Notes on Determining and Using Force Constants for Constraints in Gromacs

MJV 8/18

1 Calculating Force Constants

- Data on a distance d can be described by a Boltzmann distribution,

$$p(d) = \frac{e^{-\frac{E(d)}{k_B T}}}{Z} \tag{1}$$

where

$$Z = \sum_{d=0}^{\infty} \Delta d \; e^{-\frac{E(d)}{k_B T}}$$

- Practically, we take the data and create a histogram ($h(d)$), which can be turned into a probability distribution function (PDF) through normalization,

$$p(d) = \frac{h(d)}{N} \tag{2}$$

where

$$N = \sum_{d=0}^{\infty} \Delta d \; h(d) \tag{3}$$

- Combining eqs. 1 and 2, we then have

$$-k_B T \; \ln \; p(d) = E(d) + k_B T \; \ln \; Z \tag{4}$$

which can then be fit to the Gromacs constraint of choice (eqs. 5, 6, 7) to find the desired force constant and ‘‘equilibrium” angle for the constraints.

- So, in practice we have:
  - Collect whatever angle/distance data you need a constraint for, and create normalized histogram.
    - For binning, I use the Rice rule, $b = 2n^{1/3}$, where b is the number of bins and n is the number of data points.
  - Get $E(d)$ using $-k_B T \; \ln \; PDF$ (this should invert the histogram data points.
  - Fit to the constraint function of choice (eqs. 5, 6, 7) to get k and θ_0/d_0.
    - See below for how to do this in Python

2 Gromacs Constraints

2.1 Restraints to fix angles

- Not harmonic
- Can use one or two pairs of atoms; one pair will fix pitch or yaw, two pairs will fix both
- Form is similar to fixing a dihedral:
  - one pair:

$$V_{\text{ar}}(\mathbf{r}_i, \mathbf{r}_j) = k_{ar}(1 - \cos(n(\theta - \theta_0))) \tag{5}$$

where

$$\theta = \arccos\left(\frac{\mathbf{r}_j - \mathbf{r}_i}{||\mathbf{r}_j - \mathbf{r}_i||} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\right)$$

- two pair:

$$V_{\text{ar}}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k, \mathbf{r}_l) = k_{ar}(1 - \cos(n(\theta - \theta_0))) \tag{6}$$

where

$$\theta = \arccos\left(\frac{\mathbf{r}_j - \mathbf{r}_i}{||\mathbf{r}_j - \mathbf{r}_i||} \cdot \frac{\mathbf{r}_j - \mathbf{r}_k}{||\mathbf{r}_j - \mathbf{r}_k||}\right)$$

- n is the multiplicity (2 doesn't distinguish between parallel and anti-parallel vectors). θ should be between 0 and 180 degrees for $n = 1$ and between 0 and 90 degrees for $n = 2$

2.2 Restraint to fix distances

- Use the function

$$E(d) = \frac{1}{2}k(d - d_0)^2 \tag{7}$$

3 Curve fitting in Python

- This method uses `matplotlib`, `numpy`, and `scipy` so the following examples will use

```
# to properly handle floating point division
from __future__ import division

import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
```

3.1 Creating a histogram

- In `matplotlib` a histogram is created using a 1D list of data and a number of bins.

```
data = [...] # some 1D data

# Use the Rice Rule to get the number of bins
numbins = int(np.floor(2*(len(data)**(1/3))))

# Create the histogram with plt.hist(data, **kwargs...)
# where **kwargs are optional arguments.
# density=True -> histogram should be normalized
# bins=numBins -> make histogram with this number of bins
# Note that hist will be saved as a 2D numpy array
hist = plt.hist(data, density=True, data)
```

- If your data set is particularly large, we need to set the ticks so that they are actually legible:

```
# this gets the current list of ticks on the x axis
xt = plt.xticks()[0]

# find the min and max
xMin, xMax = min(xt), max(xt)

# create an evenly distributed set of numbers
lnspc = np.linspace(xMin, xMax, len(data))
```

- This should produce, upon plotting, something like Fig. 3.2.

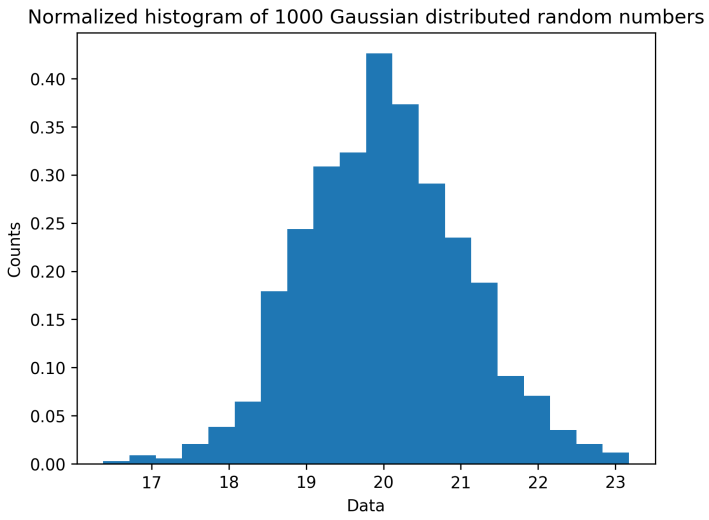

Figure 1: Example of a normalized histogram created in Python

3.2 Fit

- Since the `hist` that `matplotlib` creates has bin edges as the x-axis data, we need to get the bin centers before we get the data to fit.

```
    # Get bin centers for the new x-axis
    binCenters = 0.5*(hist[1][1:] + hist[1][:-1])

    # Calculate the energy, -ln(pdf)kT
    # Note that log is numpy's ln (I don't understand why)
    invertHist = -np.log(hist[0])*0.008314462*310
```

  - Upon plotting, this should produce something like fig. 4.

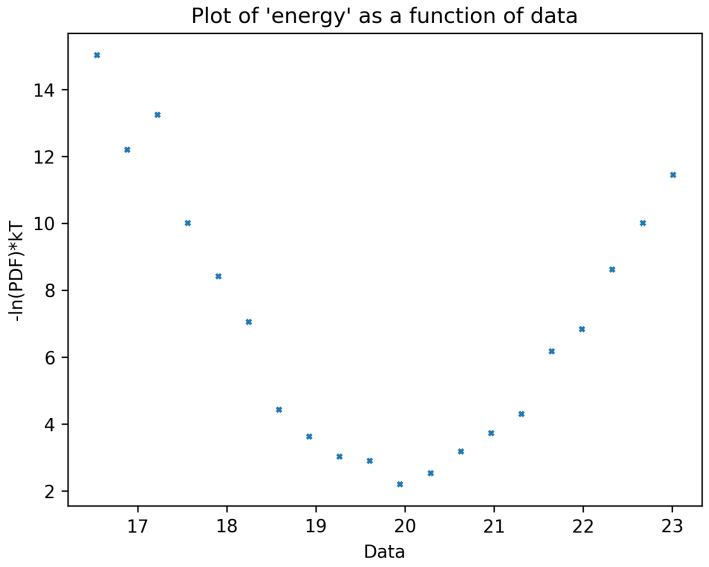

Figure 2: Example of the 'inverted' histogram, using eq. 4.

- Once we have this data, we can fit the constraint function to it to calculate k and the ‘‘equilibrium” value of the distance/angle.
  - We first need to remove any points in `invertHist` which are infinite (I’m sure there’s more efficient and ‘‘Pythonic” way of doing this, but it’s how I did it):

```
    # Get the indices of the data points to delete
    delete = []
    for i in range(0, len(invertHist)):
        if (np.abs(invertHist[i]) == np.inf):
            delete.append(i)

    # Sort them from biggest to smallest so we don't mess up
    # the numbering as we delete
    delete.sort(reverse=True)

    # Delete them
    for i in delete:
        invertHist = np.delete(invertHist, i)
        binCenters = np.delete(binCenters, i)
```

  - We need to define the function we want to fit to the data (I’ll use eq. 5 as an example).

```
def angle_fit(theta, k, theta_0, c):
    return k*(1-np.cos((np.pi/180)*(theta-theta_0))) + c
```

  - Now let’s fit the parameters to the data. We’ll be using `scipy.optimize.curve_fit`.

```
    # curve_fit(function, x, y) returns a tuple of the parameters, and the [0]
    # at the end captures each and places it into the individual float
    k, theta_0, c = curve_fit(angle_fit, binCenters, invertHist)[0]

    # Plot this using the angle_fit function
    plt.plot(lnspc, angle_fit(lnspc, k, theta_0, c))
```

  - If you want to use bounds on the parameters, include `bounds=(min,max)`, which is a tuple Ours would look like, using 0 as the min for all,

```
k, theta_0, c = curve_fit(angle_fit, binCenters, invertHist, \
    bounds=(0,[2000., 360., np.inf]))[0]
```

    using `np.inf` for when we don’t need a bound for it (or `-np.inf`, whichever the case may be).
  - This should produce something like fig. 3

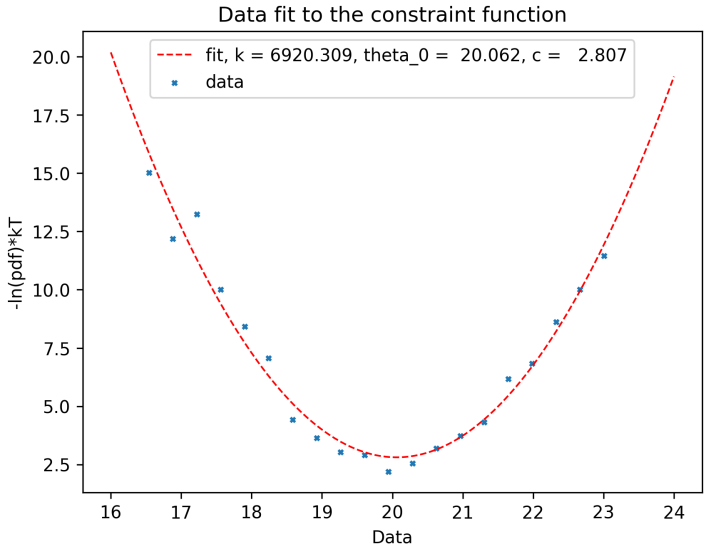

Figure 3: Example of fitting the energy function (eq. 5) to the data to obtain k and θ_0.