ENGINEERING TRIPOS PART II A

EIETL **MODULE EXPERIMENT 3F3**

RANDOM VARIABLES and RANDOM NUMBER GENERATION

You should read this handout and understand the theory *before* you arrive for the laboratory session. Otherwise you will find it difficult to complete the experiment in the allotted time and you risk being penalised by the demonstrators.
The lab can be completed using Matlab OR Python - you may choose which language to use

OBJECTIVES

- To introduce the idea of random variables and functions of random variables
- To study the Jacobian as used with random variables
- To experiment with methods for non-uniform random number generation

1 Introduction

Random number generation is a vital part of many engineering, computer science and physical science applications. Random numbers are used in simulation of physical systems, in performing inference in otherwise intractable problems and in the study of random phenomena such as mechanical vibrations, earthquakes, etc. While uniformly distributed pseudo-random numbers can readily be generated by several reliable means, how do we convert these uniform variates into samples from other desired distributions which may be required in particular applications?

In this practical we will study the generation of non-uniformly distributed random numbers in Matlab/Python, introducing some of the principal tricks involved in generation from desired probability distributions. We will assume that basic number generators for generating uniform or normally distributed random numbers are already available, as in Matlab. Random number generation will be used as a vehicle to demonstrate the theory of change of variables using Jacobians, as detailed below.

The Matlab commands `rand` and `randn` can be used to generate vectors of pseudo-random numbers from the uniform distribution:

$$p(x) = \mathcal{U}(x|0,1) = \begin{cases} 1, & 0 < x < 1 \\ 0, & \text{otherwise} \end{cases}$$

or from the normal distribution (mean zero, variance 1):

$$p(x) = \mathcal{N}(x|0,1) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$$

Sometimes for conciseness we will simply write $\mathcal{U}(0,1)$ or $\mathcal{N}(0,1)$ for these distributions. The equivalent commands in Python are `numpy.random.rand` and `numpy.random.randn`.

1.1 A few revision basics

- The normal distribution:

  $$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

- The exponential distribution:

  $$\mathcal{E}(x|\mu) = \frac{1}{\mu}\exp(-x/\mu), \qquad x \geq 0,$$

  where μ is the mean.

- The cumulative distribution function for a random variable:

  $$F(x) = \Pr(X \leq x)$$

- Probability density function:

  $$p(x) = dF(x)/dx$$

- Expectation of a random variable:

  $$\mu = \mathbb{E}[X] = \int_{x=-\infty}^{\infty} xp(x)dx$$

- Expectation of a *function* $f(x)$ of a random variable:

  $$\mathbb{E}[f(X)] = \int_{-\infty}^{\infty} f(x)p(x)dx$$

  In the experiment we will sometimes refer to this formula as the expectation of $f(x)$ wrt the probability density $p(x)$.

- Variance of a random variable:

  $$\sigma^2 = \mathbb{E}[(X-\mu)^2] = \mathbb{E}[X^2] - \mu^2$$

  and σ is termed the *standard deviation* of X.

1.2 Histogram estimation

During the practical you will use histogram count data. Here we outline some of the theory of histogram counts which you can use in the practical.

Take a set of N independent random samples from a probability distribution:

$$x^{(i)} \sim p(x), \quad i = 1, ..., N$$

We may estimate the shape of the density by histogramming the data (Matlab function `hist`, Python function `matplotlib.pyplot.hist`). The probability that a sample $x^{(i)}$ lies within a particular bin of the histogram is:

$$p_j = \int_{c_j-\delta/2}^{c_j+\delta/2} p(x)dx$$

where c_j is the jth bin centre, $j = 1, ..., J$, and δ the bin width. If we draw N random variables and count the number that lie within each bin, say n_j, the probability of the histogram data is given by:

$$\frac{N!}{n_1!n_2!...n_J!} p_1^{n_1} p_2^{n_2} ... p_J^{n_j} \tag{1}$$

This is the *multinomial* distribution. The mean of the count data in bin j is Np_j and the variance is $Np_j(1-p_j)$.

1.3 Matlab/Python and general Preparation

In order to get through this lab in the recommended time you will need to be familiar with basic Matlab functions and plotting capabilities. The following, largely uncommented, snippet of code gets you started on the first part of the practical. Make sure you understand the arguments supplied to the `hist` and `ksdensity` functions:

```
% Plot normal distribution:
figure(1)
x=randn(1000,1);
subplot(211),
hist(x,20)
subplot(212),
ksdensity(x,'width',0.1)

% Plot uniform distribution:
figure(2)
x=rand(1000,1);
subplot(211),
% Be aware that when specifying bin centres as below, Matlab will include everything
% from -infinity to the first bin centre in the first bin,
% and the top bin to +infnity in the last bin
hist(x,[-1.45:0.1:1.45])
subplot(212),
ksdensity(x,'width',0.1)
```

And for Python the `ksdensity` function is provided below:

```
import numpy as np
import matplotlib.pyplot as plt

def ksdensity(data, width=0.3):
    """Returns kernel smoothing function from data points in data"""
    def ksd(x_axis):
        def n_pdf(x, mu=0., sigma=1.):  # normal pdf
            u = (x - mu) / abs(sigma)
            y = (1 / (np.sqrt(2 * np.pi) * abs(sigma)))
            y *= np.exp(-u * u / 2)
            return y
        prob = [n_pdf(x_i, data, width) for x_i in x_axis]
        pdf = [np.average(pr) for pr in prob]  # each row is one x value
        return np.array(pdf)
    return ksd

# Plot normal distribution
fig, ax = plt.subplots(2)
x = np.random.randn(1000)
ax[0].hist(x, bins=30) # number of bins

ks_density = ksdensity(x, width=0.4)
# np.linspace(start, stop, number of steps)
x_values = np.linspace(-5., 5., 100)
ax[1].plot(x_values, ks_density(x_values))

# Plot uniform distribution
fig2, ax2 = plt.subplots(2)
x = np.random.rand(1000)
ax2[0].hist(x, bins=20)

ks_density = ksdensity(x, width=0.2)
x_values = np.linspace(-1., 2., 100)
ax2[1].plot(x_values, ks_density(x_values))

plt.show()
```

There are also some required theoretical calculations that can be drafted out before the experiment if you would like to speed your progress prior to the lab session itself - see below.

The whole practical can be carried out in your own time if preferred. However, you will need to attend at least the start of the lab session, and it is recommended to attend the whole session so that you can check your ideas and working with the demonstrators.

2 Experimental Work

1. **Uniform and normal random variables** Use Matlab/Python to generate a vector of 1000 Gaussian random numbers and 1000 uniform random variables. Plot histograms of the generated numbers and overlay these on plots of the exact normal and uniform probability density functions (scaled appropriately to match the histogram count data).

   Experiment also with using the `ksdensity` function in Matlab/Python, to plot a smooth density function estimate from the samples using the formula:

   $$\pi_{KS}(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\sigma}\mathcal{K}\left(\frac{x - x^{(i)}}{\sigma}\right) \tag{2}$$

   using for example the Gaussian kernel, $\mathcal{K}(x) = \mathcal{N}(x|0,1)$. Comment on the advantages and disadvantages of the kernel density method compared with the histogram method for estimation of a probability density from random samples.

   Long report: Explain the form of the multinomial distribution; in particular why do we get the terms $p_j^{n_j}$, and why the factorial terms?

   For the uniform distribution, calculate the theoretical mean and standard deviation for the histogram count data, as a function of N (using the multinomial distribution theory from above). Explain what happens to the histogram estimate as N becomes large.

   For $N = 100$, $N = 1000$ and $N = 10000$, plot the data histogram, the theoretical mean of the histogram data and also the ± 3 standard deviation lines. Are your histogram results consistent with the multinomial distribution theory? (in other words, are the histograms consistent with Matlab/Python having generated accurate uniformly distributed random variates?).

   Long report: repeat this for normally distributed data. Comment on how the histogram variance depends on the bin probabilities (consider p_i close to zero, one, and intermediate). For this part you will need to carefully calculate the bin probabilities p_j using the cdf function, `normcdf` *in Matlab,* `scipy.stats.norm.cdf` *in Python.*

2. **Functions of random variables** In engineering systems, random variables very often pass through functions, for example we might take the magnitude or the squared value of a random signal when performing detection of 'significant' events. In other cases we might measure a sine or cosine function at a randomly chosen phase offset. Here we will study the distribution of these transformed random variables.

   The theoretical answer involves the *Jacobian* of the transformation.

   Consider a general function of the random variable x:

   $$y = f(x)$$

   If we know that x is distributed with density function $p(x)$, then we can calculate the output density $p(y)$ as follows, assuming that $f()$ is a one-to-one, invertible, differentiable function:

$$p(y) = \left.\frac{p(x)}{|dy/dx|}\right|_{x=f^{-1}(y)}$$

If there is more than one possible value for $f^{-1}(y)$, say $x_1(y), x_2(y), ...x_K(y)$, then the solution is the sum over all possible solutions:

$$p(y) = \sum_{k=1}^{K} \left.\frac{p(x)}{|dy/dx|}\right|_{x=x_k(y)}$$

See the 3F3 lecture notes for the derivation and general case...

Example:

Take $p(x) = \mathcal{U}(-0.5, +0.5)$, i.e. uniform between -0.5 and $+0.5$:

$$p(x) = \begin{cases} 1, & -0.5 < x < 0.5 \\ 0, & \text{Otherwise} \end{cases}$$

and $f(x) = |x|$. Now,

$$df(x)/dx = \begin{cases} 1, & x > 0 \\ -1, & x < 0 \end{cases}$$

and so the Jacobian is $|dy/dx| = 1$ for all $x \neq 0$ (undefined at $x = 0$).

Now, $y = |x|$ has two possible inverses, i.e. $x_1(y) = y$ or $x_2(y) = -y$. Hence the new density function is:

$$p(y) = \sum_{k=1}^{2} \left.\frac{p(x)}{|dy/dx|}\right|_{x=x_k(y)} = p(y)/1 + p(-y)/1 = \begin{cases} 2, & 0 < y < 0.5 \\ 0, & \text{Otherwise} \end{cases}$$

where the zero case takes account of $p(x)$ being zero when $|x| > 0.5$. Thus in this simple case we get another uniform distribution, $p(y) = \mathcal{U}(0, 0.5)$, which of course has height equal to 2 to make it integrate to 1 as a probability density function.

Now, for normally distributed $\mathcal{N}(x|0,1)$ random variables, take $y = f(x) = ax+b$. Calculate $p(y)$ using the Jacobian formula as above, and explain how this is linked to the general normal density with non-zero mean and non-unity variance. Verify this formula by transforming a large collection of random samples $x^{(i)}$ to give $y^{(i)} = f(x^{(i)})$, histogramming the resulting y samples, and overlaying a plot of your formula calculated using the Jacobian.

Now take $p(x) = \mathcal{N}(x|0,1)$ and $f(x) = x^2$. Calculate $p(y)$ using the Jacobian formula and once again verify your result by histogramming of transformed random samples.

Long report: Take $p(x) = \mathcal{U}(x|0, 2\pi)$ and $f(x) = \sin(x)$ – this corresponds to measuring e.g. a carrier signal in a comms. system at a random phase offset. Determine the theoretical probability density for $f(x)$ and verify once again by comparison with transformed random samples. Now, instead of the sin function, *consider a limited sin function, an idealised version of the 'clipping' or limiting that can occur when electronics saturate:*

$$f(x) = \min(\sin(x), 0.7)$$

Determine experimentally from random samples what the output density is for y and comment on how this could be predicted from the Jacobian formula – pay particular attention to $\sin(x) > 0.7$.

3. **Inverse CDF method** Transformations of random variables leads to a very neat and general method for non-uniform random variable generation. Take $p(x) = \mathcal{U}(x|0,1)$, and consider transforming x to some other distribution using $y = f(x)$, where now we assume the simple first case where $f(x)$ is invertible and differentiable. Thus we have

$$p(y) = \begin{cases} \left.\frac{1}{|dy/dx|}\right|_{x=f^{-1}(y)}, & f^{-1}(y) \in (0,1) \\ 0, & \text{otherwise} \end{cases}$$

Now, suppose that samples are required from a specified density function $p(y)$ but we only have a uniform random number generator $p(x) = \mathcal{U}(x|0,1)$. We can choose an appropriate $f(x)$ which would achieve this. It would need to satisfy:

$$p(y) = \frac{1}{|dy/dx|} = |dx/dy| = |df^{-1}(y)/dy|$$

since $(dy/dx)^{-1} = dx/dy$ by standard derivative calculus. If we assume a function can be chosen whose derivative is always greater than zero, then we would have a simpler equation:

$$p(y) = df^{-1}(y)/dy$$

But we know that the first derivative of the cumulative distribution function (CDF) $F(y)$ is the density function, by definition. We know too that it is non-decreasing, so its derivative is always greater than (or equal to) zero, as required (assume the CDF is always strictly increasing for now, although the method presented here also works for the general case). Thus an appropriate solution is:

$$f^{-1}(y) = F(y),$$

or, equivalently:

$$f(x) = F^{-1}(x)$$

Thus, if we can evaluate the inverse CDF function $F^{-1}(x)$, we can generate exact random samples from $p(y)$ by generating uniform samples $x^{(i)} \sim \mathcal{U}(0,1)$ and computing

$$y^{(i)} = F^{-1}(x^{(i)})$$

This is the inverse CDF method, one of the simplest and most general methods for generation of random variates with specified probability density.

Now consider the exponential distribution with mean one:

$$p(y) = \exp(-y), \;\; y \geq 0$$

Calculate the CDF and the inverse CDF for this distribution.

Hence use the inverse CDF method to generate samples from the exponential distribution. Plot histograms/ kernel density estimates and overlay them on the desired exponential density to convince yourself of successful results.

Long report: Use your random samples to estimate the mean and variance of the exponential distribution, and verify that they are close to the expected theoretical values for this exponential distribution. The mean can be estimated by Monte Carlo:

$$\mu = \mathbb{E}[Y] = \int_{y=0}^{\infty} y p(y) dy \approx \frac{1}{N}\sum_{i=1}^{N} y^{(i)} = \hat{\mu}$$

Intuitively, the Monte Carlo approach estimates expectation of a function $f(x)$ by simply taking the arithmetic mean of the sampled function values $f(x^{(i)})$.

Long Report: Show that the Monte Carlo mean estimate $\hat{\mu}$ is unbiased, *i.e. that $\mathbb{E}[\hat{\mu}] = \mu$, where the first expectation is taken wrt the distribution of the random samples $y^{(i)}$, i.e. $p(y)$.*

The variance can be estimated in a similar way by Monte Carlo:

$$\sigma^2 = \mathbb{E}[Y^2] - \mu^2 = \int_{y=0}^{\infty} y^2 p(y) dy - \mu^2 \approx \frac{1}{N}\sum_{i=1}^{N} \left(y^{(i)}\right)^2 - (\hat{\mu})^2 = \hat{\sigma^2}$$

Optional Challenge (Long Report): show from expectation formulae that the Monte Carlo mean estimator has variance equal to σ^2/N, i.e. that

$$\mathbb{E}[\hat{\mu}^2 - \mu^2] \propto 1/N$$

Once again, the expectation is taken wrt the distribution of the $y^{(i)}$s.

Long Report: Illustrate this property by plotting the squared error $(\hat{\mu} - \mu)^2$ as the number of Monte Carlo samples increases. This should strictly be done by averaging the error over lots of Monte Carlo estimates of μ from different samples $y^{(i)}$, but you should be able to see the trend clearly from one set of samples. Here you are showing that the Monte Carlo estimator is also consistent *in that it delivers the true value of μ as the number of samples N goes to infinity. Note that this is considered quite a slow convergence rate as the values of the estimate only converge as $1/\sqrt{N}$ and hence Monte Carlo should only be used when other approximations fail.*

4. **Simulation from a 'difficult' density.** Sometimes random variables cannot be simulated using inverse cdf or other convenient means. However, in such cases there may still be numerical 'recipes' for generating such variables. One such case is considered now. This particular distribution is fundamental in the study of communications data, signal processing and Internet of Things, where interfering noise can be quite far from the Gaussian assumptions often imposed.

   The 'recipe' is as follows:

(a) Choose some parameters $\alpha \in (0,2)$ $(\alpha \neq 1)$, $\beta \in [-1,+1]$ and calculate some constants:

$$b = \frac{1}{\alpha}\tan^{-1}(\beta\tan(\pi\alpha/2)), \quad s = (1+\beta^2\tan^2(\pi\alpha/2))^{\frac{1}{2\alpha}}$$

(b) Generate a uniform random variable $U \sim \mathcal{U}(-\pi/2, +\pi/2)$.

(c) Generate an exponential random variable with mean 1: $V \sim \mathcal{E}(V|1)$.

(d) Use U and V to calculate X:

$$X = s\frac{\sin(\alpha(U+b))}{(\cos(U))^{1/\alpha}}\left(\frac{\cos(U-\alpha(U+b))}{V}\right)^{\frac{1-\alpha}{\alpha}},$$

(e) X is the random variable of interest

Write some Matlab code using this recipe to generate N random numbers drawn from the distribution of X. You may use your own exponential random number generator, or Matlab's own integrated version `exprnd`.

Plot some histogram density estimates with $\alpha = 0.5$, 1.5 and several values of β. Choose your bin centres carefully so that the centre of the distribution is clearly visible on your plots – note that this probability distribution can generate some occasionally very large numbers!

Hence comment on the interpretation of the parameters α and $\beta \in [-1,+1]$.

Long report: with $\beta = 0$, use your random numbers to estimate the tail probability *for the distribution, i.e. the probability that $|X| > t$, where $t = 0, 3, 6$, and for the two cases $\alpha = 0.5$ and $\alpha = 1.5$. How does this compare with the standard Gaussian? For these two values of α, try to determine the* tail behaviour *of the pdf $p(x)$, in this case a curve of the form $p(x) \approx cx^\gamma$ for large $|x|$, where γ is to be determined in each case and c is a constant. Make sure you use enough random samples to get reliable estimates for all of these quantities. Suggest a general expression for γ in terms of α and verify this for several values of α.*

Long report: Inspect the characteristics of the distribution as α gets closer to 2 and comment on links with the Gaussian.

[For hints and further reading, see Wikipedia entry for the α-stable distribution, a highly intractable distribution for which even pdfs cannot usually be written down in closed form.]

3 Write-up.

All reports are to be handed in electronically through the 3F3 Moodle website, within the general deadlines specified for IIA coursework.

The Laboratory Report (Short Report)

In the short report, provide answers to all of the questions and tasks in the instructions. You should include brief and clear derivations of any required theoretical results, Matlab/Python graphs of results and you should cut and paste the code you wrote for each part into the relevant section of the report.

You may write the report in Latex (recommended) or Word.

For the short report you are *not* required to include introduction/ methods/ apparatus/ conclusion sections. Rather, you may follow the template provided on the Moodle page for 3F3, including your results and calculations at each step. You do not need to provide responses to the items in italics beginning '*Long Report:*'

The Long Report

The Long report requires a couple of hours of extra coding work plus a more detailed write-up. For the long report, provide a full write-up of the practical, including the extra '*Long Report:*' items. You should have an Introduction and Conclusion, with clearly reasoned answers to all questions, from both the short and long report topics.