Laboratory 3: Moving Averagers

Goals

1. To create and document a program that transforms a sequence of scalar inputs into a moving average;
2. To understand the interaction between the rate parameter of the moving averager and the variance of the input sequence as it affects the statistics of the moving averager's output;
3. To demonstrate that a moving averager reduces fluctuation and smoothes a random sequence; and
4. To gain intuition about the concept of stochastic convergence.

In the last lab, we laid the foundation for neurons as decision-makers. This week we lay the foundation for memory storage via synaptic modification. Next week, we will begin to bring all of the pieces together by creating simple network models that can learn about their inputs and then make a decision.

Introduction

Everyone is familiar with the idea of an average. It is so common that we rarely even think about what it means. However, if we stop to think about the properties of an average in terms of cognition, some interesting points emerge: averages have memory and they are abstractions.

An average is a function of the history of a set of events. Often these events occur sequentially. For example, we can follow the baseball statistics across the season and watch the averages change with each game played. Importantly, such an averaging process has a memory of the past. Indeed, in Laboratory 4, we will use an averaging process to understand memory storage at synapses.

Secondly, an average is really an abstraction. You could take five physics tests, and your average score over these five tests could easily be a number that wouldn't match any of the tests you took. Thus the 'average test score' might never have occurred as any one particular test score; That is, it does not really exist-it is an abstraction. It is, to use cognitive terms, a prototype that is representative of the category. It is even, arguably, the best representation although it has never occurred.

Also important to cognition is the idea of memory—the average represents a way of storing a large amount of data with great efficiency. If you wanted to have some idea of an entire set of 1000 test scores, for example, you could either store each of the 1000 test scores, or you could store some number that would describe this set of scores. The average is one such number. Therefore, an average can be used to predict the future. Suppose there are two students, Adam and Becky. Adam's cumulative GPA is 2.0 and Becky's is 3.9. If they were both enrolled in the same class, which student would you expect to do better? Forced to guess, most would expect Becky to do better, and we can make this prediction based upon two pieces of information. The first is the respective grade point average of the two students, and the second is the assumption that these averages mean something with respect to the future performance of these students.

Consistent with our credo of hierarchical recapitulation (see page 3 of lab 2), we will look for a microscopic version of averaging. That is, we will explore the notion that a neuron or its parts can act as some sort of averaging device.

To summarize, an average is both a form of memory and an abstract entity that is a prototypical representation of a category. Moreover, the characterizations of an average also hold for a *moving average*. Since many physical processes are, in the abstract, moving averagers, it is not surprising that natural selection can incorporate such functions into biological computations.

Section 3.1. The moving average

A minimal definition of learning is change as a function of experience. Learning is the dynamic aspect of cognition, whereas memory is what exists when learning is tested. In this laboratory, we are concerned with a dynamic process; we explore how an average changes as a function of its sequential experiences, or equivalently, its particular history of sample values. The moving averager investigated here is an example of non-associative memory storage—next week, we will look at associatively-based memory storage.

In the present lab, your simulations will produce a moving averager in terms of what it learns, or encodes. Specifically, it learns a statistic of a particular stationary environment. Metaphorically, this environment is our world model, and this model controls, or generates, the input sequence that our brain model, the moving averager, experiences.

Finally, we must comment that memory and moving averagers are not some special property or ability exclusive to biological environments. In a sense, the temperature of an outdoor, unheated swimming pool displays a memory by producing a moving average of the temperature of the surrounding environment. In the summer, the pool heats up, but not as quickly as the air around it. The day-to-day fluctuations in the temperature of the pool are much smaller than the day-to-day fluctuations in temperature of the environment. The larger the pool is, the slower the water temperature changes. The pool temperature is a moving averager of the sequence of daily temperatures. Apparently, the size of the pool helps to determine the rate constant of a moving averager. There are many physical processes that act like such an averager, including electrical circuits and some simple chemical reactions, both of which are available to neurons. Just like these processes, a moving averager exhibits what we call exponential forgetting.

Since you already know about creating an average, let's relate this knowledge to a moving average

.

Sequentially updating an average

How is an average computed? For a given set of N individual values $x(i),\ i \in \{1,2,\cdots,N\}$, you simply sum all the values and divide by the number of individual values. In a more compact form to obtain the average (*Ave*),

$$Ave[x(i)|N] := \frac{1}{N}\sum_{t=1}^{N} x_t \tag{3.1}$$

where we abuse notation after the vertical "conditioning" line (read the vertical line as the word 'given') and use N to indicate N sequential samples starting with the first one and ranging through the Nth one.

This is a correct way to get an average (properly called the *sample mean*), but what if you need an average after every new value? For example, suppose you're trying to figure out a life-long cumulative test average. It would be inefficient to remember all the individual test scores and, each time a new test occurs, to sum all the individual scores. Better and minimizing memory requirements, we want a way to calculate the average that requires only minimal computations with the addition of each new score, or 'sample.' In fact, such a formula does exist, and it is a clever manipulation of the original average formula. You might remember implementing this formula to update the current average from Laboratory 1.

The average of some set of values, $Ave[x(i) \mid N+1]$ with *i* ranging from *1* to *(N+1)*, is:

$$Ave\left[x(i) \quad \middle| \quad N+1\right] = \frac{\sum_{i=1}^{N+1} x(i)}{N+1} = \frac{x \quad (N+1)}{N+1} + \frac{\sum_{i=1}^{n} x(i)}{N+1}$$

$$= \frac{\mathrm{x}(\mathrm{N}+1)}{\mathrm{N}+1} + \frac{N}{N+1}\cdot\frac{\sum_{i=1}^{N} x(i)}{N}$$

$$= \frac{1}{N+1}\cdot x(N+1) + \frac{N}{N+1}\cdot Ave[x(i)|N] \tag{3.2}$$

$$= \frac{1}{N+1}\cdot x(N+1) + (1-\frac{1}{N+1})Ave[x(i)|N]$$

Homework 3.1. The recursive true averager

Now generate 1,000 random numbers and calculate the mean of the sample using a 'for' loop instead of MatLab's `mean` command. In other words, update the mean as you iterate through the elements in the vector, starting at 1 and ending at 1,000. Plot the mean as a function of sample number and comment on how the mean changes as you iterate through the sample.

```
>>VectorC = rand(1000,1);
               %generate a column vector of 1000 random
               %numbers and suppress output; set VectorC equal
               %to this set

>>CurrentAve=0;
               %initialize CurrentAve to zero
>>for I=1:1000
    CurrentAve=(CurrentAve*(I-1)+VectorC(I,1))/I;
               %update CurrentAve by adding the value of
               %CurrentAve at the previous timestep to the
               %Ith value of VectorC and dividing by I, the
               %sample number
    StoredAve(I,1)=CurrentAve;
  end
>>plot(StoredAve)
               %the plot command is placed outside the loop to
               %speed the program
```

Check that this program is actually working. Use the command

```
>>mean(VectorC(1:N,1))
```

to calculate the mean of the first N values stored as part of the vector named VectorC. Explain why the averaging algorithm works. (Hint: Write the formula for an average as you have been taught $\sum_{i=1}^{n}\frac{x_i}{n}$ and show equality to the algorithm).

Notice how the new average is now an updating of its old but most recent value based on the newest sample. Let's call this the sequential updating algorithm for the mean. It drastically reduces the number of addition operators needed when N is large because it only requires us to keep track of the last average and N. That is, we can dispense with remembering all previous samples.

Now that the operation is simplified to the recursive form of (3.2), we begin to wonder if there is a way that a biological or physical process could imitate this algorithm. However in light of the biology of neurons, there is one problem; it is unreasonable to assume that a neuron could act as an averager using this formula because it requires a neuron or a synapse to keep track of the precise value of the ever changing N (i.e., how many items it has incorporated as samples). How big is N? Count the number of things you have looked at in your life. Suppose neurons had to keep track of the number of times you moved your eyes (saccaded) to look (fixate) at something—anything—through out your whole life up to the point just now. Go ahead and calculate it, and suppose you fixate twice a second for your entire waking life. It's a big number; it has lots of digits to keep track of. Instead of tracking the exact average which requires the exact N, suppose the $\frac{1}{N+1}$ term is replaced by an appropriate constant, then it is reasonable to hypothesize that a neuron could approximate the sequential updating algorithm for the mean. Let' see how it would be done.

Return to Equation 3.2

$$Ave[x(i) \mid N+1] = \left(1 - \frac{1}{N+1}\right) Ave[x(i) \mid N] + \frac{1}{N+1} \cdot x(N+1)$$

And now replace the always shrinking $\frac{1}{N+1}$ by the constant ε that is small, greater than zero but less than one. The formula, which we rename MAve, then becomes:

$$MAve[x(i) \mid N+1] = (1-\varepsilon)\, MAve(i) + \varepsilon\, x(N+1), \tag{3.3}$$

where ε is called the *rate constant*. We will always choose a value of ε between zero and one, 0< ε <1, and this is sensible since ε is replacing $\frac{1}{N}$, which is in the range after the first sample.

To understand how the rate constant ε works, let's consider two extreme cases, zero or one, which in fact are not allowed. If you substitute into Equation 3.3 the value ε=0, then the equation becomes:

$$MAve[x(i)+1 \mid N+1] = (1-0) \cdot MAve[x(i) | N] + 0 \cdot x\ (N+1)$$

$$MAve[x(i) \mid N+1] = MAve[x(i) \mid N].$$

In this case, the value of the moving averager will never change. On the other hand, if you use ε=1 and apply this to Equation 3.3, then the equation becomes:

$$MAve[x(i)\ | N+1] = (1-1) \cdot MAve[x(i) \mid N] + 1 \cdot x(N+1)$$

$$MAve[x(i+1) \mid N+1] = x(N+1).$$

Now the moving averager completely disregards the past and moves exactly to the latest score just as if it has 'forgotten' previous scores. Of course no one would ever use the extreme values just described. But for a value between these two extremes we hypothesize that a large value of ε quickly forgets the past while a moving averager equipped with small value of ε is slow to forget in the sense that it is slow to change.

The parameter ε will be important in the neural networks you build. As you've seen in the example with extreme input values, the rate constant has a powerful effect on a moving averager and its memory because this constant controls how quickly the process 'forgets.' You will see more evidence for the strong effect of the rate constant as you work through the exercises in this laboratory.

About the averaging approximation

You could say that Equation 3.3, the moving averager, is not a good averager because its value will rarely, if ever, be correct. But consider what happens when ε is very small, which we equate to N being very large. In this case, the influence of each individual score decreases as the N increases. Indeed,

$$\lim_{N\to\infty}\left(\frac{1}{N+1}\right)=0.$$

Moreover, small ε might be a very good approximation for a small value 1/(N+1). In fact, relative to the moving averager staying close to the mean, a smaller ε produces a more precise approximation.

Because ε is a non-zero value, a moving averager is adaptable and responsive to changes of the process that is generating the input. Suppose that Adam, the student described earlier, has been keeping a lifetime average of his test scores. After nearly 15 years of schooling, Adam has taken 500,000 tests. Suppose further that Adam, with his 2.0 average, was struck by lightning last Thursday and is now a genius. From that point forward, Adam's test scores will be much higher than his previous ones. If his average were calculated using the original formula of Equation 3.2, years of perfect scores would be required before Adam's average would approximate the scores he received after his accident because of the decreasing influence the (1/(N+1)) term has on each new score before it is added to the average. In contrast, using Equation 3.3 with a constant ε greater than 1/500,000, e.g. 1/50, many fewer additional scores would be needed for the average to reflect Adam's post- accident test-taking abilities.

The moving averager algorithm of equation 3.3 can be made more intuitive by rearranging the terms so we can factor ε.

$$MAve[x(i)\mid N+1]=MAve[x(i)\ |N]+\varepsilon\left(x(N+1)-MAve[x(i\mid N)]\right) \qquad (3.4)$$

Now this rearranged equation is sensibly conceptualized as

NEW Average = Current Average + Change, where

$$\text{Change} \overset{def}{\equiv} \Delta MAve = \varepsilon\left(X(N) - MAve\left[x(i) \mid N\right]\right) \tag{3.5}$$

This simple relationship is very important; the new value is equal to the old value plus the change. We can now stare into the heart of the process and imagine what it takes for $\Delta MAve$ to be positive, negative, or zero. In your laboratory write-up, be certain to address this fundamental insight.

We begin the lab with a uniform probability generator over the interval [0,1). The nominal mean of this generator is 0.5, and its nominal variance is 0.0833. As shown in Laboratory 1, the empirical mean and empirical variance of your sample will, with very high probability, differ somewhat when you take only a finite number of samples. Using exactly the set of input data (the random number), let's now compare the true averager to a moving average.

Section 3.2. Creating a moving averager and understanding its rate constant

The moving average `mave` is a recursive algorithm that looks very much like the true averager. It is an averager that must make a computation at each timestep. Let's write it in a way that might appear in a MatLab function

mave(t+1) = mave(t) + Δmave(t),

where Δmave(t) is calculated as

Δmave(t+1) = ε * (input(t) - mave(t)).

As noted above, the performance of a moving averager depends on a rate constant designated, $\varepsilon \in (0,1)$. Epsilon (ε) is a rate constant in the sense that its value controls the rate at which successive values of the moving averager approach (or move away from) the mean of the process generating the input. Here we begin to investigate the relationship between the value of ε and the performance of the moving averager.

First we need some data to average. Then you will feed each datum, in sequence. into a moving averager. We want to observe what happens so we must save each sequential updating of the averager. Call the value of the moving averager `ybar.` We will update `ybar` with 100 samples. As a result, `ybar` is a vector of size 101, not 100, because it must first be initialized. Get MatLab running and follow along entering the statements at successive prompts.

```
>>ybar(1,1) = 0
                  %choose zero as the initialization value; %MatLab
                  does not allow the zero position
                  %designation; e.g., ybar(1,0) is not legal
>>epsilon=0.1
                  %set the rate constant at 0.1 for now
```

Generate 100 inputs for the moving averager and just for fun, rescale the inputs to the range [0,0.5). Call these numbers `z,` a vector of dimension 100.

```
>>z = 0.5 * rand(1,100);
                  %create a set of random numbers with a
                  %reduced range
```

Operate the moving averager for one timestep.

```
>>ybar(1,2) = ybar(1,1) + epsilon * (z(1,1)-ybar (1,1) )
                  %calculate the first new value of the %moving %averager
```

Update the average with the second input.

```
>>ybar(1,3) = ybar(1,2) + epsilon * (z(1,2)-ybar(1,2))
```

Although the average can be updated in this way with each successive input, the method is tedious. The following method is more efficient—we will use a 'for' loop that iterates through all 100 samples.

```
>>for i = 1:100,
      ybar(1,i+1) = ybar(1, i) + epsilon * (z(1,i)-ybar(1,i));
  end
```

To further analyze this moving averager, have your program calculate the statistics. To determine what happened, compare the final value of `ybar` to the mean of `z`:

```
>>mean(z)
>>ybar(1,101)
>>(mean(z)-ybar(1,101))/mean(z)
                  %here's the fractional inaccuracy of the moving %averager
                  compared to the true mean
>>plot (ybar)
                  %now let's look at the dynamics of the moving
                  %averager so we plot its state as a function of %time
>>hold on
>>plot(z,'o')
                  %here we show each successive sample over time too
>>var(z(1:100))
                  %we are also interested in what the moving %averager does
                  to the variance of the input of %course it might matter
                  when we look.
>>var(z(52:100))
>>var(ybar(2,101))
>>var(ybar(52:101))
>>var(ybar(2:21))
>>figure
>>plot(ybar)
                  %here we connect the plotted points; Connecting
                  %the z points would be a mess. Try it. They %fluctuate
                  wildly across time
```

Comparing the `ybar` plot to the `z` plot yields three observations (see Fig. 3.1). First, the variability of `ybar` is less than the variance of `z`, but their means are quite similar. Second, the largest dispersions (or swings) of `ybar` are less than those of `z`. Finally, the first few data points of `ybar` are not representative of the average value or perhaps even ultimate variance.

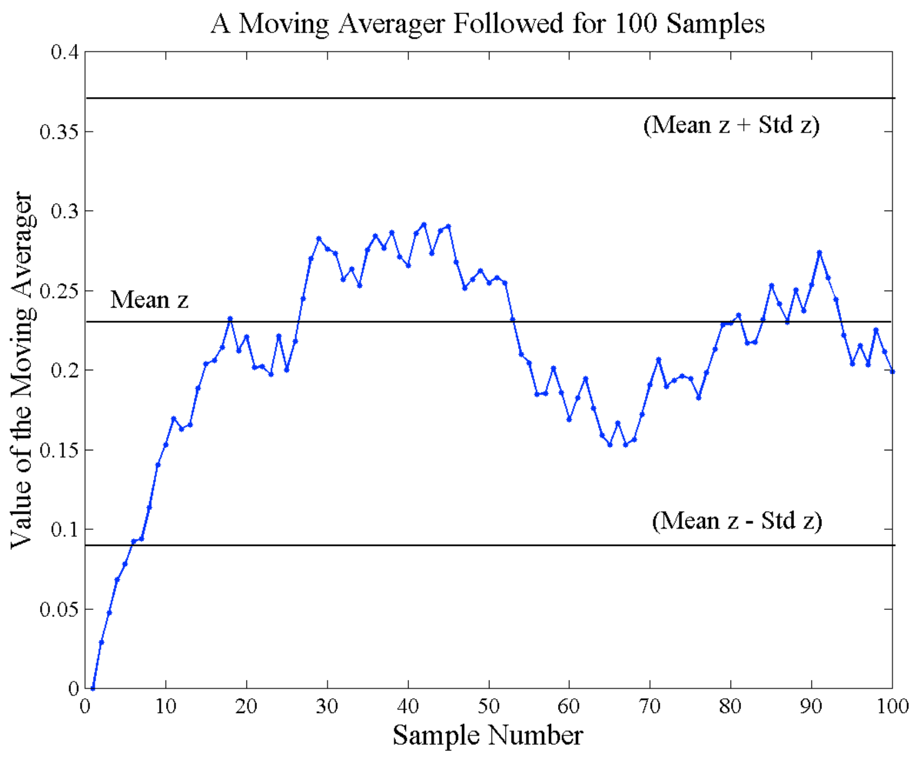


Fig. 3.1. The output of the moving averager over time. The initialized value of the moving averager is zero, but it gradually, though noisily, approaches the mean of the input set. Once reaching the vicinity of the mean, the output of the moving averager oscillates about it. The middle solid line (mean z) represents the mean of the input set; the outer solid lines represent the mean of the input set plus or minus the standard deviation of the input set.

(To produce a plot like ours you need to plot four lines on one plot.)

Homework 3.2.

1. *An exact averager vs. a moving averager.* Plot a moving averager of 1000 randomly generated points using a rate constant of 0.08. Calculate the true average with each successive sample. Plot both the true average and the moving average as a function of successive inputs on the same graph. Comment on their similarities and differences. Suppose that just by chance, both a moving averager and a true averager are at the same value after the 449th sample. Under what circumstance (input value) will they have the same value after the next sample?
2. *Uniform convergence to a point for a constant input.* Given a sequence of identical, unvarying input samples, the input values of a moving averager converge uniformly, always moving toward the value of the input, never moving away. Create a 100 dimension vector of all ones (*Hint:* MatLab has a command called `ones`). Use this set as an input to three moving averagers: with $\varepsilon=0.5$, with $\varepsilon=0.05$, and with $\varepsilon=0.005$. Describe the effect of the value of the rate constant on the convergence speed of the moving averager's value.
3. *Parameter effects.* Try out data sets with different variances. Cases of random input value, postulate a relationship between the rate constant, the number of samples, and how close the averager's value is to the random number generator's theoretical mean value. Be as quantitative as possible. If you have no ideas, do some simulations that vary the parameters of interest, and then describe what you observe. In any case, you will want to refer to some of your plots to illustrate your conclusions or conjectures.
4. *Moving averager as a function of sample number.* For some specific random number generator which you are free to specify (as long as it is not one that we have used and not uniform on [0,1)), plot the **change** of a moving average as a function of sample number. Replot this **change** as an absolute value (nonnegative). Compare and contrast the average absolute change for the first 10 samples vs. the last 10 samples. What happens to this mean for the last 10 samples as the rate constant is decreased and the sample size is increased? Use your plot to illustrate your argument.

Section 3.3. Exponential convergence

Let's try to quantify one effect of the rate constant.

Homework 3.3. A

```
$MAve(1) = 0$
for $t = 2{:}1000$,
        $MAve(t) = .99 \cdot MAve(t-1) + .01;$
end
```

Explain how this above computer code reproduces a moving averager with ε: 0.01 and 999 inputs values of one. Now create plots of *MAve* and then plot –log(1-*MAve*).

Explain why this example illustrates exponential convergence according to the equation $MAve(t) = 1 - Exp(-t \cdot 0.01)$

Note: You might be wondering what 'loglinear' means: the generic linear equation is usually written as

$Y = mx+b$

This equation implies a straight relationship between x and y with slope m and intercept b.

When Y is an exponential function of X, we can write

$$LogY = mx + b$$

which plots as a straight line when a log scale is used.

Homework 3.3.B

Exercise showing the averaging principle and that the average converges to zero.

Use Bernoulli processes for the next two problems. A Bernoulli process is a random process that takes on two values, usually 0 and 1. A Bernoulli process has one parameter *p*, the probability of generating a 1.

1. Demonstrate via simulation that the average value of the moving average goes to *p* if the simulation continues long enough. Explain your demonstration.
2. In contrast to the average value of the moving average, the squared difference between the moving average and the average of the generating process does not go to zero. Using simulations, find the relationship between the rate constant ε and the root mean squared deviation from the central value of the generating process. How does the mean to the generating process affect your proposed relationship? It might be useful to remember the variance of a Bernoulli process is $p(1-p)$.

Section 3.4. Variance of a moving averager

From the plots you have already generated, it is obvious that the moving averager has a nonzero variance even when it is in the vicinity of its ultimate average value. The variance after many samples is, in part, a function of the implicit variance of the generating process and, in part, a function of ε. Your goal is to discover something about the relationship between input and output variance. To do this, you need some examples. You will need input sequences with different underlying variances but, to simplify things, with the same mean. You will also need to try out different rate constants for the different input sets. There is no need to plot the individual moving averagers over time, but you should want to plot the variance of the moving average as a function of input variance and, separately, as a function of ε. One more thing—here's a trick that will help speed things up: instead of initializing the moving averagers at zero, initialize them to the mean of your input generator.

Section 3.5. Binary inputs and continuous outputs

When working with a binary variable, such as an action potential (spike) or the category membership decisions we studied last week, the average *x* value of the binary {0,1} variable is also approximately its probability. If a ballplayer has 321 hits after a total of 1000 times at bat, her hitting average is 0.321, and this might be the manager's estimate of her future hitting performance.

A moving averager can create a similar probability, but as always, ε is a consideration.

The size of ε affects the precision available to a moving averager. A good example occurs when we, or a postsynaptic neuron, try to use a moving averager to average spike trains. A spike train is a sequence of 0's and 1's when time is sliced into equally spaced sequential bins and in each bin is either a zero (for no spike) or a one (indicating a spike). We will let MatLab be our spike source—e.g.,

```
>>spiketrain = round(rand(1,10000)-0.355);
               %here's a spike train that fires about 14.5
               %percent of the time. By subtracting 0.355 and
               %rounding, only zeros and ones are generated as
               if %by a Bernoulli process

>>mean(spiketrain)
               %check to see how many times it really did fire
```

Homework 3.4.

1. Use the spike train data set and consider a moving averager with ε = 0.5. How does ε affect the precision of the estimate? Suppose the value of the moving averager is changed to 0.3. What values can the averager move to with one more sample (a sample is either zero or one)? What values can it move to after two samples? How close can the moving averager with an epsilon of 0.3 get (however briefly) to the mean of the Bernoulli generator? Find the average Euclidean distance the averager spends away from the mean. That is, calculate $\frac{1}{N}\sqrt{\sum_{t}^{N}(MAve(t)-\mu)^2}$, with μ being the Bernoulli generator's probability of a one and *n* equal to the number of samples.

Section 3.6. Relationship of a moving averager to synaptic memory storage

A moving average process is at the heart of synaptic modification. Synapses are used to store memories, and thus it is only natural to use this aspect of the averaging process, i.e., its memory of the past, for synaptic modification.

The statistical meaning of what synapses encode is more complicated than simple moving averagers. You'll learn more about this in Laboratory 4. Some synapses learn conditional averages. Even more sophisticated statistical functions are also encoded, as you'll discover in later labs. However, just like simple moving averagers, both the parameter-governing rate of change (ε) and the size of your samples will affect synaptic convergence. In future labs, setting ε will be a critical factor in determining synaptic modification in your neural networks. Setting rate constants is an issue that pervades all neural network research.

Section 3.7. Nonstationary environments

In Laboratory 1, you manipulated the statistics of a process that generated random numbers by changing the range of the allowed output values. In particular, you should have found that expanding the range by multiplying changed the mean and variance of the set of random numbers that MatLab produced for you. This example provides one instance that allows us to describe a stationary environment versus a nonstationary one. For the case that the range is left constant, we say that the random number generator is *stationary* in its underlying statistics. That is, when the random number generator is stationary, the statistic of descriptions of the generating process is independent of the number of samples generated. Of course, the statistics of the set of generated samples are always changing even when the generator is stationary. In contrast to a stationary generator, there is a *nonstationary* one. In this latter case, the implied statistical characterization of the generator changes over time. For instance, we could randomly, over time, change the range of the random number generator. The time between changes could be random and/or the range itself could be random.

The following exercise illustrates a nonstationary generator. Specifically, the mean of the input generator shifts halfway through the sequence of samples that comprise the input set. As you shall see, this shift presents a challenge for any type of averager.

Homework 3.5.

1. Generate two sets of random numbers with different means and concatenate them (turn them into a one tall column vector). Compare the statistics (mean and variance of `z1`, `z2`, and `zboth`). What do you observe? How does a moving averager react to a system with two local means?

```
>>z1 = 0.5*rand (1000,1);
          %mean of 0.25
>>z2 = 0.5*rand (1000,1) + 0.25;
          %mean of 0.5
>>zboth = [z1',z2']';
          %concatenates two column vectors into one
          %tall column vector can also concatenate
          %without the %transpose operation by using
          %a <enter> between the two vectors and by
          %using no punctuation, %e.g.,
          zboth=[z1,z2]
```

2. Using these same vectors, execute at least three moving averagers, try ε=0.1, ε=0.01, and ε=0.001, and also run a true averager. Use these results to discuss moving averagers versus true averagers; discuss the role of ε in a moving averager. When is a large ε good, and when is it bad? When is a small ε good, and when is it bad? Explain what good and bad mean.

Homework 3.6. A.

Consider the following topics in your laboratory write-up:

1. What complexity might you add to the moving averager to retain the good aspects of large and small values of epsilon while avoiding their bad aspects? Consider a moving averager with a dynamic rate constant. When would a dynamic ε be advantageous? How would you want the dynamic ε to work? (A well-conceived dynamic ε changes as a function of how well it is doing)In your discussion, consider what effect a dynamic ε would have on the ability of the moving averager to learn the underlying statistics of a set of numbers?
2. Describe what measurements you might use to modulate the size of ε. When should ε increase? When should it decrease? The questions in homework 3.6.B may help your thinking.

Homework 3.6.B. (For the aggressive student)

1. Suppose more than approximate convergence in mean is desired, and suppose that the input is stationary in its mean. How would you modulate the averaging rate constant as a function of the number of samples? (*Hint*: look at the exact averaging algorithm. Also consider looking at the amount of each change of the averager).
2. Suppose the environment is nonstationary but jumps unexpectedly with a known, but low probability. Describe how you would adjust the rate constant and modulate it as a function of time and various measurable statistics. Does the problem get significantly harder if the probability of change is unknown? What if this probability is not constant? Will your suggested implementation(s) work? Suggest more than one algorithm and speculate on its problems.

Section 3.8. Averaging averagers to assess performance

Let's extend our example of exponential convergence. Here are a couple of facts that might be obvious to some, but that will help you discuss the next set of computations.

The average of an average is approximately equivalent to one of the original averages. That is,

$$Ave\left[Ave\left[x(i) \mid N\right] \mid M\right] \approx Ave\left[x\,(i) \mid N\right]$$

where we use M to indicate the outer average is averaging M averages.

Here is some MatLab code to illustrate what we mean:

```
>>numbers400by20 = (1 + rand(400,20)).^2;
               %create a 400 by 20 matrix of random numbers that
               %are the square of the rand generator plus one
>>twentyaves = mean(numbers400by20)
               %average the values of each column
>>avesofaves = mean(twentyaves)
               %calculate the average of these averages,
>>result = (twentyaves-avesofaves)/avesofaves
               %show percent inaccuracy for each of the
               %approximations
```

Note that any one value of the twenty original averages is a pretty good approximation of each original average.

One way of thinking about this exercise: A variable such as x(i), with no specific sample values attached to it, is called a random variable. Curiously (or perhaps confusingly), something called a "random variable" is thought of as the random generating function itself.

As function or a variable, a random variable can be difficult to describe. One way of characterizing a random variable is through its statistics. For example, in the case of the command `rand()` we might describe the average value produced by the command after running it forever. The mean of the generated samples would go to an exact, unambiguous answer (the population mean).

But what if you are unwilling to wait forever (or even a long, long time), then, just as we saw in Laboratory 1, rather than a population mean, you would get a sample mean. And if you repeated the smaller sampling again with a different seed, you would get a sample average again but a slightly different value. Thus you should now see that even as x(i) is a random variable, so too is the sample average. And then, just as we use the sample mean of x(i) to say something reasonably useful and accurate about what x(i) generates, so too do we feel the need to take the mean of the sample's means. Importantly for our interests, this same idea applies to a moving averager; we can use averages of several moving averagers with the same rate constant to describe their performance under similar circumstances. Now let's return to the idea of exponential convergence by the moving averager.

This time you are to observe and discuss the average convergence. Pick two different values for your rate constants that you will compare, and pick them in such a way that will appropriately illustrate your discussion.

Homework 3.7.

1. Create a matrix of at least 30 by 400 random numbers. For each of the 30 rows, initialize a moving averager to zero; run the average for 400 timesteps; and store the sequential values of each step for each of these averagers. Now average across the thirty averages for each of the 400 time points (you should get 400 values). You now have a single curve that shows average convergence. Plot this curve time point by time point. Plot the differences from the presumed asymptotic value. Plot these differences on a log scale.
   Use these 30 averages as input to the thirty-first averager. That is, you are obtaining the average of the averagers. In what sense(s) can you argue that the average of the averagers does a better job than any one averager? To be quantitative compare the standard deviations around the asymptotic value for the last 100 timesteps. How do these simulations extend, or perhaps help illustrate, your earlier discussions?

Section 3.9. Conclusion and summary

In this laboratory, you've learned about averages and, in particular, about a moving averager. We also pointed out some facts about averages (means) that you may have overlooked. The average of a set of numbers is an abstraction which possesses equal contributions from each number in the set but which is not necessarily an instance in that set. (In more advanced mathematics, a concept of an average over the entire population is called an expectation.) You can think of an average as an expectation, more precisely an expectation when given the history of an infinity of samples. Whether or not the moving average algorithms can converge to an expectation (or to the vicinity of an expectation) that is also a statistic of the random generator is a question that eventually leads to some very heady mathematics falling under the rubrics of stochastic averaging and as well advanced statistical theory and even Kalman filtering. Fortunately, we can understand something about such processes without getting into these advanced mathematical areas.

A moving average depends on three factors:

1. a number that is an abstraction of all past samples;
2. the current sample;
3. and a rate constant, ε.

A moving average is computationally efficient because it does not require a working knowledge of all past elements—instead, it uses a condensed version of the past. Accuracy is traded for a rapid response to changes in the general trend of sequentially received data. Thus a moving average can have an important advantage over the exact average—namely, a quick convergence to the vicinity of a true average when input statistics shift.

Section 3.10. Possible quiz questions

If there is an in-class quiz next week, you should be able to answer the following questions:

1. Suppose that an input X is bounded as [0,1) and suppose that we initialize the averager at one value strictly between zero and one. What are the largest and the smallest values that this averager can possibly obtain with a finite number of samples? Explain your answer. Assume that the rate constant of the averager lies strictly between zero and one, i.e. $\varepsilon \in (0,1)$. (*Hint*: for your explanation use the same value as inputs to the averager forever).
2. Suppose the current value of the moving averager is zero. As a function of ε, what is the largest value it can take on when it incorporates one new input? What about a true averager? Be sure to mention the parameters your answer depends upon.
3. Suppose the moving averager is currently valued at 0.99. What is the largest value that it can take on when it incorporates the next input? As above, be sure to describe the parametric dependencies of your answer.

Glossary Terms

[0,1)	the real numbers from zero to one including zero but not including one.
average	something that is not extreme, but instead something in the middle; for our purposes, an *average* is the arithmetic mean
convergence	a process in which a sequence of numbers goes to some describable location—e.g., convergence to a point such as: 0.9 + 0.09 + 0.009 + … converges to one; convergence to a point or to a region around a point.
moving average	the sequential updating of an average
nonstationary process	the opposite of a stationary process (see also stationary process) MatLab example:

```
x = zeros (10,1)
     for J = 1:10
        x(J,1) = J·rand(1,1)
             %note that J is always end
             %rescaling rand() and
             %the net process implies a
             %changing variance
```

random	The term *random* can be used in many ways:

- Not deterministically describable or exactly knowable
- Apparently not deterministically describable (even though, in fact, it is deterministically describable); often pseudorandom is the more precise term.
- Maximally unpredictable or uncertain
- Stochastic
- There are others uses of this term, so be careful—if you use this word, be sure your reader knows what kind of random you mean.

random sample	A single output generated by a stochastic process is a random sample of the population (perhaps infinite) implied by the definition of the process.
rate constant	called epsilon (ε), a crucial value that has a powerful effect on a moving averager and its memory

stationary process	a stochastic process governed by a set of statistics that do not change over time. MatLab example:

```
x = zeros (10,1)
     %initialize x
  for J = 1:10
     %generate 10 pseudorandom
    x(J,1) = rand(1,1)
     %numbers with uniform
   end
     %probability on [0,1) and store as a
     %single column vector
```

stochastic process	A particular collection of random variables. MatLab example:

The MatLab command `rand()` calls a computational implementation of a process that selects a number from 0 to 1, but not including 1, (supposedly) with uniform probability. Such a number can only be generated with finite accuracy. Thus, although we speak of the generator on the interval [0,1), in actuality the generator can only produce values from a finite set.

Appendix 3.1. A mathematical explanation of a moving averager

For the random variable $x(t)$, the update rule is defined as:

$$m(t+1) = m(t) + \Delta m(t)$$

$$= m(t) + \varepsilon(x(t)-m(t))$$

$$= \varepsilon x(t) + (1-\varepsilon)\, m(t)$$

where ε is a constant and $0 < \varepsilon < 1$.

Thus, each new value of the moving average is calculated by adding a calculated value to the previous moving average. The calculated value – which is the change between one moving average and the next, is computed as a function of the difference between the new value to be added in to the average and the previous value of the moving average. Knowing this, we can calculate the change in the moving averager in terms of *t*. This will be a difference equation, which is defined as the relationship between consecutive elements in a sequence. Essentially, it is a differential equation, but the changes in *t* are not so small as to become inconsequential. Let's compute the difference equation for this moving averager.

We have already established:

$m(i+1) = m(i) + \varepsilon(x(i)-m(i))$ where $m(i)$ is the averager, $x(i)$ is the added value,

and ε is a constant and $0 < \varepsilon < 1$.

$$m(i+1) - m(i) = m(i) + \varepsilon(x(i)-m(i)) - m(i)$$

$$\Delta m = \varepsilon(x(i)-m(i))$$

$\Delta m/\Delta t = \varepsilon(x(t)-m(t))$ to indicate the dependence on *t*.

This is the difference equation for a moving average. The function that we used in MatLab to compute successive values of `ybar` was computing them by finding the change in `ybar` as shown by this equation.

But for this equation it is equivalent to write

$$m(t) = \varepsilon x(t-1) + (1-\varepsilon)\, m(t-1)$$

Substituting gives

$$m(t+1) = \varepsilon x(t) + (1-\varepsilon)\, (\varepsilon x(t-1) + (1-\varepsilon)\, m(t-1))$$

$$= \varepsilon x(t) + \varepsilon(1\text{-}\varepsilon)\, x(t\text{-}1) + (1\text{-}\varepsilon)^2\, m(t\text{-}1)$$

and we can continue this process to find that

$$m(t+1) = \varepsilon \sum_{k=0}^{\infty} (1-\varepsilon)^k\, x(t-k)$$

where we suppose this process continues for an infinite length of time.

Now *m*(*t*) is a function of a random variable *x*(*t*), and we assume *x*(*t*) is stationary. Therefore,

$$E[m(t)] = \varepsilon\, E[x(t)] \sum_{k=0}^{\infty} (1-\varepsilon)^k$$

We recall that, for $0 < t < 1$,

$$\frac{1}{1-t} = 1 + t^2 + t^3 + \cdots$$

and with $t = 1-\varepsilon$

$$\sum_{k=0}^{\infty} (1-\varepsilon)^k = \frac{1}{1-(1-\varepsilon)} = \frac{1}{\varepsilon}.$$

Therefore,

$$E\left[m(t)\right] = \varepsilon \quad E\left[x(t)\right]\frac{1}{\varepsilon} = E\left[x(t)\right]$$

(Note: we can do the summation if the process has not been going on forever using

$$\frac{t^p}{1-t} = 1 + t^2 + t^3 + \cdots + t^{p-1}$$

and supposing that m(0) = 0, or for any starting value.)

Now we will calculate the second moment for a stationary variable x (that is, E[x(t)] is the same for all t):

$$E\left[m(t)^2\right] = E\left[\left(\varepsilon \sum_{k=0}^{\infty} (1-\varepsilon)^k\, x(t-k)\right)^2\right] = \varepsilon^2 E\left[\left(\sum_{k=0}^{\infty} (1-\varepsilon)^k\, x(t-k)\right)^2\right]$$

$$=\varepsilon^2 E\Big[\big(x(t)+(1-\varepsilon)x(t-1)+(1-\varepsilon)^2 x(t-2)+\mathrm{L}\ \big)\big(x(t)+(1-\varepsilon)x(t-1)+(1-\varepsilon)^2 x(t-2)+\mathrm{L}\ \big)\Big]$$

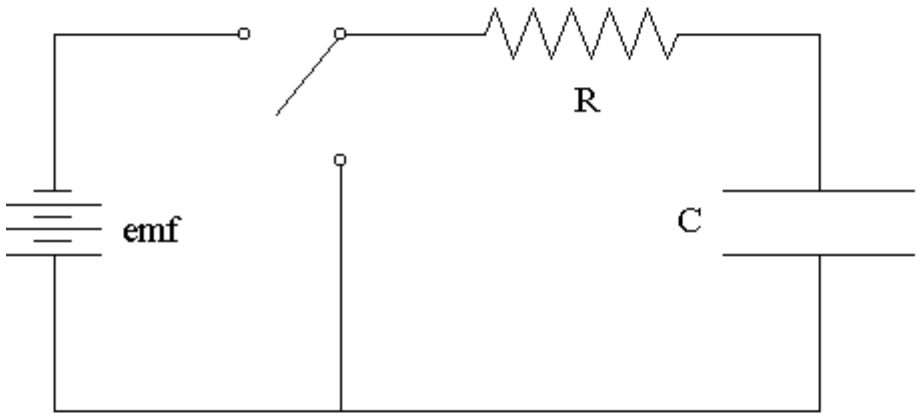
$$=\varepsilon^2 E[x(t)^2]\cdot(1+(1-\varepsilon)+(1-\varepsilon)^2+\mathrm{L}\ )\cdot(1+(1-\varepsilon)+(1-\varepsilon)^2+\mathrm{L}\ )$$

where we have used the independence and stationarity of the $x(t)$ terms to factor the variate out of each parenthetical expression. From the last line we again recognize $\sum_{k=0}^{\infty}(1-\varepsilon)^k=\frac{1}{1-(1-\varepsilon)}=\frac{1}{\varepsilon}$, implying the last line equals

$$=\varepsilon^2 E[x(t)^2]\cdot 1/\varepsilon^2=E[x(t)^2].$$

Appendix 3.2: The continuous exponential averager

The discrete moving averager is required for simulations of processes with memory. Many students will already be familiar with the continuous version, here represented by an RC circuit with a switch the moves between charging and discharging the voltage on the capacitor.



Here we solve for the time-dependent voltage across the capacitor, $V_C(t)$, with a battery of constant voltage V_B switched on at $t=0$ with initial condition $V_C(t=0)=0$.
The voltage dropped by the circuit components equals the battery voltage:

$$V_B=V_R(t)+V_C(t) \qquad (1)$$

$$=I(t)\bullet R+\frac{1}{C}\int_0^t I(s)ds \qquad (2)$$

Taking the derivative of (1) and then substituting from (2)

$$\frac{dV_B}{dt} = \frac{dV_R(t)}{dt} + \frac{dV_C(t)}{dt}$$

$$0 = \frac{dV_R(t)}{dt} + \frac{dV_C(t)}{dt} == R\frac{dI(t)}{dt} + \frac{1}{C}I(t), \qquad (3)$$

or equivalently,

$$\frac{dI(t)}{I(t)} = -\frac{1}{RC}dt .$$

Integrating both sides yields $\log_e I(t) = -t / RC + const$ or

$$I(t) = const \cdot Exp(-t / RC) . \qquad (4)$$

Note in this form t / RC is unitless (RC is the well-known time constant of the circuit) while the constant outside the exponent must be in units of current. Now we must for this constant. Multiply both sides by *R*; use (1), and the initial condition. That is, at time zero all the voltage of the battery is dropped across the resistor

$V_B = V_R(0) = I(0)R$ or $I(0) = V_B / R$. Therefore, substituting into (4) with *t=0*, this is the value of the constant. Therefore, using this result and (4) we can write the time-dependent voltage across the resistor.

$I(t)R = V_R(t) = V_B Exp(-t / RC)$. To find the time-dependent voltage across the capacitor use this result and (1)

$V_B = V_B Exp(-t / RC) + V_C(t)$, which re-arranges to

$V_C(t) = V_B - V_B Exp(-t / RC) = V_B(1 - Exp(-t / RC))$.