thermal_denaturation_fitting

September 29, 2024

Fitting thermal denaturation data

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import gridspec
%matplotlib inline
from scipy.optimize import curve_fit
import os

basedir = '/home/david/gh/intro_curve_fitting_python'

try:
    os.chdir(basedir)
except:
    print('\n\nproblem changing to the directory you specified; does it exist?
 ↪\nthe kernel will now restart; rerun this program.\n\n')
    quit()
```

The Brautigam laboratory's equation for fitting protein thermal denaturation data is:

$$\frac{m_1 * T + b_1 + (m_2 * T + b_2) * e^{-\frac{\Delta H}{RT}(1-\frac{T}{TM})}}{1 + e^{-\frac{\Delta H}{RT}(1-\frac{T}{TM})}}$$

in which: | Symbol | Meaning | | ——:| ————:| | T | absolute temperature, the independent variable | θ | arbitrary temperature-dependent signal reporting the denaturation K_{eq} | | m_1 | slope of line in low-temperature region | | b_1 | intercept of line in low-temperature region | | m_2 | slope of line in high-temperature region | | b_2 | intercept of line in high-temperature region | | $ H $ | denaturation enthalpy, units as per choice of R | | R | the gas constant, in whichever units aree preferred | | T_M | melting temperature, Kelvin |

This equation is an adaptation of the integrated form of the van't Hoff equation, and my own derivation of it may be found here. As I derive both a four- and a six-parameter version of the equation, I have also included an exercise in F-statistics to evaluate if the more complex equation actually provides a better fit.

Encode the fitting function:

[12]:
```
# if the value of a variable is specified in the definition of a function
# that is its default value, and the function may be called without specifying␣
 ↪that parameter
```

```
# or its value. if, when called, the value of the variable is specified,
# that value overrides the default

# note then that the default units for deltaH are J/mol
# define the function
def tdmodel(T, m1, b1, m2, b2, deltaH, TM, R=8.314):
    line1=m1*T+b1
    line2=m2*T+b2
    Q=np.exp(-deltaH/(R*T)*(1-T/TM))
    return (line1+line2*Q)/(1+Q)
```

Good initial guesses for fittable parameters Evidently this equation, which has 6 fittable parameters, is quite a bit more complicated than those necessary to fit linear and exponential decay data. With this number of parameters, the trick to gettting the right fit, as opposed to a fit that is stuck in some pathological local minimum, is to make good initial guesses for the unknown parameters. In what follows I walk through one example of how I accomplished that for a PupB NTSD mutant using actual Colbert Lab data. In the '…/intro_curve_fitting_python/thermal_denaturation_data' directory, find the following files:

[13]:
```
datadir = basedir+'/thermal_denaturation_data/'
os.listdir(datadir)
```

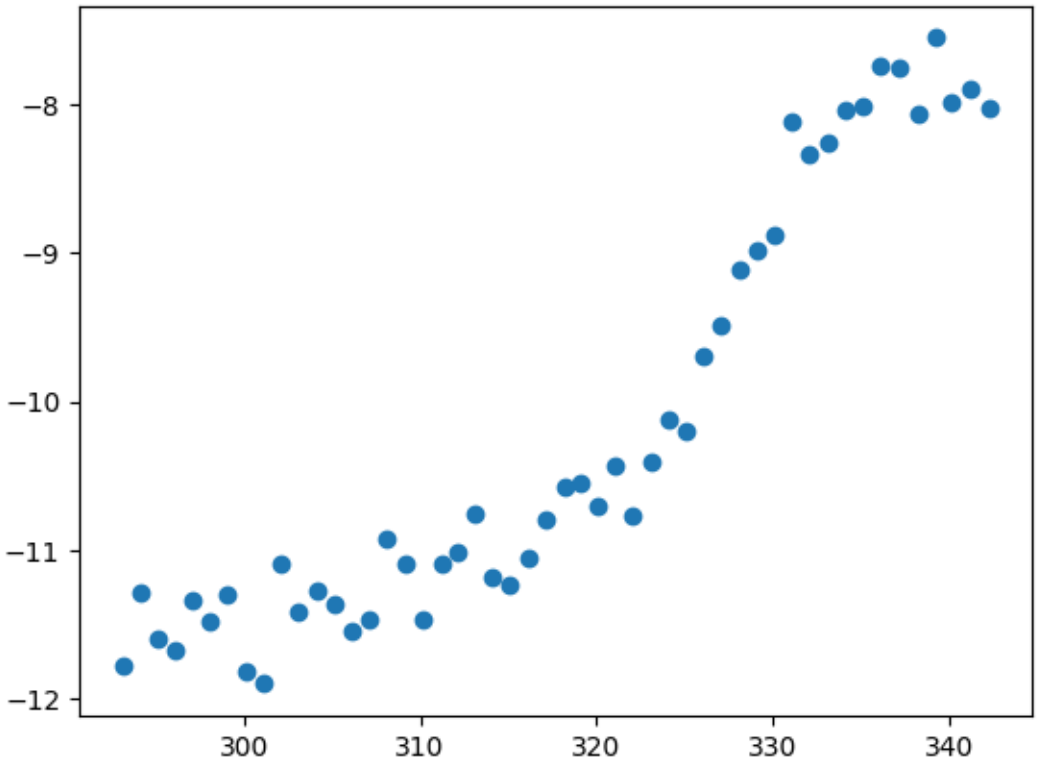
[13]:
```
['PupB NTSD L74A 25 uM 217 nm F.csv',
 'PupB NTSD Q69K 25 uM 217nm F.csv',
 'PupB NTSD H72D 25 uM 217nm F.csv']
```

An alternative means of reading csv files: See here for more.

[14]:
```
import csv

fn = datadir+'PupB NTSD L74A 25 uM 217 nm F.csv'
csvfile = open(fn)
csvreader = csv.reader(csvfile, delimiter=',')

# create empty x and y arrays
x = []
y = []

for row in csvreader:
    x.append(float(row[0]))
    y.append(float(row[1]))

csvfile.close()

# make x and y proper np arrays
x = np.array(x)
y = np.array(y)
```

Scatter Plot Always look at your raw data:

[15]:
```
# scatter plot:
plt.scatter(x,y)
plt.show()
```



Get the low- and high-temperature lines:As you've seen now, linear fits are insanely easy, so it is straightforward to get good initial estimates for m_1, b_1, m_2, and b_2. Let's do that:

[16]:
```
# get the first 20 points for a low temp line

# the following 'slices' the array x beginnning with the (implied)
# 0 prior to the colon, and ending with the 20 indicating the
# first array element _not_ to include in the 'slice'
eks = x[:20]
why = y[:20]

def func(x, m, b):
    return m*x+b

ltlineparam, pcov = curve_fit(func, eks, why)
m1 = ltlineparam[0]
b1 = ltlineparam[1]
```

```
# get the last 12 points for a high temp line

# more slicing syntax
# if the argument prior to the colon is negative, it means
# 'start counting from the end of the array'
# therefore, 'cut from the 12th from the last element of the
# array through to the end'
eks = x[-12:]
why = y[-12:]

htlineparam, pcov = curve_fit(func, eks, why)
m2 = htlineparam[0]
b2 = htlineparam[1]
```

Estimates for ΔH and T_M

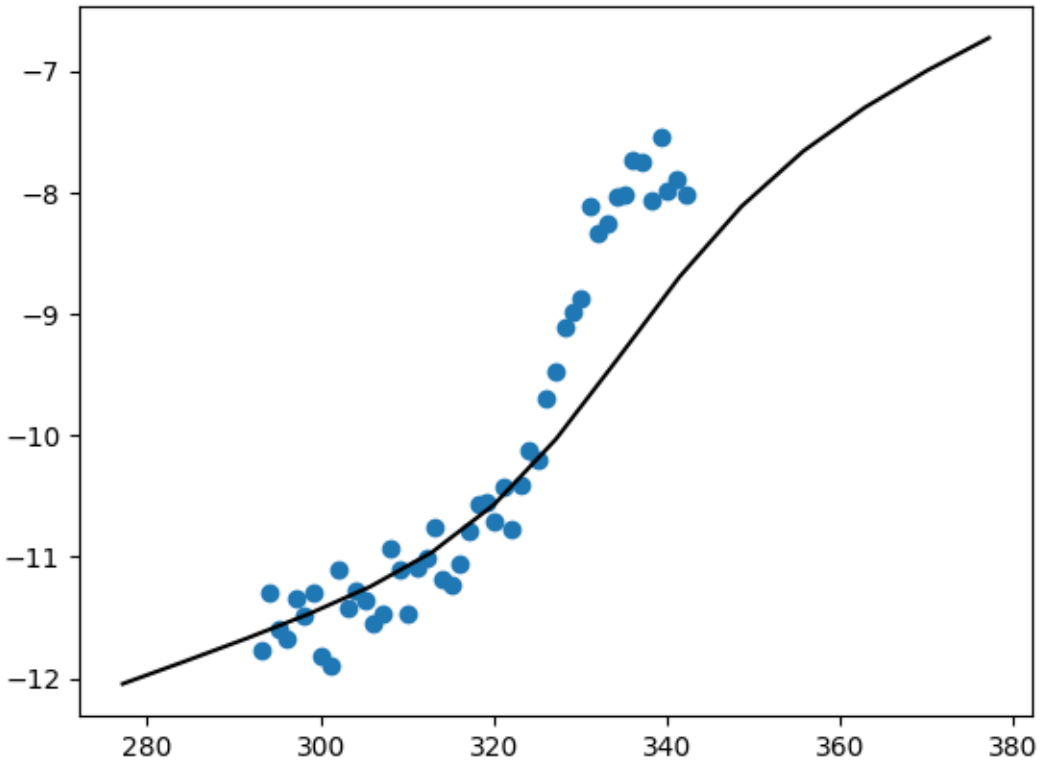
For ΔH, because an order of magnitude guess is sufficient, let's go with 100 kJ / mol. T_M may be eye-balled from the curve, I make it 335 K.

[17]:
```
deltaH = 100000
TM = 335
print('initial guesses for fitting:')
print('m1: %0.3f, b1: %0.3f, m2: %0.3f, b2: %0.3f, deltaH: %0.0f, TM: %0.0f' %␣
 ↪(m1, b1, m2, b2, deltaH, TM))
```

```
initial guesses for fitting:
m1: 0.025, b1: -19.107, m2: 0.032, b2: -18.782, deltaH: 100000, TM: 335
```

Compare Initial Paramters Against Data:

[21]:
```
mintemp = 4 # degrees C
maxtemp = 104
mintemp += 273.15 # conversion to Kelvin
maxtemp += 273.15
eks = np.linspace(mintemp, maxtemp, 15)

sigma = tdmodel(eks, m1, b1, m2, b2, deltaH, TM)
plt.plot(T,sigma, 'k-')
plt.scatter(x,y)
plt.show()
```



Yes, that is close enough.Do the fitting using the values just estimated as initial guesses:

[22]:
```
p = [m1, b1, m2, b2, deltaH, TM]

# do it
# argument 'p0=p' supplies initial guesses for fittable parameters
popt, pcov = curve_fit(tdmodel, x, y, p0=p)
popt
```

[22]:
```
array([ 2.85739746e-02, -2.00378479e+01, -6.32331433e-02,  1.37202477e+01,
        2.85627011e+05,  3.29470759e+02])
```

Compute residuals and R^2:

[24]:
```
residuals = y - tdmodel(x, *popt)
ssqresid = np.sum(np.square(residuals))
totssq = np.sum(np.square(y-np.mean(y)))
rsq = 1-ssqresid/totssq
print('Rsquared: %0.3f' % rsq)
```

```
Rsquared: 0.981
```

Prepare a figure:

[38]:
```
fn = basedir+'/thermal_denaturation_data/dmm_L74A_F_fit_figure.png'
eks = np.linspace(np.min(x), np.max(x), 100)
why = tdmodel(eks, *popt)

# create a figure
fig = plt.figure()
fig.set_figwidth=(6)
fig.set_figheight(6)

# create grid for different subplots
spec = gridspec.GridSpec(ncols=1, nrows=2,
                                                hspace=0.1, height_ratios=[4,␣
 ↪1])

ax0 = fig.add_subplot(spec[0])
ax0.scatter(x,y, marker='.', label='\ntheta, 217 mdeg\n')
ax0.plot(eks,why,'k-', label='$m_1$: %0.3f\n $b_1$: %0.1f\n $m_2$: %0.3f\n␣
 ↪$b_2$: %0.1f\n deltaH: %0.0f J/mol\n $T_M$: %0.1f K' % tuple(popt))
ax0.legend()
ax0.set_ylabel('ellipticity at 217nm, mdeg')
plt.tick_params('x', labelbottom=False)

ax1 = fig.add_subplot(spec[1])
ax1.scatter(x, residuals, marker='.')
ax1.set_ylabel('residuals')
ax1.set_xlabel('Temperature, Kelvin')

plt.suptitle('Brautigam Equation Curve Fit to PupB NTSD L74A Forward Melting␣
 ↪Data')
plt.savefig(fn)
plt.show()
```

Brautigam Equation Curve Fit to PupB NTSD L74A Forward Melting Data