Tutorial 2 Random variables, Central Limit Theorem, and Error Propagation

Overview

It is very easy today to generate random numbers in any computer language. This is the core of Monte Carlo methods, often the most efficient way to solve complex engineering problems in radiation heat transfer, error propagation in complex systems, and industrial process simulation. In this tutorial we will generate random variables to get a better feel for different distributions, error propagation, and the Central Limit Theorem.

Objectives:

This tutorial has 3 specific objectives:

a) Work with Matlab functions rand and randn, to compare different probability density functions
b) Determine when the distribution for a sum of "coin tosses" converges to the normal distribution
c) Use and compare traditional "error propagation" to Monte Carlo simulation

Pre-tutorial questions

The assigned textbook questions will not be marked, but the tutorials are intended as a place to discuss the questions that you found challenging. Also, the assigned questions are intended to exercise skills needed for the tutorials and exams.

The assigned questions for this period are:

3.24, 3.42, 4.24, 4.28, 5.21, 5.23, 5.31, 5.35[1], 5.36, 5.37

[1] A strange thing about the Taylor text is the use of "t" as the test statistic to check acceptability with the Normal Distribution. Most texts use "z" for this, and "t" when we use the "Student's t Test". The difference between Student's distribution and the Normal distributions is usually not large, but when working with small samples it can be important. This may be discussed further in the lectures, and Wikipedia can give you the basics (and more) if you are curious.

Background

Random Number Generation

Simulation of random processes requires a "Random Number Generator" (RNG). What do want in an RNG? Firstly we want each number produced to be independent of the previous number (and ideally of all previous numbers). Secondly, we would like the number to have a prescribed pdf, because typically we want to model a process with a prescribed pdf.

Most RNGs are actually *pseudo* random number generators (PRNGs) because the algorithms are in fact deterministic, typically producing numbers that repeat after a (very) long period. A commonly used PRNG is the linear congruential generator

$$X_{i+1} = mod\left((aX_i + b),\ m\right)$$

where a, b and m are chosen so that they don't have any common factors. Recall that the mod function takes (aX+b), divides by m and gives the remainder, a number between 0 and 1, uniformly distributed. More information can be found at: https://en.wikipedia.org/wiki/Linear_congruential_generator. In Matlab, the equation above would be written x(i+1)=mod (a*x(i)+b, m).

Once you have a PRNG that will give uniformly distributed numbers x between zero and 1 (pdf u(x)=1) you can generate variables z with any desired pdf f(z). The key idea is that the mapping function between x and z=g(x) should satisfy u(x)dx=f(z) dz. Why? Suppose we generate 10,000 random values of x, each of which gets mapped onto 10,000 random values of z. If 100 of them occur in the range x to x+dx, then we will have the same number in the range z to z+dz. Well, this number should be proportional to the probability of these values of x and z, given by u dx and f dz. Further, the integrals over all x (or z) must be 1, and integrating both sides from the lower end of the variable range, U(x)=F(z) where U and F are the cumulative pdf functions. If x is a uniformly distributed number between 0 and 1, then U(x)=x. Assuming we can find the inverse F^{-1} of F, we obtain our random variable $z=F^{-1}(x)$. The PRNGs in Matlab use this approach to generate a wide range of distributions. Excel does not have such an extensive library of PRNG functions and you might have to use the above approach to generate random variables of the type you need. For example, if you want to generate normally distributed random variables in Excel, use NORMINV(mean, sigma,RAND()).

Central Limit Theorem (CLT)

According to the Central Limit Theorem, a random variable z that is itself a sum of M random variables x_i, will be distributed normally if M is large *even if the component variables x_i are NOT normally distributed.* Because of this, many forms of distribution tend towards normality in limiting cases. Taylor's text does not address CLT directly; the closest to this is Section 10.5. The CLT is not easy to prove, but quite easy to demonstrate using simulations.

In this tutorial we will consider what happens when the total random error in our measurements is actually the sum of M small "binary" ("coin toss") errors,

$$\varepsilon_i = \sum_{j=1}^{M} \delta_j$$

$$\delta_j = \pm \frac{s}{\sqrt{M}}$$

The sign in the second equation is chosen randomly for each δ_j, s is a constant in our simulation that will turn out to be close to the standard deviation of ε, thanks to the square root term in the denominator.

Error Propagation

Suppose that z is a linear function of random variables w,x,y,

$$z = aw + bx + cy + d$$

then the mean of z is

$$\mu_z = a\mu_w + b\mu_x + c\mu_y + d$$

The standard deviation of z will be (<u>if the variables are independent</u>)

$$\sigma_z = \sqrt{a^2\sigma_w^2 + b^2\sigma_x^2 + c^2\sigma_y^2}$$

This is a useful result in experimental work, because we may make measurements *w, x, y* with known uncertainty (proportional to standard deviation), but we actually want to know the uncertainty of some computed result z. If z is a linear function of w,x,y (or any number of measurements), the results above can be used immediately, provided that the error for each measurement is normally distributed, and independent of the errors for the other measurements (this last assumption is often questionable, as there are often factors that affect the errors of all

measurements). If $z = f(w,z,y)$ is NOT linear, then the relations above do not hold exactly but it is common to assume that

$$\mu_z = f(\mu_w, \mu_x, \mu_y)$$

If we linearize *f* about these mean values,

$$z = \frac{\partial f}{\partial w} w + \frac{\partial f}{\partial x} x + \frac{\partial f}{\partial y} y + constant$$

The partial derivatives (evaluated at the mean values of x, y, z) replace *a, b* and *c* so now the standard deviation can be computed with the result for a linear function we had above.

Tutorial Assignment

You should attempt the tutorial assignment before the tutorial period, so that you can identify the difficult points before class and therefore have more focused questions during the tutorial.

1. Write a Matlab script to generate random "errors" according to

$$\varepsilon_i = \sum_{j=1}^{M} \delta_j \quad ; \quad \delta_j = \pm \frac{s}{\sqrt{M}}$$

   Take s=1 and try M=1,4,16,32. Plot the histograms using "pdf" normalization such that the area under them is 1. For each value of *M*, use 1000 values of ε_i to obtain the histogram. Starter code attached provides some guidance on the syntax. Compare with a normal distribution with mean zero and standard deviation 1.

2. Suppose we wish to determine the Young's modulus E of wood by measuring the stiffness of a cantilever beam. We need to measure the beam length L, depth h, width b and deflection δ for load F at the end. You know the relation:

$$\delta = \frac{FL^3}{3EI}$$

$$I = \frac{bh^3}{12}$$

   Consider specific measurements given in the table below.

		Error in X (sigma)	
Parameter X	Measurement	dX/X*100	dX
L (m)	0.7	2	0.014
h (m)	0.03	10	0.003
b (m)	0.04	1	0.0004
F (N)	80	1	0.8
delta (m)	0.008	1	0.00008

   Each of our measurements contain uncertainty, given below as a percent of the measured value, and as the absolute value (take dX in the table as σ).

   a. Use the error propagation formulas in the Background to analytically estimate the uncertainty (σ_E) of E derived from the measurements (ie, not MATLAB program here).

b. Use the randn matlab function to generate 10,000 sets of simulated parameters L,h,b,δ, F that go into computing E. For each of these sets compute E using the beam formula and plot the histogram of E at the end. Compare this with the result from a.
c. Suppose you wanted to improve your measurement of E. What measurements could be improved to give the biggest benefit?

Appendix

Matlab functions needed are rand and randn. Google "matlab"+ function name and you will get good documentation. One thing to watch for: rand(N) gives you an NxN matrix of random numbers, not a vector. Instead call rand(1,N) to get the vector.

Starter code for part 1

This code was used in 2017W when we covered poisson and binomial distributions earlier. This year you will use your own random variable, but your code will still look similar.

```
% Mech 305 306 Statistics Functions
% Illustration of the central limit theorem
lambda=10
NN=9999
z=poissrnd(lambda,1,NN);
y=randn(1,NN);% this is the standard normal
%transform variables so that the Poisson and normal distributions can be
%compared
sig= ;
meany= ;% poisson variance is the mean
yy=
figure(1)
histogram(yy,'Normalization','pdf','DisplayStyle','stairs')
hold on
histogram(z,'Normalization','pdf','DisplayStyle','stairs')

% one can see that the poisson distribution approximates the normal
% distribution for large lambda.  This is an illustration of the central
% limit theorem
%the corresponding binomial distribution with the same mean for say
N=20
p=

b=binornd(N,p,1,NN);
histogram(b,'Normalization','pdf','DisplayStyle','stairs')
legend('normal','poisson','binomial')
hold off
figure(3)
normplot(b)
hold on
normplot(yy)
hold off
```

Starter code for part 2

```
% Tutorial 2 simulation of beam bending experiment
clear all
%% Here are the baseline measurements, in SI
L0=0.7 ;% beam length in m
h0=0.03;% beam depth in m
b0=0.04;% beam width in m
F0=80;% applied force at end of cantilever beam, N
delta0=0.008 ;% displacement at end
% put these into the beam bending equation to get baseline estimate of E
E0=
% to construct random variables out of these, need standard deviations as
% as PERCENT OF MEAN
sLp=2;%
shp=10;
sbp=1;
sFp=1;
sdelp=1;
% express these now as standard deviations in the physical units (not
% relative values
sL=sLp*L0/100;% same units as L
sh=shp*h0/100;
sb=sbp*b0/100;
sF=sFp*F0/100;
sdel=sdelp*delta0/100;
%% simulation
N=    ;% number of trials
L= % construct normally distributed variables for all parameters

E=   %this is now a vector of values for Young's modulus

sErel=std(E)/E0  %Much easier to view results relative to E0
meanE=mean(E)/E0

figure(1)
histogram(L/L0,'Normalization','pdf','DisplayStyle','stairs');
hold off
```