

Do Random phenomena exist in Nature?

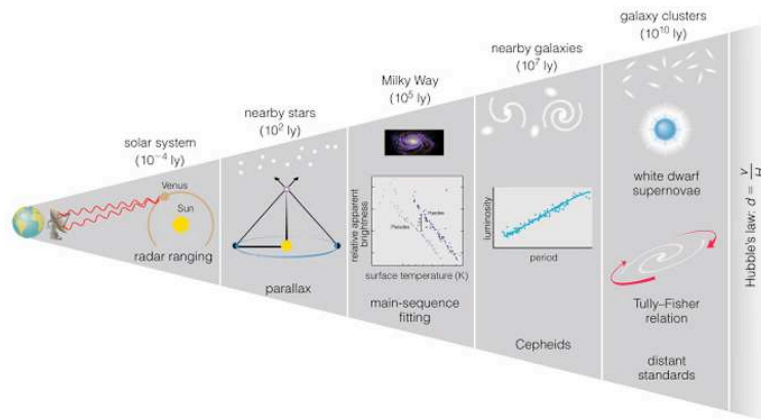
Which way a coin tossed in air will fall may be completely determined by laws of physics. The only problem in figuring out the trajectory and hence the face of the coin when it is on ground is that we have to measure too many parameters, e.g. angular momentum of rotation, force at the time of toss, wind pressure at various instants during the rotation of the coin, etc.!

Which way an electron will spin is also not known and so modelling it will require incorporating a **random** structure. But we cannot exclude the possibility that sometime in future, someone will come up with a theory that will explain the spin.

Thus we often come across events whose outcome is uncertain. The uncertainty could be because of our inability to observe accurately all the inputs required to compute the outcome.

It may be too expensive or even counterproductive to observe all the inputs. The uncertainty could be due to the current level of understanding of the phenomenon. The uncertainty could be on account of the outcome depending on choices made by a group of people at a future time - such as outcome of an election yet to be held.

Cosmic distance ladder The objects in the solar system were measured quite accurately by ancient Greeks and Babylonians using geometric and trigonometric methods.



see also www.math.ucla.edu/~tao/preprints/Slides/Cosmic%20Distance%20Ladder.ppt

The distance to the stars in the second lot are found by ideas of parallax, calculating the angular deviation over 6 months. First done by the mathematician

Friedrich Bessel. The error here is of the order of 100 light years.

The distances of the moderately far stars are obtained by a combination of their apparent brightness and the distance to the nearby stars. This method works for stars upto 300,000 light years and the error is significantly more.

The distance to the next and final lot of stars is obtained by plotting the oscillations of their brightness. This method works for stars upto 13,000,000 light years!

At every step of the distance ladder, errors and uncertainties creep in. Each step inherits all the problems of the ones below, and also the errors intrinsic to each step tend to get larger for the more distant objects; thus the spectacular precision at the base of the ladder degenerates into much greater uncertainty at the very top.

So we need to understand **UNCERTAINTY**.

And the only way of understanding a notion scientifically is to provide a structure to the notion.

A structure rich enough to lend itself to quantification.



The structure needed to understand a coin toss is intuitive.

We assign a probability $1/2$ to the outcome **HEAD** and a probability $1/2$ to the outcome **TAIL** of appearing.



Similarly for each of the outcomes **1,2,3,4,5,6** of the throw of a dice we assign a probability $1/6$ of appearing.



Similarly for each of the outcomes $000001, \dots, 999999$ of a lottery ticket we assign a probability $1/999999$ of being the winning ticket.

Of course, we could obtain the structure of the uncertainty in a coin toss from the example of throwing a dice.

In particular if we declare as **HEAD** when the outcome of a throw of a dice is an even number, and if we declare as **TAIL** when the outcome of a throw of a dice is an odd number, then we have the same structure as that we had from a coin toss.

More generally, associated with any experiment we have an outcome space Ω consisting of outcomes $\{o_1, o_2, \dots, o_m\}$.

Coin Toss – $\Omega = \{H, T\}$

Dice – $\Omega = \{1, 2, 3, 4, 5, 6\}$

Lottery – $\Omega = \{1, \dots, 999999\}$

Each outcome is assigned a probability

Coin Toss – $p_H = 1/2, p_T = 1/2$

Dice – $p_i = 1/6$ for $i = 1, \dots, 6$

Lottery – $p_i = 1/999999$ for $i = 1, \dots, 999999$

More generally, for an experiment with an outcome space $\Omega = \{o_1, o_2, \dots, o_m\}$, we assign a probability p_i to the outcome o_i for every i in such a way that the probabilities add up to 1.

The set $\Omega = \{o_1, o_2, \dots, o_m\}$ is called a sample space.

A subset $E \subseteq \Omega$ is called an event.

We may be gambling with dice, so we could have a situation like

<i>outcome</i>	1	2	3	4	5	6
<i>money amount</i>	-8	2	0	4	-2	4

Our interest in the outcome is only *vis-à-vis* its association with the monetary amount.

So we are interested in a mapping (i.e. a function) of the outcome space Ω to the reals \mathbb{R}

Such functions are called random variables.

The probabilistic properties of these random variables can be read out from the probabilities assigned to the outcomes of the underlying outcome space.

The probability that you win 4 rupees, i.e. $P\{X = 4\}$ means you want to find that the number 4 or the number 6 came out on the dice, i.e. $P\{4, 6\}$. Thus $P\{\omega : X(\omega) = 4\} = P\{4, 6\} = (1/6) + (1/6) = 1/3$.

Similarly the probability that you do not lose any money is the probability of the event that either 2, 3, 4 or 6 came out on the dice, and this probability is $(1/6) + (1/6) + (1/6) + (1/6) = 2/3$.

What are we doing?

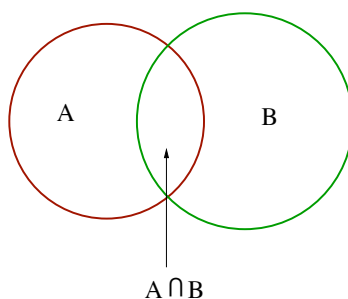
Recall our assignment of probabilities $P(o_i) = p_i$ on the outcome space $\Omega = \{o_1, o_2, \dots, o_m\}$.

For an event $E = \{o_{i_1}, o_{i_2}, \dots, o_{i_k}\}$, we define

$$P(E) = p_{i_1} + p_{i_2} + \dots + p_{i_k}.$$

Easy to check that if A, B are mutually disjoint, i.e. $A \cap B = \phi$ then

$$P(A \cup B) = P(A) + P(B)$$



More generally, we can check that for any two events A, B

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

Similarly, for three events A, B, C

$$\begin{aligned} P(A \cup B \cup C) &= P(A) + P(B) + P(C) \\ &\quad - P(A \cap B) - P(A \cap C) - P(B \cap C) \\ &\quad + P(A \cap B \cap C) \end{aligned}$$

This has a generalization to n events.

How do we assign the probabilities p_i to the elementary outcomes?

The simplest case is when due to inherent symmetries present, we can model all the elementary events (*i.e.* outcomes) as being **equally likely**.

vspace2mm

When an experiment results in m equally likely outcomes o_1, o_2, \dots, o_m , the probability of an event A is

$$P(A) = \frac{\#A}{m}$$

i.e. the ratio of the number of favourable outcomes to the total number of outcomes.

Example: Toss a coin three times

$\Omega = \{HHH,$ $HHT, HTH, THH,$ $HTT, THT, TTH,$ $TTT\}$ $p(***) = 1/8$	No. of Heads in 3 tosses $\Omega = \{0, 1, 2, 3\}$ \longleftarrow 3 Heads \longleftarrow 2 Heads \longleftarrow 1 Head \longleftarrow 0 Heads $p(0) = 1/8, p(1) = 3/8,$ $p(2) = 3/8, p(3) = 1/8$
--	--

Note we could have done the calculations in the red part without even associating it with the blue sample space etc.

Conditional probability Let X be the number which appears on the throw of a dice.

Each of the six outcomes is equally likely, but suppose I take a peek and tell you that X is an even number.

Question: What is the probability that the outcome belongs to $\{1, 2, 3\}$?

Given the information I conveyed, the six outcomes are no longer equally likely. Instead, the outcome is one of $\{2, 4, 6\}$ – each being equally likely.

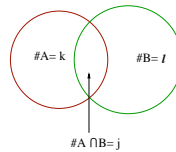
So with the information you have, the probability that the outcome belongs to $\{1, 2, 3\}$ equals $1/3$.

Consider an experiment with m equally likely outcomes and let A and B be two events.

Given the information that B has happened, what is the probability that A occurs?

This probability is called the **conditional probability of A given B**

and written as $P(A | B)$.



Let $\#A = k$, $\#B = l$, $\#(A \cap B) = j$.

Given that B has happened, the new probability assignment gives a probability $1/l$ to each of the outcomes in B .

Out of these l outcomes of B ,
 $\#(A \cap B) = j$ outcomes also belong to A .

Hence $P(A | B) = j/l$.
 Noting that $P(A \cap B) = j/m$ and $P(B) = l/m$, it follows that

$$P(A | B) = \frac{P(A \cap B)}{P(B)}.$$

In general when A, B are events such that $P(B) > 0$, the conditional probability of A given that B has occurred $P(A | B)$ is defined by

$$P(A | B) = \frac{P(A \cap B)}{P(B)}$$

This leads to the **Multiplicative law of probability**

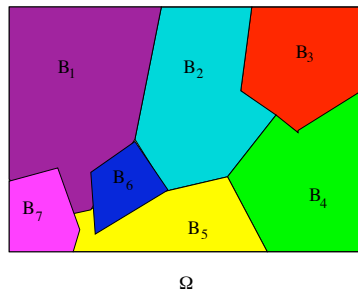
$$P(A \cap B) = P(A | B)P(B)$$

This has a generalization to n events:

$$\begin{aligned} &P(A_1 \cap A_2 \cap \dots \cap A_n) \\ &= P(A_n | A_1, \dots, A_{n-1}) \\ &\quad \times P(A_{n-1} | A_1, \dots, A_{n-2}) \\ &\quad \times \dots \times \\ &\quad \times P(A_2 | A_1)P(A_1) \end{aligned}$$

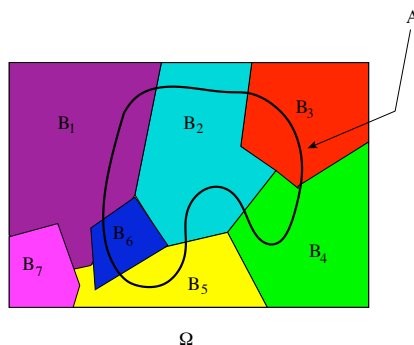
The Law of Total Probability

Let B_1, \dots, B_k be a partition of the sample space Ω



The Law of Total Probability

Let B_1, \dots, B_k be a partition of the sample space Ω and A an event



Then
$$P(A) = P(A \cap B_1) + \dots + P(A \cap B_k)$$

Also we know
$$P(A \cap B_i) = P(A|B_i)P(B_i)$$

so we get the Law of Total Probability

$$P(A) = P(A|B_1)P(B_1) + \dots + P(A|B_k)P(B_k)$$

Example

Suppose a bag has 6 one rupee coins, exactly one of which is a **Sholay coin**, i.e. both sides are **HEAD**. A coin is picked at random and tossed 4 times, and each toss yielded a **HEAD**.

Two questions which may be asked here are

- (i) what is the probability of the occurrence of $A = \{\text{all four tosses yielded HEADS}\}$?
- (ii) **given** that A occurred, what is the probability that the coin picked was the **Sholay coin**?

The first question is easily answered by the laws of total probability. Let

B_1 = coin picked was a regular coin

B_2 = coin picked was a **Sholay coin**

Then

$$\begin{aligned} P(A) &= P(A | B_1)P(B_1) + P(A | B_2)P(B_2) \\ &= \left(\frac{1}{2}\right)^4 \frac{5}{6} + \frac{1}{6} \\ &= \frac{21}{96} = \frac{7}{32} \end{aligned}$$

For the second question we need to find

$$\begin{aligned}
 P(B_2 | A) &= \frac{P(B_2 \cap A)}{P(A)} \\
 &= \frac{P(A | B_2)P(B_2)}{P(A)} \\
 &= \frac{1/6}{7/32} \\
 &= \frac{16}{21}
 \end{aligned}$$

The previous example is atypical of the situation where we perform scientific experiments and make observations. On the basis of the observations we have to infer what was the theoretical process involved in the experiment to obtain the given observation. Occasionally we may have some (though not complete) information of the process, in which case we can use this information to help in our inference.

In particular, in the example we had the **prior** information that there was exactly one **Sholay coin** among the six coins.

Suppose we have observed that A occurred.

Let B_1, \dots, B_m be all possible scenarios under which A may occur, i.e. B_1, \dots, B_m is a partition of the sample space. To quantify our suspicion that B_i was the cause for the occurrence of A , we would like to obtain $P(B_i | A)$.

Bayes' formula or Bayes' theorem is the prescription to obtain this quantity. The theorem is very easy to establish and is the basis of **Bayesian Inference**.

Bayes' Theorem:

If B_1, B_2, \dots, B_m is a partition of the sample space, then

$$\begin{aligned}
 P(B_i | A) &= \frac{P(A | B_i)P(B_i)}{P(A)} \\
 &= \frac{P(A | B_i)P(B_i)}{\sum_{j=1}^m P(A | B_j)P(B_j)}
 \end{aligned}$$

Suppose that A, B are events such that

$$P(A | B) = P(A)P(B).$$

Then we get

$$P(A | B) = P(A).$$

i.e. the knowledge that B has occurred has not altered the probability of A .

In this case, A and B are said to be **independent** events.

Let X, Y, Z be random variables each taking finitely many values. Then X, Y, Z are said to be independent if

$$P(X = i, Y = j, Z = k) = P(X = i)P(Y = j)P(Z = k)$$

for all possible values i, j, k of X, Y, Z respectively.

This can be generalized to finitely many random variables.

Expectation of a random variable

Let X be a random variable taking values x_1, x_2, \dots, x_n . The expected value μ of X (also called the **mean** of X) denoted by $E(X)$ is defined by

$$\mu = E(X) = \sum_{i=1}^n x_i P(X = x_i).$$

The **variance** of a random variable is defined by

$$\sigma^2 = \text{Var}(X) = E\{(X - \mu)^2\}.$$

Example

Let X be a random variable

taking values	taking values
+1 or -1	+10 or -10
with prob. 1/2 each	with prob. 1/2 each
$\mu = E(X) = 0$	$\mu = E(X) = 0$
and	and
$\sigma^2 = \text{Var}(X) = 1$	$\sigma^2 = \text{Var}(X) = 100$

The variance of a random variable describes the spread of the values taken by the random variable.

Notation

We will denote by CAPITAL letters the random variables, and by small letters the values taken by the random variables.

Thus X, Y, Z will stand for random variables, while x, y, z will stand for the values attained by the random variables X, Y, Z respectively.

Examples of random variables

Consider n independent trials where the probability of success in each trial is p and let X denote the total number of successes, then

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k = 0, 1, \dots, n$, $0 \leq p \leq 1$. This is known as Binomial distribution, written as $X \sim B(n, p)$.

$$E(X) = np \text{ and } Var(X) = np(1-p).$$

Consider a random variable X such that

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

for $k = 0, 1, 2, \dots$ and $\lambda > 0$. This is known as Poisson distribution. Here $E(X) = \lambda$ and $Var(X) = \lambda$.

If X has Binomial distribution $B(n, p)$ with large n and small p , then X can be approximated by a Poisson random variable Y with parameter $\lambda = np$, i.e.

$$P(X \leq a) \approx P(Y \leq a)$$

In order to consider random variables that may take any real number or any number in an interval as its value, we need to extend our notion of sample space and events. One difficulty is that we can no longer define probabilities for all subsets of the sample space. We will only note here that the class of events - namely the sets for which the probabilities are defined is large enough.

We also need to add an axiom called **Countable additivity axiom**: If $A_1, A_2, \dots, A_k, \dots$ are pairwise mutually exclusive events then

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i).$$

A real valued function X on a sample space Ω is said to be a random variable if for all real numbers a , the set $\{\omega : X(\omega) \leq a\}$ is an event.

For a random variable X , the function F defined by

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

is called its distribution function. If there exists a function f such that

$$F(x) = \int_{-\infty}^x f(t) dt$$

then f is called the density of X .

Examples of densities:

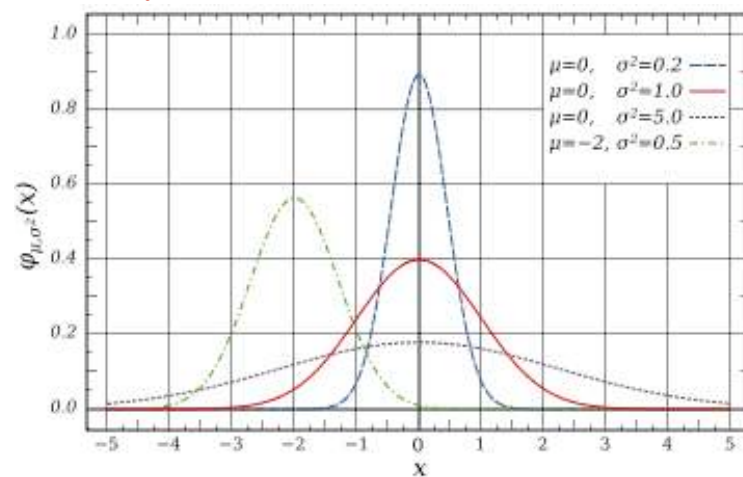
$$f(x) = \begin{cases} \lambda \exp(-\lambda x) & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

This is called exponential density.

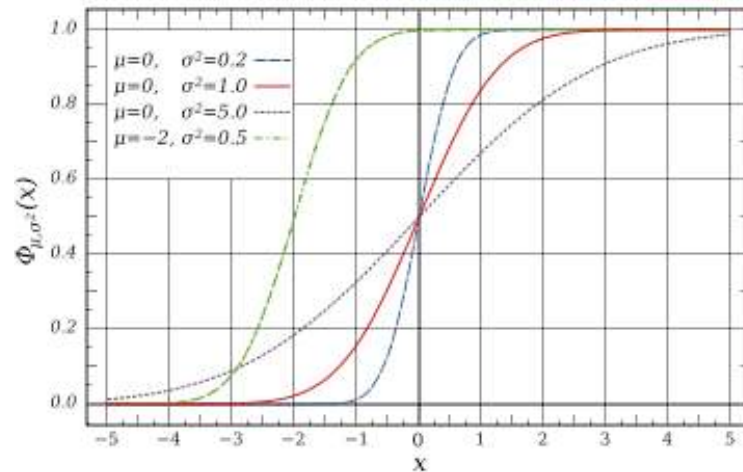
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

This is the Normal density.

Normal density function



Normal density function



A very common density function encountered in astronomy is the globular cluster luminosity function GCLF.

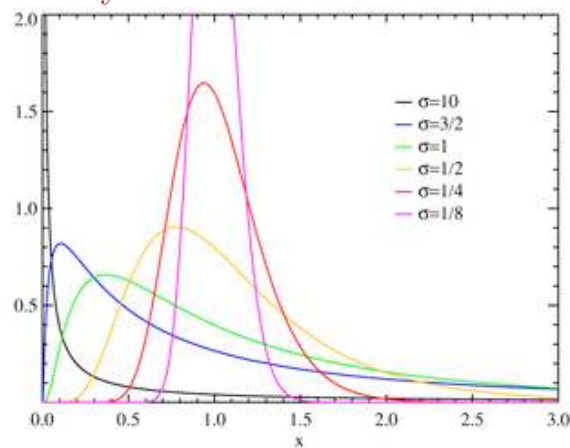
A globular cluster (GC) is a collection of 10⁴-10⁶ ancient stars concentrated into a tight spherical structure structurally distinct from the field population of stars.

The distribution of GC luminosities (i.e. the collective brightness of all of its stars) is known as the globular cluster luminosity function (GCLF).

The shape of this function is said to be **lognormal** i.e.

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left\{-\frac{(\ln(x) - \mu)^2}{2\sigma^2}\right\} \text{ for } x > 0.$$

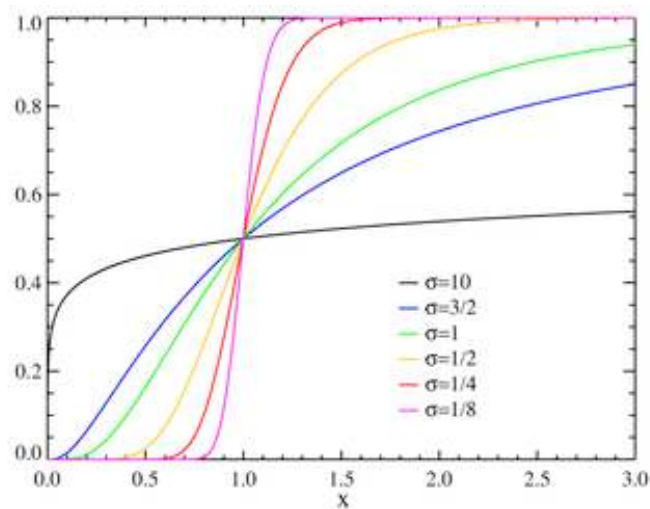
Lognormal density function



The distribution function of this random variable is difficult to compute explicitly.

It may be shown that if X is a normal random variable, then e^X has a log-normal distribution.

Lognormal density function



For a random variable X with density f , the expected value of X , where g is a function is defined by

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

For a random variable X with Normal density

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

$$E(X) = \mu$$

$$Var(X) = E[(X - \mu)^2] = \sigma^2.$$

We write $X \sim N(\mu, \sigma^2)$.

If $X \sim N(0, 1)$ (i.e. the mean is 0 and variance is 1) then we call X a **standard normal random variable** and denote its density function and distribution function as

$$\begin{aligned}\phi(z) &= \frac{1}{\sqrt{2\pi}} e^{(-z^2/2)} \\ \Phi(x) &= \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} e^{(-z^2/2)} dz\end{aligned}$$

The values of $\Phi(x)$ and those of $F(x)$ for other standard distributions are available in various computer spreadsheets.

For a random variable Y with lognormal density

$$\begin{aligned}f(x) &= \frac{1}{x\sigma\sqrt{2\pi}} \exp\left\{-\frac{(\ln(x) - \mu)^2}{2\sigma^2}\right\} \text{ for } x > 0. \\ E(X) &= e^{\mu + (\sigma^2/2)} \\ \text{Var}(X) &= E[(X - \mu)^2] = (e^{\sigma^2} - 1)e^{2\mu + \sigma^2}.\end{aligned}$$

When an experiment is conducted many times, e.g. a coin is tossed a hundred times, we are generating a sequence of random variables. Such a sequence is called a sequence of **i.i.d.** (independent identically distributed) random variables.

Suppose we gamble on the toss of a coin as follows – if **HEAD** appears then you give me 1 Rupee and if **TAIL** appears then you give me -1 Rupee.

So if we play n round of this game we have generated **i.i.d. sequence of random variables** X_1, \dots, X_n where each X_i satisfies

$$X_i = \begin{cases} +1 & \text{with prob. } 1/2 \\ -1 & \text{with prob. } 1/2 \end{cases}$$

Now

$$S_n = X_1 + X_2 + \dots + X_n$$

represents my gain after playing n rounds of this game.

Suppose we play the game n times and observe my gains/losses

OBSERVATION	Probability CHANCE
$S_{10} \leq -2$ i.e. I lost at least 6 out of 10	0.38 moderate
$S_{100} \leq -20$ i.e. I lost at least 60 out of 100	0.03 unlikely
$S_{1000} \leq -200$ i.e. I lost at least 600 out of 1000	1.36^{-10} impossible

OBSERVATION	PROPORTION	Probability
$ S_{10} \leq 1$	$\frac{ S_{10} }{10} \leq 0.1$	0.25
$ S_{100} \leq 8$	$\frac{ S_{100} }{100} \leq 0.08$	0.56
$ S_{1000} \leq 40$	$\frac{ S_{1000} }{1000} \leq 0.04$	0.8

Law of Large Numbers

Suppose X_1, X_2, \dots is a sequence of *i.i.d.* random variables with $E(|X_1|) < \infty$.
Then

$$\bar{X}_n = \sum_{i=1}^n \frac{X_i}{n}$$

converges to $\mu = E(X_1)$: *i.e.* for all $\epsilon > 0$

$$P(|\bar{X}_n - \mu| > \epsilon) \longrightarrow 0.$$

Event	Probability	Normal
$\sqrt{1000} \frac{S_{1000}}{1000} \leq 0$	0.5126	$\Phi(0) = 0.5$
$\sqrt{1000} \frac{S_{1000}}{1000} \leq 1$	0.85	$\Phi(1) = 0.84$
$\sqrt{1000} \frac{ S_{1000} }{1000} \leq 1.64$	0.95	$\Phi(1.64) = 0.95$
$\sqrt{1000} \frac{ S_{1000} }{1000} \leq 1.96$	0.977	$\Phi(1.96) = 0.975$

For the sequence of *i.i.d.* random variables X_1, X_2, \dots with

$$X_i = \begin{cases} +1 & \text{with prob. } 1/2 \\ -1 & \text{with prob. } 1/2 \end{cases}$$

we have for $\bar{X}_n = S_n/n$,

$$P\left\{\sqrt{n}\left(\frac{\bar{X}_n - \mu}{\sigma}\right) \leq x\right\} \longrightarrow \Phi(x)$$

Central Limit Theorem

Suppose X_1, X_2, \dots is a sequence of *i.i.d.* random variables with $E(|X_1|^2) < \infty$. Let $\mu = E(X_1)$ and $\sigma^2 = E[(X_1 - \mu)^2]$. Let

$$\bar{X}_n = \sum_{i=1}^n \frac{X_i}{n}.$$

Then

$$P\left\{\sqrt{n}\left(\frac{\bar{X}_n - \mu}{\sigma}\right) \leq x\right\} \longrightarrow \Phi(x)$$

$X \sim \text{Binomial}(n, p)$, n large. Then

$$P(X \leq a)$$

can be approximated by

$$\Phi\left(\sqrt{n}\left(\frac{a-np}{\sqrt{np(1-p)}}\right)\right)$$

Introduction to R

Arnab Chakraborty

Statistical Consultant

Kolkata

Introduction to R

Hello, R!


R is a free statistical software. It has many uses including

1. performing simple calculations (like a very powerful pocket calculator)
2. making plots (graphs, diagrams etc),
3. analysing data using ready-made statistical tools (*e.g.*, regression),
4. and above all it is a powerful programming language.

We shall acquaint ourselves with the basics of R in this tutorial.

Starting R

First you must have R installed in your computer. Then typically you have to hunt for an

icon like  and double click on it. If everything goes well, you should see a window pop up containing something like this.

```
R : Copyright 2005, The R Foundation for Statistical Computing
Version 2.1.1 (2005-06-20), ISBN 3-900051-07-0
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
```

```
Natural language support but running in an English locale
```

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.
```

```
>
```

The `>` is the R *prompt*. You have to type commands in front of this prompt and press the **Enter** key on your keyboard.

Simple arithmetics

R may be used like a simple calculator. Type the following command in front of the prompt and hit **Enter**.

```
2 + 3
```

Ignore the [1] in the output for the time being. We shall learn its meaning later. Now try

```
2 / 3
```

What about the following? Wait! Don't type these all over again.

```
2 * 3
2 - 3
```



Just hit the ↑ key of your keyboard to *replay* the last line. Now use the ← and → cursor keys and the **Delete** key to make the necessary changes.

Exercise: What does R say to the following?

```
2/0
```

Guess what is going to be the result of

```
2/Inf
```

and of

```
Inf/Inf
```

So now you know about three different types of 'numbers' that R can handle: ordinary numbers, infinities, NaN (Not a Number).

Variables

R can work with variables. For example

```
x = 4
```

assigns the value 4 to the variable x . This assignment occurs silently, so you do not see any visible effect immediately. To see the value of x type

```
x
```

This is a very important thing to remember:



To see the value of the variable just type its name and hit **Enter**.

Let us create a new variable

```
y = -4
```

Exercise: Try the following.

```
x + y
x - 2*y
x^2 + 1/y
```

The caret (^) in the last line denotes power.

Exercise: What happens if you type the following?

```
z-2*x
```

and what about the next line

```
X + Y
```

Well, I should have told you already: R is case sensitive!

Exercise: Can you explain the effect of this?

```
x = 2*x
```

Standard functions

R knows most of the standard functions.

Exercise: Try

```
sin(x)
cos(0)
sin(pi) #pi is a built-in constant
tan(pi/2)
```



The part of a line after # is called a **comment**. It is meant for *U*, the User who use R! R does not care about comments.

Exercise: While you are in the mood of using R as a calculator you may also try

```
exp(1)
log(3)
log(-3)
log(0)
log(x-y)
```

What is the base of the logarithm?

Getting help

R has many many features and it is impossible to keep all its nuances in one's head. So R has an efficient online help system. The next exercise introduces you to this.

Exercise: Suppose that you desperately need logarithm to the base 10. You want to know if R has a ready-made function to compute that. So type

```
?log
```

A new window (the help window) will pop up. Do you find what you need?



Always look up the help of anything that does not seem clear. The technique is to type a question mark followed by the name of the thing you are interested in. All words written like **this** in this tutorial have online help.

Sometimes, you may not know the exact name of the function that you are interested in. Then you can try the **help.search** function.

Exercise: Can R compute the Gamma function? As your first effort try

```
Gamma(2)
```

Oops! Apparently this is not the Gamma function you are looking for. So try

```
help.search("Gamma")
```

This will list all the topics that involve Gamma. After some deliberation you can see that "Special Functions of Mathematics" matches your need most closely. So type

```
?Special
```

Got the information you needed?

Searching for functions with names known only approximately is often frustrating.



Sometimes it is easier to google the internet than perform **help.search**!

Functions

We can type

```
sin(1)
```

to get the value $\sin 1$. Here **sin** is a standard built-in function. R allows us to create new functions of our own. For example, suppose that some computation requires you to find the value of

$$f(x) = x/(1-x)$$

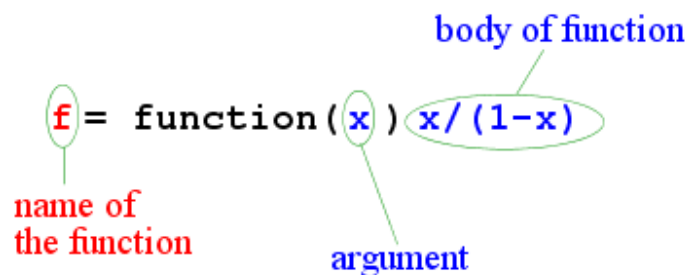
repeatedly. Then we can write function to do this as follows.

```
f = function(x) x/(1-x)
```

Now you may type

```
f(2)
y = 4
f(y)
f(2*y)
```

Here `f` is the name of the function. It can be any name of your choice (as long as it does not conflict with names already existing in R).



Anatomy of an R function

A couple of points are in order here. First, the choice of the name depends completely on you. Second, the name of the argument is also a matter of personal choice. But you must use the same name also inside the body of the function. It is also possible to write functions of more than one variable.

Exercise: Try out the following.

```
g = function(x,y) (x+2*y)/3
g(1,2)
g(2,1)
```

Exercise: Write a function with name `myfun` that computes $x+2*y/3$. Use it to compute $2+2*3/3$.

Vectors

So far R appears little more than a sophisticated calculator. But unlike most calculators it can handle **vectors**, which are basically lists of numbers.

```
x = c(1,2,4,56)
x
```

The **C** function is for concatenating numbers (or variables) into vectors.

Exercise: Try

```
y = c(x, c(-1,5), x)
length(x)
length(y)
```

There are useful methods to create long vectors whose elements are in arithmetic progression:

```
x = 1:20
4:-10
```

If the common difference is not 1 or -1 then we can use the **seq** function

```
y=seq(2,5,0.3)
y
```

Exercise: Try the following

```
1:100
```

Do you see the meaning of the numbers inside the square brackets?

Exercise: How to create the following vector in R?

```
1, 1.1, 1.2, , ... 1.9, 2, 5, 5.2, 5.4, ... 9.8, 10
```

Hint: First make the two parts separately, and then concatenate them.

Working with vectors

Now that we know how to create vectors in R, it is time to use them. There are basically three different types of functions to handle vectors.

1. those that work entrywise
2. those that summarise a vector into a few numbers (like finds the sum of all the numbers)
3. others

Exercise: Most operations that work with numbers act entrywise when applied to vectors. Try this.

```
x = 1:5
x^2
x+1
2*x
sin(x)
exp(sqrt(x))
```

It is very easy to add/subtract/multiply/divide two vectors entry by entry.

Exercise:

```
x = c(1,2,-3,0)
y = c(0,3,4,0)
x+y
x*y
x/y
2*x-3*y
```

Next we meet some functions that summarises a vector into one or two numbers.

Exercise: Try the following and guess the meanings of commands.

```
val = c(2, 1, -4, 4, 56, -4, 2)
sum(val)
mean(val)
min(val)
max(val)
range(val)
```

Exercise: Guess the outcome of

```
which.min(val)
which.max(val)
```

Check your guess with the online help.

Extracting parts of a vector

If x is vector of length 3 then its entries may be accessed as $x[1]$, $x[2]$ and $x[3]$.

```
x = c(2, 4, -1)
x[1]
x[2]+x[3]
i = 3
x[i]
x[i-1]
x[4]
```

Note that the counting starts from 1 and proceeds left-to-right. The quantity inside the square brackets is called the **subscript** or **index**.



It is also possible to access multiple entries of a vector by using a subscript that is itself a vector.

```
x = 3:10
x[1:4]
x[c(2, 4, 1)]
```

Exercise: What is the effect of the following?

```
x = c(10, 3, 4, 1)
ind = c(3, 2, 4, 1) #a permutation of 1, 2, 3, 4
x[ind]
```

This technique is often useful to rearrange a vector.

Exercise: Try the following to find how R interprets **negative subscripts**.

```
x = 3:10
x
x[-1]
x[-c(1, 3)]
```

Subscripting allows us to find one or more entries in a vector if we know the position(s) in the vector. There is a different (and very useful) form of subscripting that allows us to extract entries with some given property.

```
x = c(100,2,200,4)
x[x>50]
```

The second line extracts all the entries in `x` that exceed 50. There are some nifty things that we can achieve using this kind of subscripting. To find the sum of all entries exceeding 50 we can use

```
sum(x[x>50])
```

How does this work? If you type

```
x>50
```

you will get a vector of **TRUE**s and **FALSE**s. A **TRUE** stands for a case where the entry exceeds 50. When such a True-False vector is used as the subscript only the entries corresponding to the **TRUE**s are retained. Even that is not all. Internally a **TRUE** is basically a 1, while a **FALSE** is a 0. So if you type

```
sum(x>50)
```

you will get the number of entries exceeding 50.



The number of entries satisfying some given property (like "less than 4") may be found easily like

```
sum(x<4)
```

Exercise: If

```
val = c(1,30,10,24,24,30,10,45)
```

then what will be the result of the following?

```
sum(val >= 10 & val <= 40)
sum(val > 40 | val < 10) # | means "OR"
sum(val == 30) # == means "equal to"
sum(val != 24) # != means "not equal to"
```

Be careful with `==`. It is different from `=`. The former means comparing for equality, while the latter means assignment of a value to a variable.

Exercise: What does

```
mean(x>50)
```

denote?

Exercise: Try and interpret the results of the following.

```
x = c(100, 2, 200, 4)
sum(x>=4)
mean(x!=2)
x==100
```

Sorting

```
x = c(2, 3, 4, 5, 3, 1)
y = sort(x)
y #sorted
x #unchanged
```

Exercise: Look up the help of the `sort` function to find out how to sort in *decreasing* order.

Sometimes we need to order one vector *according to* another vector.

```
x = c(2, 3, 4, 5, 3, 1)
y = c(3, 4, 1, 3, 8, 9)
ord = order(x)
ord
```

Notice that `ord[1]` is the *position* of the smallest number, `ord[2]` is the position of the next smallest number, and so on.

```
x[ord] #same as sort(x)
y[ord] #y sorted according to x
```

Matrices

R has no direct way to create an arbitrary matrix. You have to first list all the entries of the matrix as a single vector (an m by n matrix will need a vector of length mn) and then fold the vector into a matrix. To create

```
  1    2
  3    4
```

we first list the entries *column by column* to get 1, 3, 2, 4.

To create the matrix in R:

```
A = matrix(c(1, 3, 2, 4), nrow=2)
A
```

The `nrow=2` command tells R that the matrix has 2 rows (then R can compute the number of columns by dividing the length of the vector by `nrow`.) You could have also typed:

```
A <- matrix(c(1,3,2,4),ncol=2) #<- is same as =
A
```

to get the same effect. Notice that R folds a vector into a matrix *column by column*. Sometimes, however, we may need to fold *row by row* :

```
A = matrix(c(1,3,2,4),nrow=2,byrow=T)
```

The **T** is same as **TRUE**.

Exercise: Matrix operations in R are more or less straight forward. Try the following.

```
A = matrix(c(1,3,2,4),ncol=2)
B = matrix(2:7,nrow=2)
C = matrix(5:2,ncol=2)
dim(B) #dimension
nrow(B)
ncol(B)
A+C
A-C
A%%C #matrix multiplication
A*C #entrywise multiplication
A%%B
t(B)
```

Subscripting a matrix is done much like subscripting a vector, except that for a matrix we need two subscripts. To see the (1,2)-th entry (*i.e.*, the entry in row 1 and column 2) of A type

```
A[1,2]
```

Exercise: Try out the following commands to find what they do.

```
A[1,]
B[1,c(2,3)]
B[,-1]
```

Working with rows and columns

Consider the following.

```
A = matrix(c(1,3,2,4),ncol=2)
sin(A)
```

Here the **sin** function applies *entrywise*. Now suppose that we want to find the sum of each column. So we want to apply the sum function *columnwise*. We achieve this by using the **apply** function like this:

```
apply(A, 2, sum)
```

The 2 above means *columnwise*. If we need to find the *rowwise* means we can use

```
apply(A, 1, mean)
```

Lists

Vectors and matrices in R are two ways to work with a collection of objects. **Lists** provide a third method. Unlike a vector or a matrix a list can hold different kinds of objects. Thus, one entry in a list may be a number, while the next is a matrix, while a third is a character string (like "Hello R!"). Lists are useful to store different pieces of information about some common entity. The following list, for example, stores details about a student.

```
x = list(name="Rama", nationality="Indian", height=5.5,
marks=c(95,45,80))
```

We can now extract the different fields of `x` as

```
names(x)
x$name
x$hei #abbrevs are OK
x$marks
x$m[2]
x$na #oops!
```

In the coming tutorials we shall never need to make a list ourselves. But the statistical functions of R usually return the result in the form of lists. So we must know how to unpack a list using the **\$** symbol as above.

To see the online help about symbols like **\$** type

`?"$"`

Notice the double quotes surrounding the symbol.

Let us see an example of this. Suppose we want to write a function that finds the length, total and mean of a vector. Since the function is returning three different pieces of information we should use lists as follows.

```
f = function(x) list(len=length(x), total=sum(x), mean=mean(x))
```

Now we can use it like this:

```
dat = 1:10
result = f(dat)
names(result)
result$len
result$tot
```

```
result$mean
```

Doing statistics with R

Now that we know R to some extent it is time to put our knowledge to perform some statistics using R. There are basically three ways to do this.

1. Doing elementary statistical summarisation or plotting of data
2. Using R as a calculator to compute some formula obtained from some statistics text.
3. Using the sophisticated statistical tools built into R.

In this first tutorial we shall content ourselves with the first of these three. But first we need to get our data set inside R.

Loading a data set into R

We shall consider part of a data set given in

Distance to the Large Magellanic Cloud: The RR Lyrae Stars Gisella Clementini, Raffaele Gratton, Angela Bragaglia, Eugenio Carretta, Luca Di Fabrizio, and Marcella Maio *Astronomical Journal* 125, 1309-1329 (2003).

We have slightly doctored the data file to make it compatible with R. The file is called `LMC.dat` and resides in some folder `F:\astro`, say. The data set has two columns with the headings `Method`, `Dist` and `Err`. Here are the first few lines of the file:

```
Method           Dist           Err
"Cepheids: trig. paral."  18.70         0.16
"Cepheids: MS fitting"    18.55         0.06
"Cepheids: B-W"          18.55         0.10
```

There are various ways to load the data set. One is to use

```
LMC = read.table("F:/astro/LMC.dat", header=T)
```

Note the use of forward slash (/) even if you are working in Windows. Also the `header=T` tells that the first line of the data file gives the names of the columns. Here we have used the *absolute path* of the data file. In Unix the absolute path starts with a forward slash (/).

```
dim(LMC)
names(LMC)
LMC
```

This object `LMC` is like a matrix (more precisely it is called a **data frame**). Each column stores the values of one variable, and each row stores a case. Its main difference with a matrix is that different columns can hold different types of data (for example, the `Method` column stores character strings, while the other two columns hold numbers). Otherwise, a data frame is really like a matrix. We can find the mean of the `Dist` variable like this

```
mean(LMC[,2])
mean(LMC[, "Dist"])
```

Note that each column of the `LMC` matrix is a variable, so it is tempting to write

```
mean(Dist)
```

but this will not work, since `Dist` is inside `LMC`. We can "bring it out" by the command

```
attach(LMC)
```

Now the command

```
mean(Dist)
```

works perfectly. All the values of the `Dist` variable are different measurements of the same distance. So it is only natural to use the average as an estimate of the true distance. But the `Err` variable tells us that not all the measurements are equally reliable. So a better estimate might be a weighted mean, where the weights are inversely proportional to the errors. We can use R as a calculator to directly implement this formula:

```
sum(Dist/Err)/sum(1/Err)
```

or you may want to be a bit more explicit

```
wt = 1/Err
sum(Dist*wt)/sum(wt)
```

Actually there is a smarter way than both of these.

```
weighted.mean(Dist, 1/Err)
```

Script files

So far we are using R *interactively* where we type commands at the prompt and the R executes a line before we type the next line. But sometimes we may want to submit many lines of commands to R at a single go. Then we need to use scripts.



Use script files to save frequently used command sequences. Script files are also useful for replaying an analysis at a later date.

A script file in R is a text file containing R commands (much as you would type them at the prompt). As an example, open a text editor (*e.g.*, notepad in Windows, or gedit in Linux). Avoid fancy editors like MSWord. Create a file called, say, `test.r` containing the following lines.

```
x = seq(0,10,0.1)
y = sin(x)
plot(x,y,ty="l") #guess what this line does!
```

Save the file in some folder (say `F:/astro`). In order to make R execute this script type

```
source("F:/astro/test.r")
```

If your script has any mistake in it then R will produce error messages at this point. Otherwise, it will execute your script.

The variables `x` and `y` created inside the command file are available for use from the prompt now. For example, you can check the value of `x` by simply typing its name at the prompt.

```
x
```

Commands inside a script file are executed pretty much like commands typed at the prompt. One important difference is that in order to **print** the value of a variable `x` on the screen you have to write

```
print(x)
```

Merely writing

```
x
```

on a line by itself will not do inside a script file.



Printing results of the intermediate steps using **print** from inside a script file is a good way to debug R scripts.