AS.020.674 Spring 2017

Lab 10

Due April 28 in lab. Submit your answers as either a tgz file containing the python scripts and associated figures or as a single working Jupyter notebook file.

In [1]:
```
# Standard iPython notebook preamble.
%matplotlib inline
from __future__ import division
import matplotlib
import numpy as np
from pylab import *
matplotlib.rcParams.update({"axes.formatter.limits": (-3,3)})
plotStyles={"markersize":10,"markeredgewidth":2.0,"linewidth":3.0}
stepStyles={"markersize":12,"markeredgewidth":3.0,"linewidth":3.0,"wher
e":"post"}
```

Introduction.

In this lab we will be running kinetic simulations of an individual single molecule undergoing a folding and unfolding reaction. We will save the time series data and analyze them to learn about the kintic rates.

We will model a single molecule folding and unfolding using two first order reactions:

1. U→F with rate constant k_1
2. F→U with rate constant k_2

Below is a Python script that simulates the kinetics of a single protein folding and unfolding.

In [11]:
```
# Set k1 and k2 for the notebook.
k1=0.15
k2=0.07
```

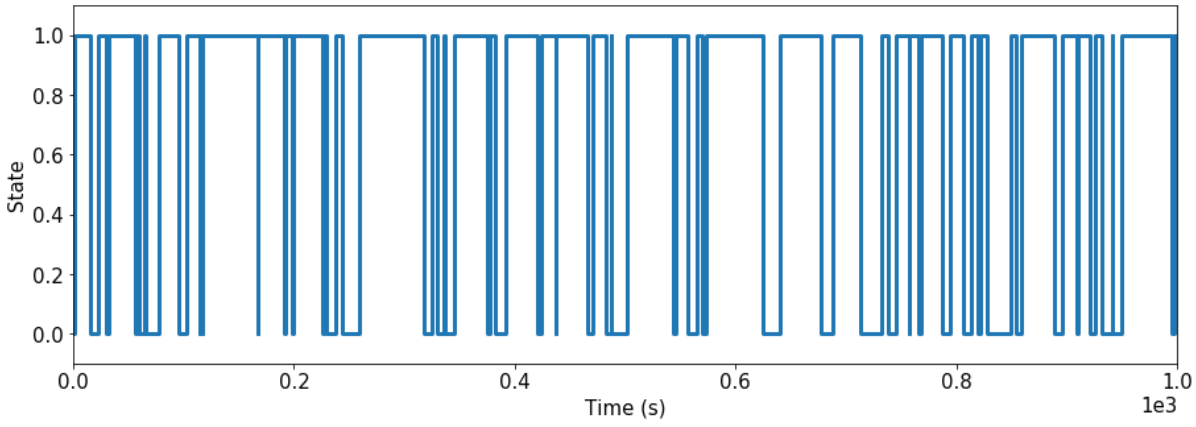
In [40]:
```
import numpy.random as rnd

def performFoldingSimulation(k1,k2,tf):
    ts=[0.0]   # a list of the times when a state change has occurred; initial time is 0.0
    states=[0] # state 0 is unfolded, state 1 is folded; initial state is unfolded
    while (ts[-1]<tf):

        # If we are in the unfolded state, figure out when the molecule transitions to the folded state.
        if states[-1] == 0:
            ts.append(ts[-1]+rnd.exponential(1/k1))
            states.append(1)

        # If we are in the folded state, figure out when the molecule transitions to the unfolded state.
        else:
            ts.append(ts[-1]+rnd.exponential(1/k2))
            states.append(0)
    return (ts,states)

# Plot an example trace.
tf=1000.0
matplotlib.rcParams.update({'font.size': 15, "figure.figsize": (15,5)})
ts,states = performFoldingSimulation(k1,k2,tf)
step(ts,states, **stepStyles)
xlabel('Time (s)'); xlim([0,tf]); ylim([-0.1,1.1]); ylabel('State')
pass
```



Read through and try to understand the code above. If you are not using Jupyter notebook, save the script to a file and make sure you can run it, you may need to adjust the plotting code for your preferred plotting method.

Problem 1

Run a folding simulation for 1000 seconds. Create a list of waiting times (the time between transitions) in both the folded and unfolded state. Use the np.histogram function to calculate a histogram of the waiting times for both states, using 100 bins.

1. Plot the two histograms as bar plots. Hint: the np.histogram function returns the bin edges, you will want to plot the bars located at the bin centers. Hint 2: for n bins there will be n+1 edges and n centers.

Problem 2

The histogram function returns the number of counts in each bin. To compare with our kinetics equations, we need to convert the counts into a probability density function (PDF). To do so, divide the histogram counts by the sum of all the counts **and** also by the width of the bin. The bins are all the same size by default, so you can divide each count by the width of the first bin.

1. Plot the calculated PDFs for the folding and unfolding transition times as a bar plot.
2. On top of the bar plots, plot the PDF for an exponential distribution: $p(t) = k \cdot e^{-k \cdot t}$, where k is k_1 or k_2 as appropriate.

Problem 3

Write a python script to fit your calculated PDFs to exponential distributions to estimate k_1 and k_2.

1. What were the k_1 and k_2 values you obtained from fitting?
2. What is the relative error in your estimated k_1 and k_2 vs the actual k_1 and k_2?
3. Plot the calculated PDFs as bar plots overlaid with the best fit estimates.
4. Change the time of the simulation to 10000 seconds and run the analysis again. Now what are the estimated errors in k_1 and k_2?