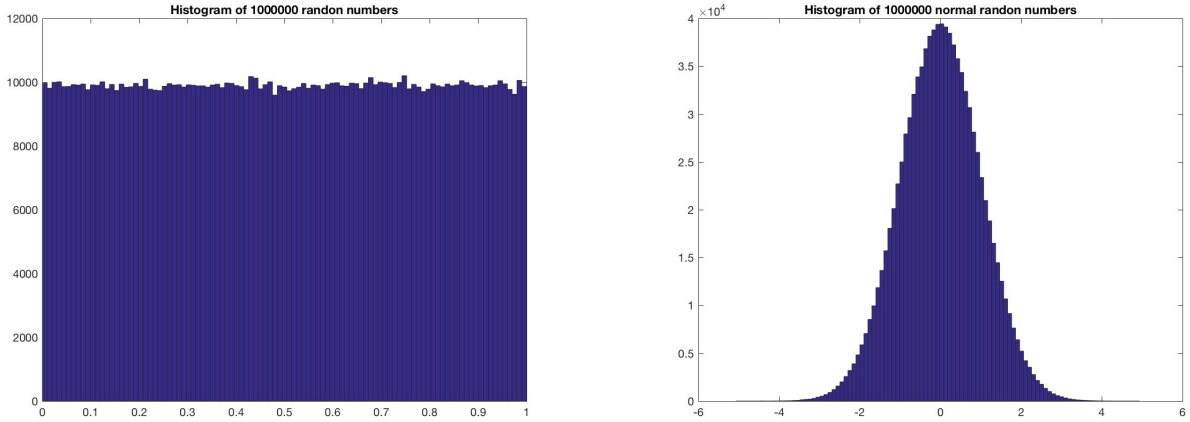
CS2035 - Assignment 1 - 2017

Statistical Function Programming Out: January 9th, 2017

In: Sunday, February 5th, 2017, at 11:55pm via Owl

Introduction

You are to write 4 MatLab functions for this assignment. All your functions should be coded in a single file called **ass1_2017.m**. The first function in this file is called **ass1_2017** with a single input parameter **num_x**. There are no output parameters. **num_x** specifies the number of random numbers to be computed and stored in variables **x** (uniformly distributed random numbers) or **y** (for normally distributed random numbers). Note that parameter **num_x** is originally a character string and so has to be converted to an integer (**num_x=str2num(num_x)**) before any calculations can be done with it. For example, **str2num('123456')** return the number **123456** as its output. You call your program from the command line, for example **ass1_2017 1000000** will set **num_x** to 1000000. bf x and **y** can be computed as **x=rand(num_x,1,'double')** and **y=randn(num_x,1,'double')** respectively. One possible set of histograms you might obtain are shown in Figure 1 below.



(a) (b)

Figure 1: (a) Uniform and (b) normal random number distributions.

Your program should perform the following tasks:

1. The **ass1_2017** function first plots titled histograms of **x=rand(num_x,n)** and **y=randn(num_x,n)** using the MatLab built-in **hist** function. The title of the histograms should indicate the value of **num_x**. To generate the title you have to use string concatenation to construct the appropriate string for the title, for example, one title string, **stg**, might be **stg=['num_x=' num2str(num_x)]'** (but you can do better than this). After the histogram is plotted, use the MatLab function **title(stg)** to title your plot with your **stg**.

2. Next, the **ass1_2017** function should compute the skewness and kurtosis values of **x** and **y** using MatLab built-in functions, **skewness** and bf kurtosis. Then, using the 2nd and 3rd functions that you write, namely **my_skewness** and **my_kurtosis**, you should again compute these values for **x** and **y**. You should get the same values. Your functions should use arrays and should not use any other MatLab functions (other than **sqrt**) in their calculations. This means, for example, that you must compute the average of **x** and **y** with your own code and not use **mean(x)** or **mean(y)**. If you need the sum of some array **x**, you need to sum the elements of **x** explicitly in a loop and not use **sum(x)**.

   From MatLab online documentation:

   > Skewness is a measure of the asymmetry of the data around the sample mean. If skewness is negative, the data are spread out more to the left of the mean than to the right. If skewness is positive, the data are spread out more to the right. The skewness of the normal distribution (or any perfectly symmetric distribution) is zero.

   Skewness can be computed as:

$$s = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{\left(\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}\right)^3}, \tag{1}$$

   where $\bar{x}$ is the average of **x**, i.e. $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$.

   From MatLab online documentation:

   > Kurtosis is a measure of how outlier-prone a distribution is. The kurtosis of the normal distribution is 3. Distributions that are more outlier-prone than the normal distribution have kurtosis greater than 3; distributions that are less outlier-prone have kurtosis less than 3.

Kurtosis can be computed as:

$$k = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^2}. \quad (2)$$

Again, $\bar{x}$ is the average of **x**. Function **my_skewness1(q)** computes the skewness of 1D input array **q** using Equation (1) while function **my_kurtosis(q)** computes the kurtosis of 1D input array **q** using Equation (2).

3. Next, you must code a 4th function **generate_distribution** with header:

```
[x,min_x1,max_x1,min_x2,max_x2]=generate_distribution(low_bound_x,high_bound_x,num_x).
```

This function computes **num_x** uniformly distributed random numbers between **low_bound_x** and **high_bound_x**. Output parameter **x** contains these numbers. **min_x1** and **max_x1** contain the minimum and maximum values returned by **rand(num_x,1,'double')**. Note that these are not necessarily 0 and 1 but numbers quite close to these values. We can scale **x** to be precisely between 0 and 1 by using the relationship **x=(x-min_x1)/(max_x1-min_x1)**. in this case **x** is rescaled by the vectorized MatLab statement just given. The minimum and maximum of scaled **x** are saved in output parameters **min_x2** and **max_x2**. These should be precisely 0 and 1. Note that the real reason in having **generate_distribution** the before and after min and max values of **x** returned is so that you have experience with a function that returns more than 1 parameter.

Lastly, you must scale **x** to be between **low_bound_x** and **high_bound_x**. Ensure that **low_bound_x** is indeed less than **high_bound_x** before any calculation is done. Print a fatal error message and quit the program if this condition (**low_bound_x** < **high_bound_x**) is not satisfied (just exit the function using the **return** statement). Use an **if-then-else** statement to code this. Multiplying **x** by **(high_bound_x-low_bound_x)** makes **x** be between 0 and **(high_bound_x-low_bound_x)**. Adding **low_bound_x** finishes the calculation as the **x** values are now between **low_bound_x** and **high_bound_x**. After **generate_distribution** has be executed, print out the values of **min_x1** and **max_x1** and **min_x2** and **max_x2** and **low_bound_x** and **high_bound_x**. Also print out the average and median of **x** using **mean** and **median**.

4. Given this scaled **x** value, use **sum(x)** to add its values. Compare this sum to the addition of the sorted values of **x** (**sum(sort(x))**). Compute the difference of the two sums. Write

a comment in your code explaining why this difference occurs (why it is not 0) and which sum you expect to be more accurate.

5. Lastly consider the number **1.0e64**. Note that **e** means exponent, that is **1.0e64** is 1.0×10^{64} in scientific notation. This is a large number. Store it in variable **b**. Determine another variable **a** that is non-zero and yet when added to **b** yields **a+b=b**. Verify this addition in your program. **a** is a **relative zero** for **b**.

All numbers printed so far should be printed using **fprintf** with format **%22.12f** for all doubles and **%d** for all integers. To print out numbers in scientific notation use format **%e**.

You must document your code (with appropriate comments). Write "beautiful" code, not just with comments but with appropriate code aligned nicely and use good, meaningful variable names. Don't use "magic constants", i.e. undocumented numbers. For example, instead of **x=rand(1000,1,'double')** use **x=rand(number_of_students,1,'double')** where **number_of_students=1000** has been set elsewhere.

See the course webpage for examples of 2 well documented MatLab function examples, **commented_new_decimal2roman.m** and **commented_new_roman2decimal.m**. Have a block comment at the beginning of **ass1_2017.m** giving (at least) your full name (as it appears on your student card), your student number, your uwo email address and which assignment you are doing. For example:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  John Barron                       %
%  123456789, barron@uwo.ca          %
%  CS2035, Assignment 1, 2017        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The source code **ass1_2017.m** should via OWL by the due date. You know how to submit your assignment because your did the first lab! We will run your code while grading your assignment. It is necessary to follow the assignment specs to optimize your grade.