Statistical Analysis

Instructions

In this homework you will start by writing your own data analysis tool to measure mean, standard deviation, autocorrelation time and error of mean. You will use your tool to analyze a few data sets which are representative of simulation output.

Write Your Own Code for Statistical Analysis

You can use whatever language you want. But remember the officially supported language is Python. Here we only provide instructions on using Python.

The elementary statistical analysis operations of obtaining the mean, standard deviation, autocorrelation time, and error in the mean of a dataset are relatively straightforward to program. Although software such as Dataspork or the scipy.stats package exists to perform this analysis, you will have a deeper understanding of how to compute and interpret these quantities after you have programmed them yourself.

In this question, you will write a function that takes an array of scalar data points and computes the *mean, standard deviation, autocorrelation time, and estimated error* in the mean. You can use the datasets provided to test your code and validate that your results agree with another software you prefer (Python, Matlab, Mathematica etc.).

Note: Due to differences in convention, you may get autocorrelation times that do not agree exactly with each other; but it should be close in most cases!

If you decide to use Python, then you may follow these instructions:

1. Download analyze_trace.py and put it in your working directory.
2. The line "from __future__ import print_function" to the very beginning of the python script is only needed if your Python version is below 3.0 but above 2.6. Otherwise python2.7 for example will complain about SyntaxError for print().
3. You need to edit functions in analyze_trace.py to calculate the standard deviation, auto correlation time, and the standard error.
4. Test that your python implementation is working correctly by calling

   ```
   python analyze_trace.py AtomicScale_HW1_data1.txt
   ```

   for example, where "AtomicScale_HW1_data1.txt" is the name of the first provided data file.

---

1. Mean

We provide example code for computing the mean of a one-dimensional array of floating-point numbers (floats); the data are contained in the array "trace", which is the argument of the function "mean". If you are unfamiliar with Python, use this block of code to begin working on the problem:

```
def mean(trace):
     """ calculate the mean of a trace of scalar data
           results should be identical to np.mean(trace) """

     # pre: trace should be a 1D iterable array
     total = 0.0        # store sum of all data points
     num   = len(trace) # count total number of data points

     for i in range(num):
       total += trace[i]
     # end for i

     return total/float(num)
 # end def mean
```

Notice the following things:

- White spaces are important, python uses indentation to recognize code blocks. We recommend always using white spaces instead of tabs (you can achieve this in vim by "set tabstop=4 shiftwidth=4 expandtab" in .vimrc). You must indent underneath *function calls* and *for loops* (typically four spaces). This is designed to force the code to be readable.
- We welcome comments. Single line comments start with #, multi-line comments are enclosed by triple quotation marks. Comments make the code understandable to humans (including your future self!). The first multi-line comment of a function is considered a "doc string" in Python. One may display the doc string with mean.__doc__. In Jupyter notebook, one can shift+tab at *mean(* to reveal the doc string.
- To define a function, use "def my_function_name(param1, param2)". Notice that no value type were specified for the input parameters. In fact, you never need to specify types in python.

2. Standard Deviation

You should review a textbook on elementary statistical analysis if you are unfamiliar with standard deviation. You can find the following equation on this wikipedia page, which is an estimation of the standard deviation given a set of data points

$$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Implement this computation in your python module.

```
def std(trace):
     """ calculate the standard deviation of a trace of scalar data
     results should be identical to np.std(trace,ddof=1)
       pre:  trace should be a 1D iterable array of floating point numbers
       post: return the standard deviation of this trace of scalars
     """
     stddev = 0.0
     # calculate stadard deviation
     return stddev
 # end def std
```

To use this function, you would call:

```
mystd = std(trace)
```

3. Autocorrelation time

Elementary statistical analysis assumes uncorrelated (i.e. statistically independent) data points in your set. In many practical cases, and especially in analyzing simulation data, correlations will be present that will cause you to underestimate your error bars significantly unless they are treated correctly.

If you are unfamiliar with *autocorrelation*, you can find the following equation on this wikipedia page,

$$\hat{R}(k) = \frac{1}{(n-k)\sigma^2} \sum_{t=1}^{n-k} (X_t - \mu)(X_{t+k} - \mu)$$

Then the *autocorrelation time* is defined as $\tau = 1 + 2\sum R(k)$, where the summation is over the index k, from 1 to a cutoff Λ. Here we will choose the $(\Lambda+1)$ as the smallest k that makes $R(k) \leq 0$.

We take the interpretation that data points separated by an interval equal to or larger than the cutoff are uncorrelated.

Important Notice: This convention of choosing the cutoff is not universal. Remember this is not a fundamental result but a practical method to *estimate* the autocorrelation time from a *finite* data set. Our approach of choosing the cutoff at the first zero crossing is straightforward and easy to understand, but there are other, more sophisticated schemes you can cook up. This means your autocorrelation time **will not** agree with what you get from other software. So that you can validate your code against something, we will give a you a checkpoint. If you compute the autocorrelation time of data set 2 using our scheme, you should get an autocorrelation time of 19.735512755 (updated from the old incorrect value of 19.754267).

Heuristically, the autocorrelation time is a measure of how much statistically significant data is present in each data point. Specifically, the *effective number of statistically independent data points* in a set is N_eff = N/(autocorrelation time).

Note that the mean and standard deviation can be computed with one loop over the dataset. However, to compute the autocorrelation time, you must already know the mean and standard deviation, so an additional loop through the dataset is needed.

Implement this function.

4. Standard Error

The standard deviation is indicative of the fluctuations inherent in your dataset. The standard error reflects the fact that, in spite of any such fluctuations, the error in your estimate of the mean decreases as you include more data points (i.e. a larger sample size gives better statistics). It is the error you quote on your answer.as

Having worked up to this point, you can easily compute this error, as well. You need the standard deviation σ and effective number of data points N_eff. Then the error in the mean is σ/sqrt(N_eff).

Your code should now compute the mean, standard deviation, autocorrelation time, and error in a scalar dataset

5. Visualization

It is often instructive to look at the data you are analyzing whenever possible. "A picture is worth a thousand words."

After you have finished implementing the above functions, call analyze_trace.py with the -p or --plot flag. This will instruct the code to plot the trace along with the calculated statistics. You will need to have matplotlib installed.

visualization
```
 python analyze_trace.py AtomicScale_HW1_data1.txt -p
```

Analyze Data

Dataset 1
The initial cutoff (first data point of the array) and end cutoff (last data point of the array) should be automatically set to 0 and 999 by your code (the first data point is numbered 0, and this dataset only has 1000 data points). Note the initial/end cutoff here is not the same cutoff as defined in calculating autocorrelation time. The autocorrelation time should be nearly one for this dataset.

analyze_trace.py can take additional arguments like initial cutoff and end cutoff, you should figure out how to set them.

- What is the mean value of this data?
- What is the standard error in the estimate of the mean?
- What is the standard deviation of the data?

Change the end cutoff to 500 (which means we only do statistical analysis from data point 0 to data point 500).

- How does the standard error in the estimate of the mean change?
- How does standard deviation change?
- How would you expect these quantities to change as you gradually increase the initial cutoff from 0 to 500?

Dataset 2
This data set has correlations.

- What is the autocorrelation time of this series?
- What would be the standard error in the estimate of the mean without considering autocorrelation?
- Now what would be the error in the estimate of the mean with correlation?

Dataset 3
This data set has an initial transient.

- What is the mean (and its error) with the initial cutoff set to zero?
- Where should the initial cutoff be set?
- What is the mean (and its error) with the new cutoff?
- Is the difference between the mean in part 1 and part 3 significant?

Dataset 4

This data set was sampled from the distribution P(x) = b/(|x|^a + c) with a = 2.2 and c = 1.0, b is determined by the normalization.

- Based on this analytic expression, what do you expect for the mean? the variance? (Hint: for the variance, it's the behavior at large x that matters, so one can use an approximation for the denominator of the distribution function.)
- Look at the convergence of the mean and sigma by computing these values for five "end cutoffs" from 1000 to 5000 (i.e., 0-999, 0-1999, 0-2999, etc). Do the same for data set 1 or 2 and compare the convergence behavior?

Comparison of Datasets

This problem is meant to highlight the impact of Gauss' Central Limit Theorem, i.e.9

Given a population with a mean μ and a finite, non-zero variance σ, **the sampling distribution of the mean approaches a normal distribution** with a mean of μ and a variance of $\sigma' = \sigma/(N-1)$ as N, the sample size, increases.

The words in bold are critical... the estimated mean approaches a Gaussian distribution as more points are used.

In this question, do the computations by hand on your calculator.
Suppose when debugging a code, you have 2 versions A and B which you run 6 times each to try to determine if they give the same answers. you get the following answers:

A	**B**
1.12	1.44
1.52	1.34
1.33	1.19
1.09	1.13
1.20	1.56
1.26	1.45

- Compute the mean, variance, and the estimate of the error of the mean for A and B separately, assuming each run is **uncorrelated** with the others.
- Show that the probability that the two runs are (NOT) drawn from the same distribution is ~ 29% (71%).

To do this, first find how many standard deviations (S.D.), x ,the difference is from zero; do this by dividing the "estimate of the difference" by the "estimate of the error of the difference". From this number determine the probability the two are from the same distribution using a Normal Standard Probability Distribution Table (often referred to as P(0,x)) or the Error Function erfc((x/sqrt(2)).

A detailed explanation of how to do this can be found here.

If you need more background, the following books have good section on the subject. All of them can be found in the University libraries:

- M. Boas, *Mathematical Methods in the Physical Sciences* (good old classic for physics majors, suits for most purposes)
- Barlow, *Statistics: a guide to the use of statistical methods in the physical sciences* (a short book dedicated to experimental data analysis)
- Hogg and Tanis, *Probability and statistical inference* (used in STAT 400, more in-depth)
- Devore, *Probability and statistics for engineering and the sciences* (another in-depth book with lots of derivations and examples)

Assuming that all the data is drawn from the same distribution, estimate the mean and the error of the mean of the combined data set.

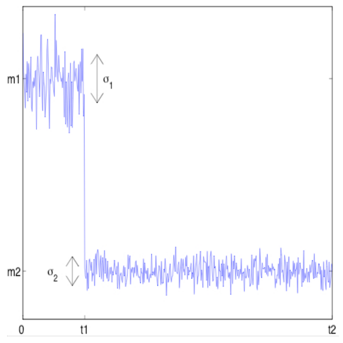
Being able to perform these last two calculations is fundamental to using and understanding error bars based on the concept of normal probability distributions provided in link.

Bias From Unequilibrated Data

Hint: You should work directly with the analytical expressions for the standard statistical formulas.

In many-particle simulations there is often an equilibration period in which the initial configuration of the simulation biases the output. The data from this period should be discarded before calculating statistics. The following exercise is to study the effect of neglecting to discard this data properly.

The figure shows a plot of the trace of a scalar quantity (e.g. energy) as a function of simulation time. The interval [0,t_1] is the unequilibrated period, in which the mean and standard deviation are m1 and σ1, respectively.
The interval [t1,t2] is the "good," equilibrated data, in which the mean and standard deviation are m2 and σ2. This mean is the true mean we are trying to calculate.



We imagine that we do not discard the first interval, even though we should, and now estimate the mean and standard deviation for the whole interval [0,t2]. We denote these quantities m and σ, respectively.

- Give an expression for m as a function of m1, m2, t1 and t2. Give also an expression of the systematic error, ε = m - m2, that comes from including the bad data, in terms of the *prolongation* λ = t2/t1 and Δ = m1 - m2.
- Give an expression for σ, grouping your terms in inverse powers of λ.
- Consider t2 » t1, i.e. λ » 1. At what time t2 will the *statistical error of the mean* equal the *systematic error ε from the unequilibrated data*, utilizing the lowest nontrivial order of approximation. Don't forget the autocorrelation time τ.
- With τ=1 and t1 = 100, estimate this time by eyeballing the appropriate ratio from the plot.

(You may wish to print this Figure and use a ruler to estimate.)