assaytools Documentation

Release 0.1.0

John Chodera and Sonya Hanson

Dec 04, 2018

Contents

Assay modeling and Bayesian analysis made easy.

AssayTools is a python library that allows users to model automated assays and analyze assay data using powerful Bayesian techniques that allow complete characterization of the uncertainty in fit models. With AssayTools, you can

- Create models of experimental assays (e.g. fluorescence or absorbance assays of ligand binding from titration curves prepared by automated liquid handlers)
- Analyze data from these assays using powerful Bayesian techniques
- Derive parameters for these experimental assays from real data to use in modeling
- Model new assay configurations to determine expected accuracy and bias across a range of ligand affinities

The library also ships with a command-line application for visualizing Tecan Infinite plate reader XML output. When you install AssayTools, the script will be installed under the name `xml2png`.

- Download the code
- See it in action
- Get involved

CHAPTER 1

Documentation

1.1 Installation

1.1.1 Install with conda

To install *AssayTools* and its dependencies with *conda <conda.pydata.org>‘_*, use the following commands:

```
$ conda install -c omnia assaytools
```

1.1.2 Install from source

You can install the latest development version of `AssayTools` from github via `pip`:

```
$ pip install git+https://github.com/choderalab/assaytools.git
```

Testing Your Installation

Running the tests is a great way to verify that everything is working. The test suite uses nose, which you can install via conda:

```
$ conda install nose
```

Then, to run the tests:

```
$ nosetests -vv assaytools
```

1.2 Theory

This section describes the theory behind the various Bayesian model fitting schemes in *AssayTools*.

1.2.1 Bayesian analysis

AssayTools uses Bayesian inference to infer unknown parameters (such as ligand binding free energies) from experimental spectroscopic data. It does this to allow the complete uncertainty—in the form of the joint distribution of all unknown parameters—to be rigorously characterized. The most common way to summarize these results is generally to extract confidence intervals of individual parameters, but much more sophisticated analyses—such as examining the correlation between parameters—are also possible.

The Bayesian analysis scheme is intended to be modular, and the user can select whether certain effects (such as *inner filter effects*) are incorporated into the model. Below, we describe the components of the model. If an effect that carries unknown nuisance parameters (such as extinction coefficients for *inner filter effects*), these nuisance parameters carry additional prior terms along with them and are inferred as part of the inference procedure.

Unknown parameters

For convenience, we define the unknown parameters in the model for reference:

- the **true total ligand concentration** L_{true} in the well (including all species involving the ligand)
- the **true receptor concentration** R_{true} (including all species involving the receptor)

Data

For each experiment, the data is given by a set of observations for each well. Each well is associated with some properties:

- the **volume** V of sample in the well (mostly buffer)
- a total concentration $[R]_T$ of **receptor** added to the well
- a total concentration $[L]_T$ of **ligand** added to the well (or potentially multiple ligands)
- the **well area** A with the assumption that the well is cylindrical (allowing the path length l to be computed)

and one or more experimental measurements:

- a **top fluorescence measurement** (returning toward the incident excitation light) F_{top}
- a **bottom fluorescence measurement** (proceeding in the same direction as the incident excitation light) F_{bottom}
- an **absorbance measurement** A

Priors

Each unknown parameter in the model is assigned a *prior* distribution that reflects the state of knowledge we have of its value before including the effects of the observed data.

Concentrations

While we design the experiment to dispense the *intended* amount of protein and ligand into each well, the true amount dispensed into the well will vary due to random pipetting error. The *true* concentrations of protein R_{true} and ligand L_{true} in each well are therefore unknown. Because we propagate the pipetting error along with the intended concentrations, we have the intended ("stated") protein concentration P_{stated} and its standard error $\delta P_{\mathrm{stated}}$ as input. Similarly, the stated ligand concentration L_{stated} and its error $\delta L_{\mathrm{stated}}$ are also known.

If we assume the dispensing process is free of bias, the simplest distribution that fits the stated concentration and its standard deviation without making additional assumptions is a Gaussian.

We assign these true concentrations for the receptor R_{true} and ligand L_{true} a prior distribution. If `concentration_priors` is set to `gaussian`, this is precisely what is used

$$R_{true} \sim N(R_{stated}, \delta R_{stated})$$
$$L_{true} \sim N(L_{stated}, \delta L_{stated})$$

This is expressed in the `pymc` model as

```
Ptrue = pymc.Normal('Ptrue', mu=Pstated, tau=dPstated**(-2)) # M
Ltrue = pymc.Normal('Ltrue', mu=Lstated, tau=dLstated**(-2)) # M
```

Note: `pymc` uses the **precision** $\tau \equiv \sigma^{-2}$ instead of the variance σ^2 as a parameter of the normal distribution.

Gaussian priors have the unfortunate drawback that there is a small but nonzero probability that these concentrations would be negative, leading to nonsensical (unphysical) concentrations. To avoid this, we generally use a lognormal distribution (selected by `concentration_priors='lognormal'`).

Note: The parameters of a *lognormal distribution* differ from those of a normal distribution by the relationship described here. The parameters above ensure that the mean concentration is the stated concentration and the standard deviation is its experimental uncertainty. The relationship between the mean and variance of the normal distribution μ_N, σ_N^2 and the parameters for the lognormal distribution is given by:

$$\mu_{LN} = \ln \frac{\mu_N^2}{\sqrt{\mu_N^2 + \sigma_N^2}} \;;\; \sigma_{LN}^2 = \ln \left[1 + \left(\frac{\sigma_N}{\mu_N} \right)^2 \right] \;;\; \tau_{LN} = \ln \left[1 + \left(\frac{\sigma_N}{\mu_N} \right)^2 \right]^{-1}$$

Binding free energy

The ligand binding free energy ΔG is unknown, and presumed to either be unknown over a large uniform range with the `uniform` prior

$$\Delta G \sim U(-\Delta G_{\mathrm{min}}, +\Delta G_{\mathrm{max}})$$

where we by default take $\Delta G_{\mathrm{min}} = \ln 10^{-15}$ (femtomolar affinity) and *Delta G_mathrm{max} = 0* (molar affinity), where ΔG is in units of thermal energy $k_B T$.

This is expressed in the `pymc` model as

```
# binding free energy (kT), uniform over huge range
DeltaG = pymc.Uniform('DeltaG', lower=DG_min, upper=DG_max)
```

This uniform prior has the drawback that affinities near the extreme measurable ranges are simply unknown with equal likelihood out to absurd extreme values.

We can attenuate the posterior probabilities at extreme affinities by using a prior inspired by the range of data recorded in ChEMBL via the `chembl` prior, with a Gaussian form

$$\Delta G \sim N(0, \sigma^2)$$
$$\sigma = 12.5\,\mathrm{kcal/mol}$$

This is expressed in the `pymc` model as

```
# binding free energy (kT), using a Gaussian prior inspired by ChEMBL
DeltaG = pymc.Normal('DeltaG', mu=0, tau=1./(12.5**2))
```

Modular components of the Bayesian model

We now discuss the various modular components of the Bayesian inference scheme.

These components generally involve models of observed spectroscopic value that are computed from concentrations of the various components $[X_i]$ which represent, for example, free receptor R, complexed receptor RL, or free ligand L. These concentrations are computed from the current samples of true total concentrations and binding affinities using one of the specified *binding models* described below.

Fluorescence

Fluorescence model.

Fluorescence can be measured from either the top, bottom, or both. The true fluorescence depends on the concentration of each species X_i:

$$F_{\mathrm{top}} = I_{ex}\left[\sum_i q_i(ex,em)[X_i] + lF_{\mathrm{buffer}} + F_{\mathrm{plate}}\right]$$
$$F_{\mathrm{bottom}} = G_{\mathrm{bottom}} \cdot I_{ex}\left[\sum_i q_i(ex,em)[X_i] + lF_{\mathrm{buffer}} + F_{\mathrm{plate}}\right]$$

Here, I_{ex} is the incident excitation intensity, $q_i(ex,em)$ are the quantum efficiencies of each species at the excitation/emission wavelengths, F_{buffer} is a buffer fluorescence per unit path length, and F_{plate} is the background fluorescence of the plate. Notably, without *inner filter effects*, the only factor that causes differences between top and bottom fluorescence is the gain factor *G_mathrm{bottom}* that captures a potential difference in detector gains between the top and bottom detectors.

Observed fluorescence.

The observed fluorescence $F_{\mathrm{top}}^{\mathrm{obs}}$ and $F_{\mathrm{bottom}}^{\mathrm{obs}}$ will differ from the true fluorescence due to detector noise. Because the observed fluorescence is reported as the mean of a number of detector measurements from independent flashes of the Xenon lamp, detector error will be well described by a normal distribution:

$$F_{\mathrm{top}}^{\mathrm{obs}} \sim N(f_{\mathrm{top}}, \sigma_{\mathrm{top}}^2)$$
$$F_{\mathrm{bottom}}^{\mathrm{obs}} \sim N(f_{\mathrm{top}}, \sigma_{\mathrm{bottom}}^2)$$

The measurement errors σ_{top} and σ_{bottom} are assigned Jeffreys priors, which are uniform in $\ln \sigma$

$$\ln \sigma \sim U(-10, \ln F_{max})$$

By default, the same detector error σ is used for both top and bottom detectors, but separate errors can be used if `link_top_and_bottom_sigma = False`.

While the detector error is inferred separately for each experiment since the detector gain may differ from experiment. If multiple datasets using the same instrument configuration and detector gain are inferred together—such as the inclusion of calibration experiments with controls—this will help improve the detector error estimate.

Quantum efficiencies.

Since the quantum efficiencies $q_i(ex, em)$ of each species X_i are unknown, they are inferred as nuisance parameters as part of the Bayesian inference process. We therefore assign a uniform (informationless) priors to these, though we use the product $F_i \equiv I_{ex} q_i(ex, em)$ for convenience since I_{ex} and the scaling factor to convert observed fluorescence into reported arbitrary fluorescence units is unknown:

$$F_i \sim U(0, F_{i,max})$$
$$F_{\mathrm{plate}} \sim U(0, F_{\mathrm{max}})$$
$$F_{\mathrm{buffer}} \sim U(0, F_{\mathrm{max}}/l)$$

For efficiency, we compute the maximum allowed values based on an upper limit of these quantities from the observed data.

We also make efficient initial guesses for these quantities, which assume that:

- F_{buffer} assumes the minimum fluorescence signal is explained by only buffer fluorescence
- F_{plate} assumes the minimum fluorescence signal is explained by only plate fluorescence
- F_L assumes the maximum fluorescence signal increase above background is explained by the free ligand fluorescence
- F_R assumes the receptor fluorescence is zero
- F_{PL} assumes that the maximum fluorescence signal increase above background is explained by complex fluorescence with 100% complex formation

These assumptions can of course be violated once the sampler starts to infer these quantities.

In the pymc model, these priors are implemented via

```
model['F_PL'] = pymc.Uniform('F_PL', lower=0.0, upper=2*Fmax/min(Pstated.max(),Lstated.max()), value=F_
↪PL_guess)
model['F_P'] = pymc.Uniform('F_P', lower=0.0, upper=2*(Fmax/Pstated).max(), value=F_P_guess)
model['F_L'] = pymc.Uniform('F_L', lower=0.0, upper=2*(Fmax/Lstated).max(), value=F_L_guess)
model['F_plate'] = pymc.Uniform('F_plate', lower=0.0, upper=Fmax, value=F_plate_guess)
model['F_buffer'] = pymc.Uniform('F_buffer', lower=0.0, upper=Fmax/path_length, value=F_buffer_guess)
```

If an estimate of F_{PL} is known from a prior experiment, this value and its standard error can be specified via a lognormal distribution

```
model['F_PL'] = pymc.Lognormal('F_PL', mu=np.log(F_PL**2 / np.sqrt(dF_PL**2 + F_PL**2)), tau=np.sqrt(np.
↪log(1.0 + (dF_PL/F_PL)**2))**(-2))
```

Top/bottom detector gain.

The bottom detector relative gain factor is assigned a uniform prior over the log gain:

$$\ln G_{\mathrm{bottom}} \sim U(-6, +6)$$

which is implemented in the pymc model as

```
model['log_gain_bottom'] = pymc.Uniform('log_gain_bottom', lower=-6.0, upper=6.0, value=log_gain_guess)
```

Absorbance

Absorbance model.

The absorbance is determined by the the extinction coefficient of each component X_i (*R*, *L*, *RL* for simple two-component binding) at the illumination wavelength, as well as any intrinsic absorbance of the plate at that wavelength.

$$A = 1 - e^{-\epsilon \cdot l \cdot [L]}$$

where ϵ is the extinction coefficient of the species (e.g. free ligand L) at the illumination wavelength (excitation or emission), l is the path length, and c is the concentration of the species.

Note: You may be more familiar with the linearized form of Beer's law ($A = \epsilon l c$). It's easy to see that this comes from a Taylor expansion of the above equation, truncated to first order: $1 - e^{-\epsilon l c} \approx 1 - \left[1 - \epsilon l c + \mathcal{O}(\epsilon l c)^2)\right] \approx \epsilon l c$. We use the equation above instead because it is much more accurate for larger absorbance values.

The plate absorbance is a nuisance parameter that is assigned a uniform informationless prior:

$$A_{\mathrm{plate}} \sim U(0, 1)$$

Currently, `AssayTools` supports absorbance measurements made at either (or both) the excitation and emission wavelengths. Absorbance measurements performed at the excitation wavelength help constrain the extinction coefficient for *primary inner filter effects*, while absorbance measurements at the emission wavelength help constrain the extinction coefficients for *secondary inner filter effects*. Note that even if plates that are not highly transparent in the excitation or emission wavelengths are used, this still provides useful information—this effect is corrected for by inferring the plate absorbance A_{plate} at the appropriate wavelengths.

Note: Currently, `AssayTools` only models absorbance for the ligand, using data from wells in which only ligand in buffer is present. In the future, we intend to extend this to support absorbance of all components.

Observed absorbance.

As the detector averages many measurements from multiple flashes of a Xenon lamp for the reported absorbance A^{obs}, the observed measurement can be modeled with a normal distribution

$$A^{\mathrm{obs}} \sim N(A, \sigma^2_{\mathrm{abs}})$$

The detector error σ_A is assigned Jeffreys priors, which are uniform in $\ln \sigma_{\mathrm{abs}}$

$$\ln \sigma_{\mathrm{abs}} \sim U(-10, 0)$$

Note: It is critical that if multiple datasets are inferred jointly, they all be from the same plate type.

Inner filter effects

Primary inner filter effect.

At high ligand concentrations, if the ligand has significant absorbance at the excitation wavelength, the amount of light reaching the bottom of the well is less than the light reaching the top of the well. This is called the *primary inner filter effect*, and has the net effect of attenuating the observed absorbance and fluorescence.

To see where this effect comes from, consider the permeation of light through a liquid with molar concentration c, and extinction coefficient ϵ.

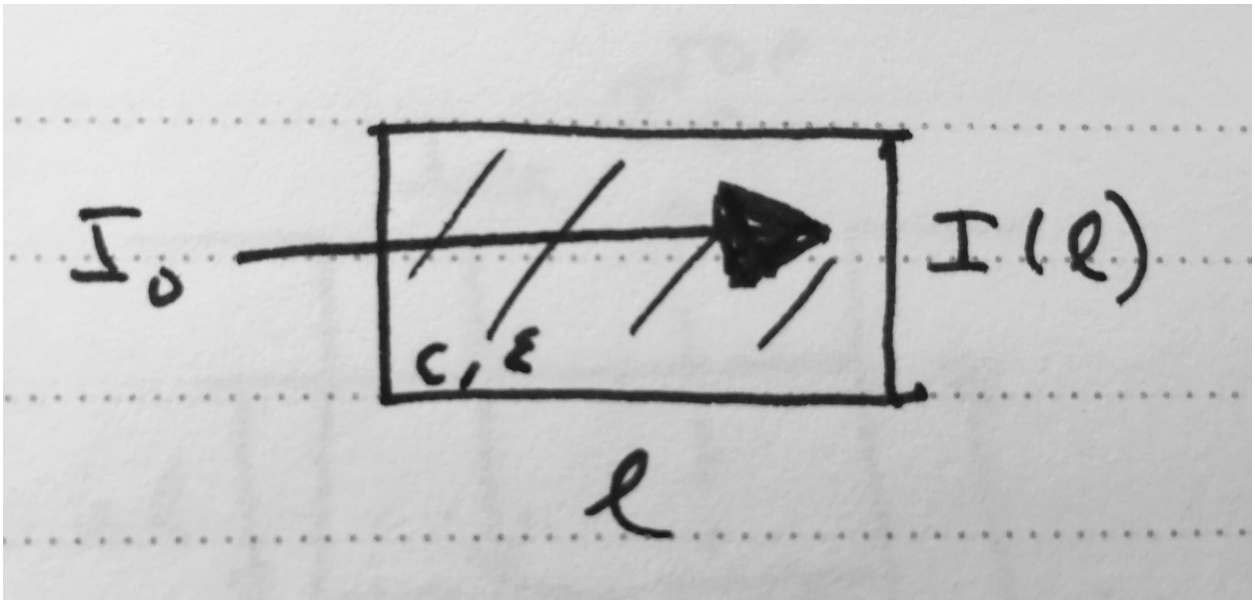

Fig. 1: Attenuation of light passing through a liquid containing absorbing species.

A slice of width Δl at depth l will absorb some of the incoming light intensity $I(l)$:

$$\Delta I = -\epsilon \cdot \Delta l \cdot c \cdot I(l)$$

If we shrink Δl down to an infinitesimal slice, this gives us a differential equation for the intensity $I(l)$ at depth l:

$$\frac{\partial I(l)}{\partial l} = -\epsilon \cdot c \cdot I(l)$$

It's easy to see that the solution to this differential equation is given by

$$I(l) = I_0 e^{-\epsilon l c}$$

since this satisfies the differential equation:

$$\frac{\partial I(l)}{\partial l} = I_0(-\epsilon c)e^{-\epsilon l c} = -\epsilon \cdot c \cdot I(l)$$

If only the primary inner filter effect is used, both top and bottom fluorescence are attenuated by a factor that can be computed by integrating the attenuation of incident light over the whole liquid path length:

$$\mathrm{IF}_{\mathrm{top/bottom}} = \int_0^1 dx e^{-\epsilon_{ex} \cdot x l \cdot c} = \left[\frac{e^{-\epsilon_{ex} \cdot x l \cdot c}}{-\epsilon_{ex} \cdot l \cdot c} \right]_0^1 = \frac{1 - e^{-\epsilon_{ex} \cdot l \cdot c}}{\epsilon_{ex} \cdot l \cdot c}$$

Note: When $\epsilon \cdot l \cdot c \ll 1$, numerical underflow of the exponential becomes a problem. To avoid negative attenuation factors, a fourth-order Taylor series approximation of the exponential is used in computing the attenuation factor if $\epsilon \cdot l \cdot < 0.01$.

Note: Currently, inner filter effects are only computed for the free ligand, but we plan to extend this to include a sum over the effects from all species.

Secondary inner filter effect.

Similarly, the *secondary inner filter effect* is caused by significant absorbance at the emission wavelength. When both effects are combined, the net attenuation effect depends on the geometry of excitation and detection:

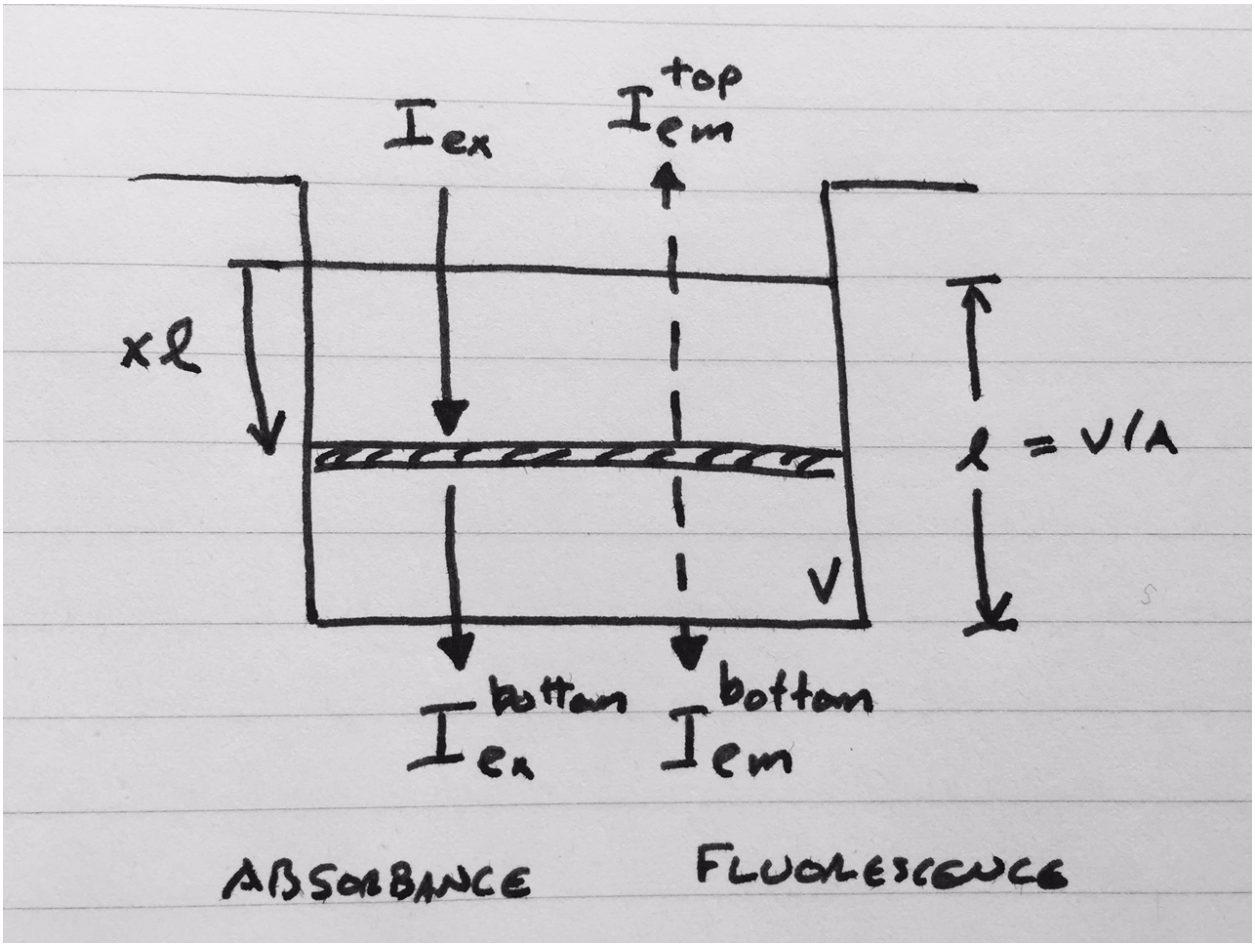

Fig. 2: Geometry and light intensities used for inner filter effect corrections.
This figure assumes top illumination, and depicts the incident excitation light intensity I_{ex}, transmitted light I_{ex}^{bottom}, and emitted fluorescent light toward the top I_{em}^{top} and bottom I_{em}^{bottom} detectors. The distance from the top liquid interface is expressed in terms of the dimensionless $x \in [0, 1]$ and the path length $l = V/A$, with liquid volume V and well area A.

Consider the shaded slice at depth xl depicted in the figure. The excitation light reaching this layer has intensity

$$I_{ex}(xl) = I_{ex} e^{-\epsilon_{ex} \cdot xl \cdot c}$$

where c is the concentration of the species with extinction coefficient ϵ_{ex} (where we are only considering the effects from the ligand species at this point, since its concentration can be high).

The secondary inner filter effect, because it considers absorbance at a different wavelength from the incident light, does not attenuate the absorbance.

If both primary and secondary inner filter effects are utilized, fluorescence is attenuated by a factor that depends on the detection geometry.

For top fluorescence, this is given by

$$\mathrm{IF}_{\mathrm{top}} = \int_0^1 dx\, e^{-\epsilon_{ex} \cdot x l \cdot c}\, e^{-\epsilon_{em} \cdot x l \cdot c} = \int_0^1 dx\, e^{-(\epsilon_{ex} + \epsilon_{em}) \cdot x l \cdot c} = \frac{1 - e^{-(\epsilon_{ex} + \epsilon_{em}) l c}}{(\epsilon_{ex} + \epsilon_{em}) l c}$$

For bottom fluorescence measurements, the path taken to the detector is different from the incident path length, so the attenuation factor is given by

$$\mathrm{IF}_{\mathrm{bottom}} = \int_0^1 dx\, e^{-\epsilon_{ex} \cdot x l \cdot c}\, e^{-\epsilon_{em} \cdot (1-x) l \cdot c} = e^{-\epsilon_{em} l c} \int_0^1 dx\, e^{-(\epsilon_{ex} - \epsilon_{em}) \cdot x l \cdot c} = e^{-\epsilon_{em} l c} \left[\frac{1 - e^{-(\epsilon_{ex} - \epsilon_{em}) l c}}{(\epsilon_{ex} - \epsilon_{em}) l c} \right]$$

Note: Just as with the *primary inner filter effect*, when $\epsilon l c \ll 1$, numerical underflow of the exponential becomes a problem. To avoid negative attenuation factors, a fourth-order Taylor series approximation of the exponential is used in computing the attenuation factor if $\epsilon l c < 0.01$.

Extinction coefficients

Extinction coefficients at excitation (and possibly emission) wavelengths are needed if either *absorbance measurements* are made or *inner filter effects* are used. These can either be measured separately and provided by the user or inferred directly as nuisance parameters.

Measured extinction coefficients.

If the extinction coefficients have been measured, we have a measurement ϵ and corresponding standard error $\delta\epsilon$ available. Because extinction coefficients must be positive, we use a lognormal distirbution to model the true extinction coefficients about the measured value

$$\epsilon \sim \mathrm{LN}(\mu, \tau) \;\; \text{where} \;\; \mu = \ln \frac{\epsilon^2}{\sqrt{\epsilon^2 + \delta^2 \epsilon}} \;\; , \;\; \tau = \ln \left[1 + \left(\frac{\delta \epsilon}{\epsilon} \right)^2 \right]^{-1}$$

Inferred extinction coefficients.

If the extinction coefficients have not been measured, they are inferred as nuisance parameters, with priors assigned from a uniform distribution with a large maximum and an initial guess based on the extinction coefficient of bosutinib at 280 nm

```
model['epsilon_ex'] = pymc.Uniform('epsilon_ex', lower=0.0, upper=1000e3, value=70000.0) # 1/M/cm
model['epsilon_em'] = pymc.Uniform('epsilon_em', lower=0.0, upper=1000e3, value=0.0) # 1/M/cm
```

1.2.2 Binding models

AssayTools has a variety of binding models implemented. Though the user must currently specify the model to be fit to the data, we plan to include the ability to automatically select the most appropriate binding model automatically using reversible-jump Monte Carlo (RJMC), which also permits Bayesian hypothesis testing. All binding models are subclasses of the `BindingModel` abstract base class, and users can implement their own binding models as subclasses.

Two-component binding model

A two-component binding model is implemented in `assaytools.bindingmodels.TwoComponentBinding`. When it is known that receptor *R* associates with ligand *L* in a 1:1 fashion, we can write the dissociation constant K_d in terms of the equilibrium concentrations of each species:

$$K_d = \frac{[R][L]}{[RL]}$$

Incorporating conservation of mass constraints

$$[R]_T = [R] + [RL]$$
$$[L]_T = [L] + [RL]$$

we can eliminate the unknown concentrations of free receptor $[R]$ and free ligand $[L]$ to obtain an expression for the complex concentration $[RL]$ in terms of fixed quantities (dissociation constant K_d and total concentrations $[R]_T$ and $[L]_T$):

$$K_d = \frac{([R]_T - [RL])([L]_T - [RL])}{[RL]}$$
$$[RL]K_d = ([R]_T - [RL])([L]_T - [RL])$$
$$0 = [RL]^2 - ([R]_T + [L]_T + K_d)[RL] + [R]_T[L]_T$$

This quadratic equation has closed-form solution, with only one branch of the solution where we require

$$0 < [RL] < \min([R]_T, [L]_t)$$

which gives

$$K_d = \frac{1}{2}\left[([R]_T + [L]_T + K_d) - \sqrt{([R]_T + [L]_T + K_d)^2 - 4[R]_T[L]_T}\right]$$

Note that this form is not always numerically stable since $[R]_T$, $[L]_T$, and K_d may differ by orders of magnitude, leading to slightly negative numbers inside the square-root. *AssayTools* uses the logarithms of these quantities instead, and guards against negative values inside the square root.

Competitive binding model

When working with N ligands L_n that bind a single receptor R, we utilize a competitive binding model implemented in `assaytools.bindingmodels.CompetitiveBindingModel`. Here, the dissociation constants K_n are defined as

$$K_n = \frac{[R][L_n]}{[RL_n]}$$

with corresponding conservation of mass constraints

$$[R]_T = [R] + \sum_{n=1}^{N} [RL_n]$$
$$[L_n]_T = [L_n] + [RL_n], n = 1, \ldots, N$$

The solution must also satisfy some constraints:

$$0 \leq [RL_n] \leq \min([L_n], [R]_T) \ , \ n = 1, \ldots, N$$
$$\sum_{n=1}^{N} [RL_n] \leq [R]_T$$

We can rearrange these expressions to give

$$[R][L_n] - [RL_n]K_n = 0 \ , \ n = 1, \ldots, N$$

and eliminate $[RL_n]$ and $[R]$ to give

$$\left([R]_T - \sum_{n=1}^{N} [RL_n] \right) \cdot ([L_n]_T - [RL_n]) - [RL_n]K_n = 0 \ , \ n = 1, \ldots, N$$

This leads to a coupled series of equations that cannot easily be solved in closed form, but are straightforward to solve numerically using the solver `scipy.optimize.fsolve()`, starting from an initial guess that ensures the constraints remain satisfied.

General binding model

A more general binding model is available in `assaytools.bindingmodels.GeneralBindingModel`.

A general series of N equilibrium reactions involving the interconversion of K components X_j, which may represent individual species or complexes, and have the form

$$K_n = \prod_{j=1}^{K} [X_j]^{s_{nj}} \ , \ n = 1, \ldots, N$$

with corresponding conservation of mass constraints

$$\sum_{j=1}^{K} c_{mj}[X_j] = q_m \ , \ m = 1, \ldots, M$$

This problem can be specified in terms of

- an $N - vector$ equilibrium constant vector $K \equiv (K_n)$
- an $N \times K$ stoichiometry matrix $S \equiv (s_{nj})$
- an $M \times K$ mass conservation matrix $C \equiv (c_{mj})$
- an M-vector $Q \equiv (q_m)$ of total concentrations across all species

For example, for a simple binding reaction $R + L \rightleftharpoons RL$, the species X_j are $\{R, L, RL\}$, and we have

$$K = [K_d]$$
$$S = \begin{bmatrix} +1 & +1 & -1 \end{bmatrix}$$
$$C = \begin{bmatrix} +1 & 0 & +1 \\ 0 & +1 & 0 \end{bmatrix}$$
$$Q = \begin{bmatrix} [R]_{\mathrm{tot}} \\ [L]_{\mathrm{tot}} \end{bmatrix}$$

A competitive binding reaction $R + L \rightleftharpoons RL$ and $R + P \rightleftharpoons RP$ with species $\{R, L, P, RL, RP\}$, we have

$$K = \begin{bmatrix} K_1 \\ K_2 \end{bmatrix}$$
$$S = \begin{bmatrix} +1 & +1 & 0 & -1 & 0 \\ +1 & 0 & +1 & 0 & -1 \end{bmatrix}$$
$$C = \begin{bmatrix} +1 & 0 & 0 & +1 & +1 \\ 0 & +1 & 0 & +1 & 0 \\ 0 & 0 & + & 0 & +1 \end{bmatrix}$$
$$Q = \begin{bmatrix} [R]_{\mathrm{tot}} \\ [L]_{\mathrm{tot}} \\ [P]_{\mathrm{tot}} \end{bmatrix}$$

Because the equilibrium constants K_n and concentrations $[X_j]$ must be positive but can range over many orders of magnitude, we represent these quantities by their logarithms, resulting in the equations

$$0 = -\log K_n + \sum_{j=1}^{K} s_{nj} \log[X_j] \;\; , \;\; n = 1, \ldots, N$$
$$0 = -\log q_m + \log \sum_{j, c_{mj} > 0} e^{\log c_{mj} + \log[X_j]} \;\; , \;\; m = 1, \ldots, M$$

The equilibrium concentrations $[X_i]$ is found as the solution to this set of equations using $y_j \equiv \log[X_j]$, solved by the numerical root-finding function `scipy.optimize.root()` using the vector-valued function $f(y)$ and its Jacobian $J(y)$:

$$f_n(x) \equiv -\log K_n + \sum_{j=1}^{K} s_{nj} \log[X_j] \;\; , \;\; n = 1, \ldots, N$$
$$f_m(x) \equiv -\log q_m + \log \sum_{j, c_{mj} > 0} e^{\log c_{mj} + \log[X_j]} \;\; , \;\; m = (N+1), \ldots, (N+M)$$

where the $\log \sum_{n=1}^{N} e^{a_n}$ operation can be computed in a numerically stable manner using the `scipy.misc.logsumexp()` function. The Jacobian $J(y)$ is given by

$$J_{nj} \equiv s_{nj} \;\; , \;\; n = 1, \ldots, N$$
$$J_{mj} \equiv \frac{c_{mj} e^{y_j}}{\sum_{k=1}^{K} c_{mk} e^{y_k}} \;\; , \;\; m = (N+1), \ldots, (N+M)$$

where the `scipy.misc.logsumexp()` function is once again used to compute rows $m = (N+1), \ldots, (N+M)$ in a numerically stable manner.

1.3 Useful Scripts

1.3.1 xml2png

Converts xml data file output from the Tecan Infinite M1000 Pro plate reader to png plot of fluorescence and absorbance values. It allows for the quick visual inspection of raw experimental results.

```
$ xml2png *.xml --singlet 'singlet_96'
```

1.3.2 quickmodel

Builds quick Bayesian model of both spectra and single wavelength two component binding experiments.

As input, it requires xml output files of the experiment form plate reader and a python script(*inputs.py*) that includes all experimental design details.

1. Run *calculate_L_stated_array* to generate ligand concentration array and copy it into *inputs.py*.
2. Construct *inputs.py* script based on experimental design.
3. Run'quickmodel'.

```
$ quickmodel --inputs 'inputs' --type 'singlet' --nsamples 10000
```

inputs.py

inputs.py should be manually constructed to record experimental design details, following the layout of of *inputs_example.py* file. Ligand concentration array (*Lstated* section) can be constructed using *calculate_L_stated_array.py* script.

Sections of 'inputs.py'

- *xml_file_path* : relative path to xml plate reader output files.
- *file_set* : option to group multiple experimental sets with a dictionary key.
- *ligand_order* : List of ligand names per each experiment set (one protein, one buffer row). If *None* Python object is specified as ligand name in this list, *quickmodel* analysis will skip the analysis of that experiment. For example:

```
'ligand_order'  :  [None, None, 'ligand3', 'ligand4']
```

- *section* : Data section label of Tecan Infinite M1000 Pro plate reader as specified in its method.
- *wavelength* : Emission wavelength picked for analysis (nm).
- *Lstated* : Experimental value of ligand concentration (M), in NumPy array form. It can be constructed using *calculate_L_stated_array.py* script.
- *L_error* : Estimated % error in stated ligand concentrations.
- *Pstated* : Experimental value of protein concentration (M).
- *P_error* : Estimated % error in stated protein concentration.
- *assay_volume* : Volume of each assay sample (L).
- *well_area* : Area of microtiter plate well (cm^2)

calculate_L_stated_array.py

Generates Numpy array of stated ligand concentration (Lstated) for logarithmic or linear dilution along a row. This numpy array is necessary to construct *inputs.py* file for *quickmodel.py* analysis. Provide information on how ligand titration is constructed: number of wells in each titration (*–n_wells*), highest and lowest ligand concentrations in molar units (–h_conc and –l_conc), and serial dilution mode (*–dilution*, linear or logarithmic) as inputs.

```
$ calculate_Lstated_array --n_wells 12 --h_conc 8e-06 --l_conc 2.53e-09 --dilution logarithmic
```

The numpy array this script prints out must be directly copied to *Lstated* section of *inputs.py*.

CHAPTER 2

API Reference

Note: This section is under heavy construction! For now, see the examples directory: https://github.com/choderalab/assaytools/tree/master/examples

2.1 Analysis Functions

AssayTools provides a number of analysis functions.

Here's an example

```
>>> import assaytools
>>> dataset = assaytools.load('dataset.xml')
```

CHAPTER 3

License

AssayTools is licensed under the Lesser GNU General Public License (LGPL v2.1+).