Chapter 1. Average filter

1.1 Recursive expression for average

As you know, average is sum of a given data set divided by the number of data in it. For example, if the given data set contains k datum($x_1, x_2, \cdots, x_k$) the average is obtained by the following expression.

$$\overline{x}_k = \frac{x_1 + x_2 + \cdots + x_k}{k} \tag{1.1}$$

The expression above treats whole data in a single batch and thus called 'batch expression.' What would happen if a datum is added to the current data set? If we follow the definition of average, all the datum should be summed again and divided by $k+1$. In this process, the average we calculated previously($\overline{x}_k$) could not be used since Equation (1.1) is not in recursive form, which reuses the previous result.

Recursive expression reuses previous result and hence it has good efficiency in computation. For a case such as the number of datum in the given data set is one million or ten million, this becomes more apparent. Also, recursive expression has advantage in the aspect of memory storage. If Equation (1.1) is used to get average, the average value from previous computation, the new datum just fed in, and updated total number of datum are required and this makes computation efficient.

Actually, Equation (1.1) could be written in recursive expression. The objective is to make $\overline{x}_{k-1}$ to appear on right hand side of Equation (1.1). First, we begin with the expression for the average of $k-1$ datum since this will be used during the derivation.

$$\overline{x}_{k-1} = \frac{x_1 + x_2 + \cdots + x_{k-1}}{k-1} \tag{1.2}$$

Multiply k to both sides of Equation (1.1).

$$k x_k = x_1 + x_2 + \cdots + x_k$$

Dividing both sides by $k-1$, we get

$$\frac{k}{k-1}\overline{x}_k = \frac{x_1 + x_2 + \cdots + x_k}{k-1}$$

The expression on the right hand side could be expressed in two terms after separating x_k from the original term.

$$\begin{aligned}\frac{k}{k-1}\overline{x}_k &= \frac{x_1 + x_2 + \cdots + x_k}{k-1} \\ &= \frac{x_1 + x_2 + \cdots + x_{k-1}}{k-1} + \frac{x_k}{k-1}\end{aligned}$$

The first term on the right hand side is identical to the definition of $\overline{x}_{k-1}$, as described in Equation (1.2) so the equation above could be rewritten as

$$\frac{k}{k-1}\overline{x}_k = \overline{x}_{k-1} + \frac{x_k}{k-1}$$

It is almost done now. Dividing both sides by $\frac{k}{k-1}$, we get the following recursive expression.

$$\overline{x}_k = \frac{k-1}{k}\overline{x}_{k-1} + \frac{1}{k}x_k \tag{1.3}$$

At last we derived a recursive expression which defines $\overline{x}_k$ with $\overline{x}_{k-1}$. If we use Equation (1.3) to get average, only the average computed in previous step($\overline{x}_{k-1}$), total number of datum in the data set(k), and freshly added datum(x_k) are all we need. Here, we do not need whole datum in the data set. This is why a recursive expression is useful for computing average of a data set with a datum being added sequentially.

Now let us see if Equation (1.3) really returns average. Suppose the following three data are being fed in sequentially (the average of this data set is 20).

$$x = 10,\ 20,\ 30$$

Average of the first datum is like expressed as following since the average of a single datum is the datum itself.

$$\overline{x}_1 = x_1 = 10$$

For the rest of the data, we apply the recursive expression sequentially.

$$\overline{x}_2 = \frac{1}{2}\overline{x}_1 + \frac{1}{2}x_2 = \frac{1}{2}10 + \frac{1}{2}20 = 15$$

$$\overline{x}_3 = \frac{2}{3}\overline{x}_2 + \frac{1}{3}x_3 = \frac{2}{3}15 + \frac{1}{3}30 = 20$$

The final result is 20, which matches with the average of the whole data set given.

Equation (1.3) could be simplified even further. If we define $\alpha \equiv \frac{k-1}{k}$, following relationship between α and k stands.

$$\alpha \equiv \frac{k-1}{k} = 1 - \frac{1}{k}$$

$$\therefore \frac{1}{k} = 1 - \alpha$$

Substituting the equation above into Equation (1.3), we get a simplified form of the recursive expression.

$$\begin{aligned} \overline{x}_k &= \frac{k-1}{k}\overline{x}_{k-1} + \frac{1}{k}x_k \\ &= \alpha\overline{x}_{k-1} + (1-\alpha)x_k \end{aligned} \tag{1.4}$$

The name of Equation (1.4) is ‘average filter.’ We will see similar expressions on our way throughout this book so please pay close attention to the equation.

Average filter is used for sensor initialization as well as computation of average. A digital weight scale could be a good example. For various reasons, zero-point of the digital weight scale changes continuously. Therefore, an initialization process which defines zero-point after gathering measurements for certain period of time after turning the power on is required. If all the measurement has to be stored to get the average, an expensive microprocessor will be needed, which is not a good choice, obviously. This could be avoided by employing an average filter.

1.2 Average filter function

`AvgFilter` function laid out in the following is a code of the average filter described in Equation (1.4). This function receives measurement as argument and returns its average.

List: AvgFilter.m

```
function avg = AvgFilter(x)
%
%
persistent prevAvg k
persistent firstRun


if isempty(firstRun)
  k = 1;
  prevAvg = 0;

  firstRun = 1;
end


alpha = (k - 1) / k;
avg   = alpha*prevAvg + (1 - alpha)*x;

prevAvg = avg;
k       = k + 1;
```

The length of the code is fairly short since the expression itself is very simple. Rather, the part not directly relevant to the algorithm takes most of the code. To use Equation (1.4), previous average and the number of datum should be retained even after executing the `AvgFilter` function since these will be used in the next step of computation. That is why the two variables, `prevAvg` and `k`, are declared as `persistent`. In MATLAB®, a `persistent` variable maintains its value even after executing a function. It is somewhat similar to static variables in C/C++.

For those not familiar with MATLAB® programming, a brief explanation of initialization part of the `AvgFilter` function is provided here. The codes being presented further in this book will also be in similar form. The following is an excerpt of the part related to initialization from the original code. The initialization of the `persistent` variable in `if` statement is executed only once when the `AvgFilter` function is being executed at the first time. To determine whether this is the very first execution or not, an intrinsic function named `isempty` is executed to see if the variable `firstRun` is empty because a `persistent` variable is empty when a function is executed for the very first time.

```
persistent firstRun


if isempty(firstRun)
  k = 1;
  prevAvg = 0;

  firstRun = 1;
end
```

1.3 Example: Voltage measurement

In this section, we will see if the `AvgFilter` function works well through a simple example.

Ms. "Ave High" is engaged in research of a battery for an electric car. She measured the voltage of a new battery that just came in, but there was too much noise which made the voltage to appear different every time. So she decided to gather the measurement data for certain period of time and get the average. The voltage was measured every 0.2 seconds.

Following is a test program to verify the `AvgFilter` function. In `TestAvgFilter.m` file, the part executing `AvgFilter` function is the only part related to average filter algorithm. All the rest are the code for running simulation and storing the result.

`GetVolt` function reads the voltage value. This function is stored in `GetVolt.m` file. As it is customary in MATLAB, the name of the file containing a function is maintained same as that of the function itself. It has been assumed that the voltage measured by `GetVolt` function has an average value of 14.4[V] with some noise, which has the average of 0[V] and standard deviation of 4[V]. The noise was generated by `randn` function, which is intrinsic in MATLAB.

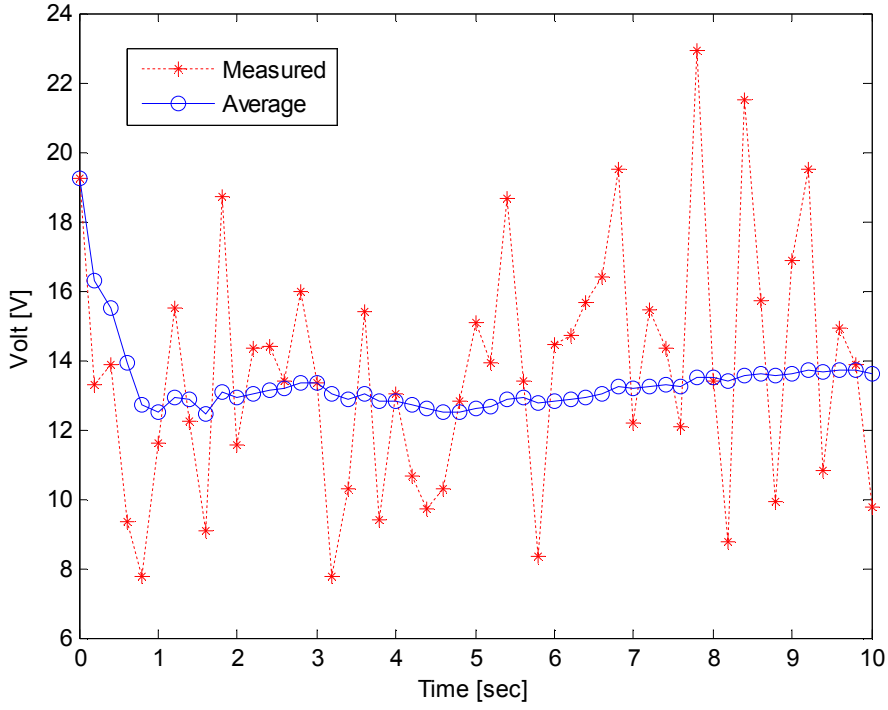
List: TestAvgFilter.m

```
clear all

dt = 0.2;
t  = 0:dt:10;

Nsamples = length(t);

Avgsaved = zeros(Nsamples, 1);
Xmsaved  = zeros(Nsamples, 1);

for k=1:Nsamples
  xm  = GetVolt();
  avg = AvgFilter(xm);

  Avgsaved(k) = avg;
  Xmsaved(k)  = xm;
end


figure
plot(t, Xmsaved, 'r:*')
hold on
plot(t, Avgsaved, 'o-')
```

List: GetVolt.m

```
function z = GetVolt()
%
%
w = 0 + 4*randn(1,1);
z = 14.4 + w;
```

The following is a plot showing measured vs. averaged voltage as a result of the average filter. The measurement shows violent variation but the output from the average filter shows stable pattern. As data accumulates, measurement noise is subsides and the output value approaches the average value of the voltage. As we could see from this example, when a physical quantity measured is averaged, noise is filtered. This is very interesting characteristics, which gave the name 'filter' to Equation (1.4).



For your convenience, all the codes in the book are separated into algorithm and test files like this example. If the two are mixed into one, it could hinder the analysis of algorithm, which is the core part. Name of the main file of the any program has 'Test' at the beginning. The algorithm and sensor measurement files are stored as their own names following MATLAB custom of naming files. For instance, `GetVolt` function is stored in the file `GetVolt.m`. For a function reading or generating measurements, 'Get' is put in front of the name of the physical quantity of interest.

Algorithm		Ex.> `AvgFilter.m`
Test Program	Main file	Ex.> `TestAvgFilter.m`
	Sensor measurement	Ex.> `GetVolt.m`

1.4 Summary

From an average filter, which is a recursive expression, an average could be obtained readily with only the average value from previous computation and total number of datum updated. There is no need for storing the whole datum in the given data set. The filter has good efficiency especially when each datum is being fed sequentially. If data is to be processed in real time, a filter of a recursive form is necessary. Exaggerating a little bit, it could be said that 'a filter which is not in recursive form is virtually useless in real world.' At the end of this chapter, it has been confirmed that averaging process also filters noise.