**Detection Problems in Assuming Gaussian Distribution for Sum of Independent Random variables**

Introduction – In estimation and detection theory many problems have closed form solution for gaussian distribution. We also know that distribution of the sum of independent and identically distributed random variables converges towards gaussian by the central limit theorem. The distribution of the random variables doesn’t matter. It is assumed that central limit theorem doesn’t take sum of too many IID random variables to converge towards a gaussian distribution. In nature, many times we only observe the sum of IID random variables. Hence, it is assumed that detection and estimation methods that work for normal distributions can be applicable to many problems involving other types of distributions. Due to this gaussian is used everywhere in industries like engineering, economics, statistics and sciences. Central limit theorem is considered central to the statistics and some people assume that name of the theorem comes from that.

But normal distribution is a very special type of distribution where probability of random variable goes down as a function of exponential squared. This means that probability of finding a random variable away form mean value is very very low. 99.7% times random variable stays within 3 standard deviations (sigma) away from the mean and probabilty of 6sigma or 7sigma events is almost zero.

In this project, we will examine consequence of making the gaussian assumption in detection errors especially for rare events. We will look at the behaviour of the following the random variable

According to the central limit theorem this random variable should converge towards the normal distribution with mean 0 and variance 1.

**How to generate an random number(Pseudo) and Independent Variable!!**

As our arguments on central limit theorem are based empirical results, its good to understand how scientific software generates so random variables and how they make sure (do they?) there independent on every iteration. Below paragraph is good to know for all those study the science of uncertainity!

Random number generators Generating (pseudo-) random numbers X ~ U(0,1) is fundamental to all experimental statistics, simulation, experiment design, and data analysis. Programming languages generally use a system-supplied random number generator, like x = rand() or x = rand(seed) where seed is an integer a starting value which will be used to "seed" the random number generator. Research on generating good random number generators was recently matured. So, if you’re using some good old software and expecting to make strong conclusions, please think again.

“*Some quotes from Numerical Recipes (3rd ed., Ch. 7): Be cautious about any source earlier than about 1995, since the field has progressed enormously in the following decade. [...] The greatest lurking danger for a user today is that many out-of-date and inferior methods remain in general use. [...] If all scientific papers whose results are in doubt because of [bad random number generators] were to disappear from library shelves, there would be a gap on each shelf about as big as your fist”.*

Numerical Recipes contains examples of good, easy to port random number generators. But the simply way to generate is to use generators that are part of a well know software packages which are based on the so-called Mersenne Twister (Matsumoto & Nishimura 1997) used by MATLAB [mt19937ar] since about 2008 is considered to be of high quality - it has passed a number of stringent test case for uncertainty, including the ‘Diehard’ test suite (Marsaglia 1998)

When MATLAB is started, and you ask for to generate four random numbers using the built in function rand(), you get>> rand(2,2) ans = 0.814723686393179 0.126986816293506

0.905791937075619 0.913375856139019 >> rand(1) ans = 0.632359246225410

If you exit MATLAB and restart again, you get for example: >> rand(2,3) ans = 0.814723686393179 0.126986816293506 0.632359246225410

0.905791937075619 0.913375856139019 0.097540404999410 After restarting MATLAB, the default random stream is initialized with the a hard coded seed(=0)

**How Python generates random variables**

To compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, python packages make use of von Mises distribution. Almost all module functions depend on the basic function [random()](https://docs.python.org/3.2/library/random.html#module-random), which generates a random float uniformly in the semi-open range [0.0, 1.0). Python uses the Mersenne Twister like Matlab as the core generator. It produces 53-bit precision floats and has a period of 2\*\*19937 (hinting that it is pseudo random after this runs in the simulation or if restart matlab). The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

The functions supplied by this module are actually bound methods of a hidden instance of the random. Random class. Python gives freedom to instantiate own instances of Random to get generators that don’t share state.

***Simulation Setup:*** A typical detection problem where the observations are distributed for a sum of independent and identically distributed random variables. Our attribute set has two parameters with equal prior probability. Monte carlo simulations are ran for 10000000 iterations, varying the numbers of random variables(k=[1,100]) in the sum and also changing the difference between the expected value of the respective conditional probability distributions(μ=[0.5,7]). The following sections share the same simulation setup as explained above.

**Irwin-Hall Distribution/Sum of I.I.D Uniform Random Variables:** The Irwin–Hall distribution is the continuous probability distribution for the sum of ***N*** independent and identically distributed Uniform random variables with mean 0.5 and variance 1/12 i.e Uniform distribution between **U[0,1]**

The probability density function (pdf) is given by

where ***sgn(x − k***) denotes the sign function:

Thus the pdf is a spline (piecewise polynomial function) of degree *n − 1* over the knots 0, 1, ..., *n*. In fact, for *x* between the knots located at *k* and *k + 1*, the pdf is equal to

The mean and variance are ***N/2*** and ***N/12***, respectively.

CDF is given by:

**Sum of Exponential Independent Random Variables:**

**Erlang Distribution:**  Sum of ***k*** independent and identically distributed exponential random variables with parameter μ are erlang distributed and probability density function is given below

From the below plot we can observe that with increase in ***k*** , tail is increasing and density and the center/mean is decreasing and after certain ***k*** the shape of distribution becomes bell shaped, so the probability of error also converges towards what we can estimate with gaussian noise.

**Sum of Correlated Gaussian Random Variables:** We generated correlated normal random variables with 2 tap filter with configurable coefficients. So, code framework can be reused by anyone who wants to study the central limit theorem empirically and gain some insights.

**Power Law distributed random variables:** A power law is the form of an important relationship taken by two quantities and is a relation of the type , where Y and X are interested quantities, ***a*** is called the power law exponent. In nature, many relationships follow power laws. Surface area to volume, the inverse-square laws of Newtonian gravity, fractals, earthquake intensity, the sizes of wars, etc. Most of the economic relationships also take the form of power laws like the distribution of income, wealth, size of cities and firms, distribution of financial variables such as returns and trading volume. Power laws are also the heavy tail distributions, meaning that frequency of rare events is much higher than that of the known events. We are particularly interested in the distribution of the sum of independent and identically distributed power laws and we presented number results in the following section and found the key issues in assuming such distribution as gaussian either knowingly or unknowingly through a detection problem. Rigorous simulations for multiple scenarios are performed to detect signals from a sum of I.I.D distributed power laws and detection errors are compared with that of the gaussian distributed to argue on our hypothesis.