Probability Calculator

Ali Khudiyev

September 2020

1 About

This program is to calculate probabilities according to the given distribution and parameters. It does the calculations according to the given distribution table instead of doing calculations from scratch according to given parameters of a chosen distribution. So, you are not going to get a valid probability if you provide the program with arguments which do not exist in a chosen distribution table. To check what each distribution table looks like you can find them in the directory called **distributions** located in the project directory. There are 5 distributions available:

- Normal distribution
- Student's t distribution
- Chi square distribution
- Binomial distribution
- Fisher distribution

2 Manual

The program takes a couple of arguments:

1. Distribution type: Normal/Binomial/Chi Square/Student's T/F
2. Z-value [for Normal distribution] or (Row, Column) [for Student's t and Chi square distributions] or (Arg, Row, Column) of the given probability distribution table

If the chosen distribution is Binomial or Fisher then (Arg, Row, Column) are used as arguments. In binomial distribution, $F(k, n, p) = F(\text{Row}, \text{Arg}, \text{Column})$ and in fisher distribution, $F(x; d_1, d_2) = F(x, \text{Column}, \text{Row}) = \text{Arg}$.

2.1 How to run

Download the project from here. After downloading the project, open the project directory in terminal and then type:

```
cd build; cmake .. && make
```

Now you can run the program by typing the following command:

```
./pc [distribution type]
```

For example,

```
./pc 1
Enter row and column arguments: 4 0.8
Probability: 0.270730
```

There are 5 supported distribution types and here are their "code numbers" shown below.

Distribution type	Code number
Normal	0
Student's t	1
Chi square	2
Binomial	3
Fisher	4

Table 1: Distribution types and code numbers

To generate *Student's t*, *Chi square* and *Fisher* distributions type **./pc -g**.

3 Technical details

Let $f(x)$ be the probability density function and $F(x)$ be the cumulative distribution function. Then the following equality holds:

$$F(x) = \int_{x_0}^{x} f(x)dx$$

x_0 may differ depending on the type of distribution. It can be 0 or $lim_{n\to 0^+}$ n for *fisher* distribution, it can be $-\infty$ for *normal* distribution and so on.

Let a distribution be denoted as $D(\bar{v})$ such that $X \sim D(\bar{v})$ where D is the name of distribution and $\bar{v}$ is the argument vector.[1] To generate any distribution table given by $D(\bar{v})$, it is enough to solve the equation shown below:

$$\int_{x_0}^{x} f(x;\bar{v})dx = P$$

$f(x)$ is a probability density function and therefore, the equation can be rewritten as shown below:

$$F(x;\bar{v}) = P \ , \quad \text{where F(x) is cumulative distribution function}$$

P and $\bar{v}$ are the arguments given as the *row*, *column* elements and the third argument or *arg*(sometimes) of a table, and x is a table element found with the help of those arguments. After fixing up those arguments(P and $\bar{v}$), x can be approximated by simply iterating values through the domain until $F(x;\bar{v}) \approx P$. The amount of deviation(Δ) in the final result(approximated x or x') that such approximation may end up with depends on the iteration factor dx.

$$0 \leq \Delta = |x' - x| < \text{dx}$$

[1]For example, in fisher distribution it is written as $X \sim F(d_1, d_2)$ where $\bar{v} = (d_1, d_2)$.

The implementation looks like this in theory:

area $= 0$
$x = x_0$
while $area < P$ **do**
  area $\leftarrow$ area $+ f(x; \bar{v}) \times$ dx
  $x \leftarrow x +$ dx
end while

However, if $x_0 = -\infty$ it would not be possible to run the algorithm shown above. That is why the actual implementation differs a bit in practice. For example, if *Student's t* distribution is chosen then it would be the case($x_0 = -\infty$) and due to that, it would be better to initialize x to 0 and P to $\frac{1-P}{2}$.

3.1 Requirements

It is better to have the library called OpenMP if you want to generate a distribution table faster. Since generating several distribution tables can take a bit longer *OMP* library takes care of parallelization of the process in order to reduce the run-time. In fact, the program run 30% faster than without parallelization(without parallelization: $10s$, with parallelization: $7s$). To use *OMP* you also have to have gcc compiler.

Requirements	**Version**
gcc	8+
OpenMP	-

Note: Do not forget to compile the project with **gcc** (not **clang**)!