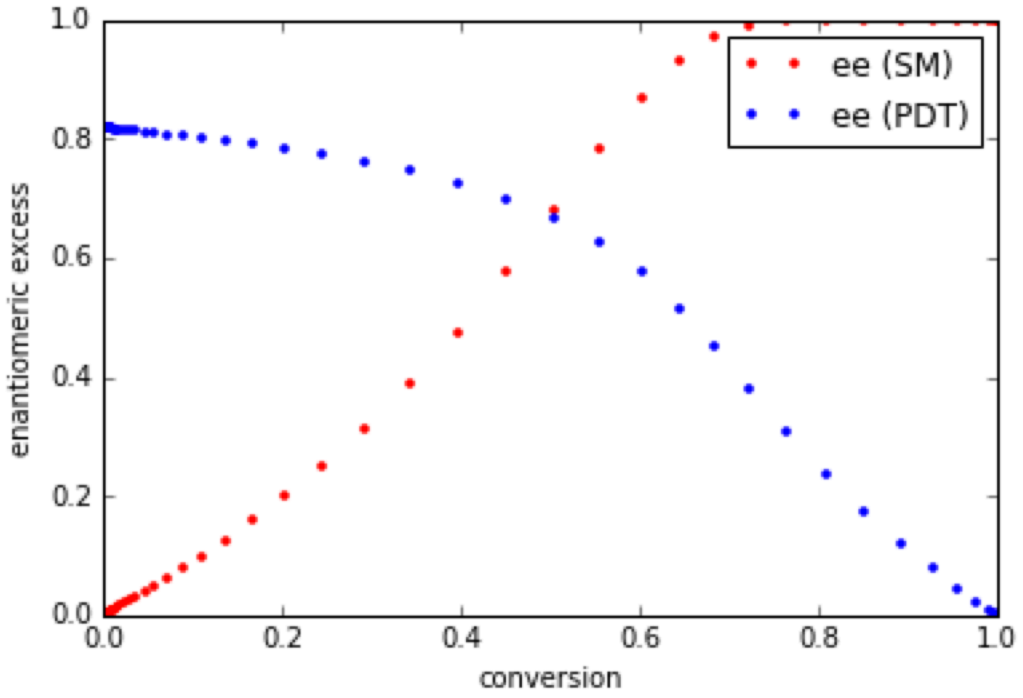
```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from math import exp

# rate constants
k_R    = 1.0
k_S    = 0.1

# starting concentrations
SM_R_0 = 0.5
SM_S_0 = 0.5
total_concentration = SM_R_0 + SM_S_0

# simulate
time = np.logspace(-3,2,50)
indices = range(len(time))

SM_R  = [ SM_R_0 * exp(-k_R*t) for t in time ]
PDT_R = [ SM_R_0 - i for i in SM_R ]
SM_S  = [ SM_S_0 * exp(-k_S*t) for t in time ]
PDT_S = [ SM_S_0 - i for i in SM_S ]
```

Solutions to Assignment 1

(continued)

```
conversion = [ (PDT_R[i] + PDT_S[i]) / total_concentration for i in indices ]
ee_SM      = [ (SM_S[i] - SM_R[i]) / (SM_R[i] + SM_S[i]) for i in indices ]
ee_PDT     = [ (PDT_R[i] - PDT_S[i]) / (PDT_R[i] + PDT_S[i]) for i in indices ]

# plot
plt.plot(conversion, ee_SM, "r.", label="ee (SM)")
plt.plot(conversion, ee_PDT, "b.", label="ee (PDT)")
plt.xlabel("conversion")
plt.ylabel("enantiomeric excess")
plt.legend(loc="best")
```

For this relatively low *s* factor of 10.0, the reaction must be run to at least 70% conversion to get high ee in the recovered starting material.



Practice Assignment 2

Plot the yield of recovered starting material, given a target ee and selectivity factor.

Strategy:

Run a series of timecourse simulations, with different selectivity factors. To do this, hold k_S fixed and adjust k_S to give the desired selectivity factor.

For each selectivity factor, find the conversion at which the starting material ee is at the desired level.

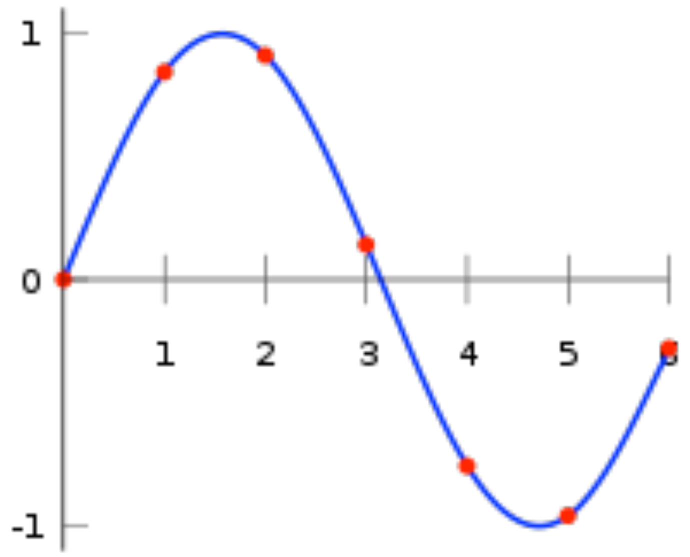
Because the timecourse simulation has a finite number of points, the starting material may never reach the desired ee exactly. However, we will have a series of points (starting material *ee*, conversion).

If we interpolate these points, we can generate a function to give us the information we need: (target starting material *ee*, conversion at the target *ee*).

Practice Assignment 2

Plot the yield of recovered starting material, given a target ee and selectivity factor.

You will need the `interp1d` function from `scipy.interpolate`. It takes a series of points (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) and creates a smooth function between them:



One can then evaluate the function at an arbitrary point x, as long as x remains in the interpolation domain. The code outline on the next page shows you how to do it in this context. (**Technical Note:** we'll use a cubic spline, the most popular algorithm.)

Code Outline 2

Fill in the blanks below. (This assumes the previous code is present.)

```
from scipy.interpolate import interp1d

# gives the yield of the recovered starting material given a target
# ee and selectivity factor
def SM_yield(target_ee, selectivity):

    # assume k_S is constant
    k_R = k_S * ___

    # recalculate with new rate constants (look at previous code)
    SM_R  = [ ___ ]
    PDT_R = [ ___ ]
    SM_S  = [ ___ ]
    PDT_S = [ ___ ]

    conversion = [ ___ ]
    ee_SM      = [ ___ ]

    # if the ee never gets up to the target ee in this simulation,
    # return a starting material yield of zero
    #
    # this occurs for low selectivity factors
    if max(ee_SM) < target_ee:
        return 0.0
```

Code Outline 2

(the function, continued)

```
def SM_yield(target_ee, selectivity):
    ...

    # interpolate the conversion vs. ee function
    interpolation_function = interp1d(ee_SM, conversion)
    interpolated_conversion = interpolation_function(target_ee)

    # calculate the starting material yield
    this_yield = ___ - ___
    return this_yield
```

Now, let’s run the simulation and plot:

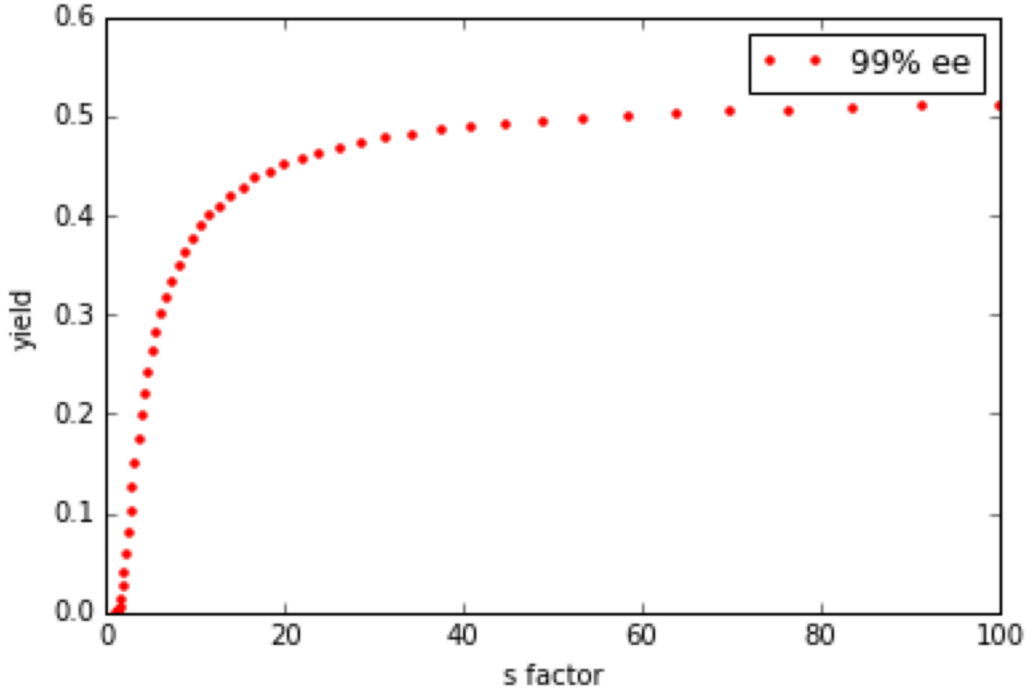
```
target_ee = 0.99
selectivities = np.logspace(0.1, 2, 50)
yields = [ SM_yield(___,___) for ___ ]

plt.plot(___, ___, "r.", label="99% ee")
plt.legend(loc="best")
plt.xlabel("s factor")
plt.ylabel("yield")
```

Solutions to Asssignment 2

```
from scipy.interpolate import interp1d

def SM_yield(target_ee, selectivity):
    k_R = k_S * selectivity

    SM_R   = [ SM_R_0 * exp(-k_R*t) for t in time ]
    PDT_R = [ SM_R_0 - i for i in SM_R ]
    SM_S   = [ SM_S_0 * exp(-k_S*t) for t in time ]
    PDT_S = [ SM_S_0 - i for i in SM_S ]

    conversion = [ (PDT_R[i] + PDT_S[i]) / total_concentration for i in indices ]
    ee_SM      = [ (SM_S[i] - SM_R[i]) / (SM_R[i] + SM_S[i]) for i in indices ]

    if max(ee_SM) < target_ee:
        return 0.0

    interpolation_function = interp1d(ee_SM, conversion)
    interpolated_conversion = interpolation_function(target_ee)
    this_yield = 1.0 - interpolated_conversion
    return this_yield

target_ee = 0.99
selectivities = np.logspace(0.1, 2, 50)
yields = [ SM_yield(target_ee, i) for i in selectivities ]

plt.plot(selectivities, yields_90, "r.", label="99% ee")
plt.legend(loc="best")
plt.xlabel("s factor")
plt.ylabel("yield")
```

Solutions to Asssignment 2

Once the selectivity factor passes 30 or so, the yield in 99% *ee* starting material is essentially quantitative.



Try plotting some other selectivity factors!

Summary

In these practice problems, we:

- imported libraries

- calculated concentrations with list comprehensions

- computed quantities like conversion and *ee* from (time, concentration) data

- made some plots

We also learned how to:

- use `np.logspace` to populate lists logarithmically
  (as opposed to linearly, as with `np.arange` or `np.linspace`)

- generate interpolations with `scipy.interpolate.interp1d`

We will use these commands in Exercise 2 to process some kinetic data.