**Proceedings of the Third International DERIVE/TI-92 Conference**

# A Computer Proof of the Central Limit Theorem

**Peter Mitic**
**Positive Corporation Ltd.,**
**Medstead, Hampshire GU34 5EW,**
**England**
**peter_mitic@compuserve.com**

## Abstract

It is uncommon to apply computer algebra software to proof, especially for topics which are rarely touched by computer algebra. In order to illustrate the efficacy of symbolic computation in these respects, DERIVE is used in a statistical context to prove the Central Limit Theorem and some advantages and disadvantages of using the techniques outlined here are assessed. The proof is illustrated by simulating random experiments.

## Introduction

The topic of proof is fundamental to mathematics, yet computer algebra software has had little impact on proof, despite having become widespread in perform actual computations. Proof and algebraic computation are intimately related, and this paper illustrates how a progression can be made from the latter to the former. Three principal points are illustrated.

1. Proof can be done as an inter-connected and logical sequence of algebraic operations, which has to be set up correctly.
2. A proof can be illustrated heuristically, which adds meaning to its result.
3. Proof is applicable, and I is beneficial to see it, outside the confines of algebraic computation alone.

In a previous discussion, (Mitic and Thomas 1995), I assessed the advantages and disadvantages of using DERIVE to perform algebraic manipulations in a derivation of a Normal distribution from a binomial distribution. DERIVE was used then, and is used here, because it is widely available, but is hard to program if particular constructs are required. Some of these problems are solved, allowing pre-programmed units to be used in the heuristic phase of this discussion. The Central Limit Theorem provides a good vehicle for this because it has a significant ‘set up’ phase, is simplified by using symbolic computation in the ‘active’ part of the proof, and makes good use of an integrated mathematical analysis environment in which computation, graphics and proof can all be done. DERIVE is useful because of its simple interface and speed of operation, but there are programming complications, which are eased using the more sophisticated programming capabilities of, for example, Maple and Mathematica.

## Simulations to Illustrate the Central Limit Theorem

As a preliminary to proving the Central Limit Theorem, the theorem is illustrated by some sampling experiments. This provides some motivation for the proof itself, since without seeing the theorem 'in action', it is harder to appreciate its impact. The simulations are intended to show that the distribution of sample means 'looks' Normal, and is independent of the background population. Consequently, a non-Normal background distribution is illustrated here. There is a proviso that the sample size is 'sufficiently large', and this term can be partially quantified with sufficient experimentation.

As an example, random numbers which are exponentially distributed with probability density function $f(x) = \begin{cases} 2e^{-2x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$ are generated. DERIVE's random number generator produces pseudo-random numbers which are uniformly distributed, and they must be transformed so that they can be regarded as having originated from a non-uniform distribution. (See, for example, Mitrani 1982). The required transformation for uniformly distributed data is $x \to \frac{\log(1-x)}{-2}$; $x \in [0,1)$. The DERIVE code to generate 10 such random numbers is:

```
APPROX(uniform01 := VECTOR(RANDOM(1), i, 1, 10), 6)

[0.835327,0.443197,0.797847,0.529578,0.924305,0.00408613,0.97003
,0.667271,0.220640,0.07573]

            LOG(1 - x)
F(x) := - ― ― ― ― ― ― ― ― ― ― ―

                2

APPROX(expran := VECTOR(F(x), x, uniform01))

[0.519000,0.0756450,0.632217,1.11895,0.282906,0.160367,0.111106,
1.23094,0.246345,0.00918248]
```

Many other background distributions can be simulated in this way, provided that the distribution function concerned is invertible. This transformation is used to generate means of samples of various sizes.

## Frequency Counts and Graphs

It proved to be very difficult to produce grouped frequency counts for a vector of random data, as generated above, despite the simplicity of the problem. For the continuous data above, we might wish to classify into bins 0–0.1, 0.1–0.2, 0.2–0.3 etc., with a maximum cut-off. To do this, it would be useful to implement a construct similar to the SWITCH statement in C directly. This cannot be done because DERIVE does not support the

required memory variables external to procedures. Instead, a functional programming solution must be adopted. A 'brute force' frequency counter is something like the following, in which 'bin' boundaries are explicit numbers and *v* is the vector which we wish to analyse.

```
FREQUENCYCOUNTER1(v) :=
SUM([IF(x < .25),  IF(.25 <= x < .5),  IF(.5 <= x < .75),
      IF(.75 <= x < 1), IF(1 <= x < 1.25), IF(1.25 <= x)], x, v)
```

In order to increase the usefulness of FREQUENCYCOUNTER1, we note that constructs such as $IF(\alpha \le x \le \beta)$ occur repeatedly, prepended and appended by terms of the form $IF(x < \alpha)$ and $IF(\beta \le x)$ respectively. In the following function, the bin boundaries are held in a vector *b* and VECTOR is used to implement the repeated string of IF constructs.

```
FREQUENCYCOUNTER2(v,b):=
SUM([IF(x<ELEMENT(b,1)),
VECTOR(IF(ELEMENT(b,i)<=x<=ELEMENT(b,i+1)),i,1,DIMENSION(b)-1),
       IF(ELEMENT(b,DIMENSION(b))<=x)],x,v)
```

The result is a nested vector of the form [$e_1$, [$e_2$, $e_3$, ... $e_{n-1}$], $e_n$], which is less useful than the unnested form [$e_1$, $e_2$, $e_3$, ... $e_{n-1}$, $e_n$]. This nested form may be 'flattened' by applying a function FLATTEN, which removes the inner brackets by appending the third and prepending the first element of *v* to the second element of *v*.

```
FLATTEN(v):=APPEND([ELEMENT(v,1)],APPEND(ELEMENT(v,2),
[ELEMENT(v,3)]))
```

Combining FREQUENCYCOUNTER2 and FLATTEN, we can define a function FREQUENCYCOUNTER as below.

```
FREQUENCYCOUNTER(v,b):= FLATTEN(FREQUENCYCOUNTER2(v,b))
```

Because this coding is extremely tricky, it is more appropriate to simply use it, having loaded the necessary code in an .MTH file. In this respect it is no different to any other .MTH file, and might usefully form part of a Descriptive Statistics utility package. The DERIVE session below shows how these functions are used on the random number data generated already, and is a useful illustration of automated numerical manipulation.

```
ran  :=  [0.519000,  0.0756450,  0.632217,  1.11895,  0.282906,
0.160367, 0.111106, 1.23094,    0.246345, 0.00918248]

bins := [0.25, 0.5, 0.75, 1, 1.25]

APPROX(FREQUENCYCOUNTER(ran, bins))

[5, 1, 2, 0, 2, 0]
```

Histograms can be produced from such frequency counts and bin boundaries, using code similar to that of Etchells (Etchells 1992). Figure 1(a) shows a histogram which resulted

from generating 200 means of samples of size 3. Figure 1(b) shows a similar frequency distributions based on samples of size 25. A frequency distribution for the background population is shown in Figure 1(c). All parts of Figure 1 are shown to the same scale.

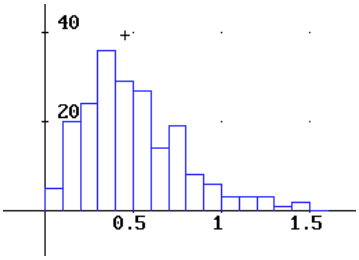


**Figure 1(a)**

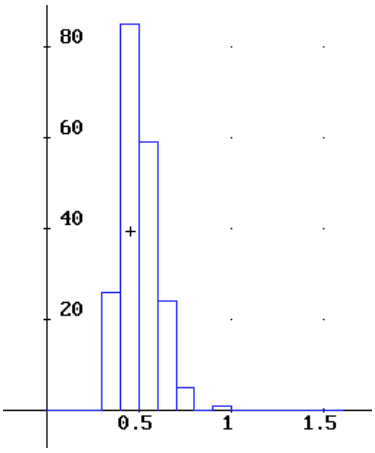


**Figure 1(b)**

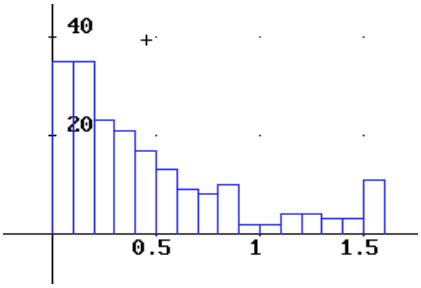


**Figure 1(c)**

The contrast between the form of the (exponential) background distribution and the Normal-like sampling distributions is readily apparent, despite the small sample size of 3. It is easy to verify that the same type of sampling distribution profiles also result from other choices of background distribution and sample sizes, and that they look more Normal as the sample size is increased. This, and similar experiments provide the heuristic motivation behind the proof of the theorem.

The mean, *m*, and standard deviation, *s*, for each of the above, and additional, trials are recorded in Table 1. It is seen that as the sample size, *n*, increases, the form of the

distribution approaches that of a Normal distribution, and clustering about the mean becomes more pronounced. In all cases, the sampling mean is close to the background population mean, 0.5. The standard deviations correspond reasonably well, especially for larger values of $n$, to the theoretical result, $\frac{1}{2\sqrt{n}}$. It is important to stress that these simulations only illustrate a proof, and that a rigorous analysis is required to advance further.

**Table 1**

| $n$ | $m$ | $s$ | $\frac{1}{2\sqrt{n}}$ |
|---|---|---|---|
| 3 | .498 | .282 | .289 |
| 10 | .507 | .173 | .158 |
| 25 | .500 | .097 | .100 |
| 40 | .495 | .077 | .079 |
| 50 | .500 | .074 | .071 |

## Simulations in Other Software

The simulations thus developed is static: we cannot see the a real-time development of the histograms of the background and sampling distributions. This is desirable because seeing profiles develop as sampling proceeds reinforces the conceptual background of the Central Limit Theorem. In (Mitic and Thomas 95) I discussed several alternative software packages for showing this type of simulation. In this context I developed a dedicated Central Limit Theorem simulator in Visual Basic (version 3), in which dynamic simulation of sampling distributions was realised for a variety of continuous and discrete distributions. Running these simulations shows the sampling distribution clearly deviating from its originating background distribution (which retains its 'shape'), and tending to a the 'shape' of a Normal distribution. The empirical relationships $m_s = m$ and $s_s = s/\sqrt{n}$ can also be verified. Behind the scenes are diverse techniques for generating random numbers distributions. Some (Monte Carlo simulations in particular) run too slowly in the VB interpreted environment, but the same techniques implemented in a compiled C++ environment run much faster. An implementation in Excel VBA is very easy to program but the desired dynamic aspect is a by-product and its position is unstable on the screen. Minitab (version 9) proved to have inadequate programming facilities compared to Excel. Of course, proof is not possible in any of these implementations. A Mathematica implementation may be found in (Mitic 96). Programming the simulations is eased considerably by the availability of memory variables.

## Formulation of the Theorem

These sampling experiments give a visual impression of the impact of the Central Limit Theorem, and we can use them to make conjectures about the form of the sampling distribution, its mean and its variance. These provide the rationale for a formal statement of the Central Limit Theorem. Mendenhall (Mendenhall 1986) gives a particularly clear version.

Let $Y_1$, $Y_2$, ..., $Y_n$ be independent and identically distributed random variables, each with mean $m$ and variance $s^2$. Then $\overline{Y} = \frac{1}{n}\sum_{r=1}^{n} Y_r$ is approximately Normally distributed with mean $m$ and variance $\frac{s^2}{n}$ for large $n$.

This is an example of a proof in which the preliminaries are considerable in any case, but the proof is quick if a symbolic computation package is used. It is dependent on a polynomial approximation for a moment generating function, and this causes some subtle problems with DERIVE. The proof depends on simplifying the expression $\lim_{n\to\infty} MZ1(t)$, below, and recognising the result as the moment generating function of a N(0, 1) distribution. The moment generating function of a discrete random variable $X$ that takes values between $a$ and $b$ inclusive is the sum $M(t) = \sum_{a}^{b} e^{xt} P(X = x)$. For a continuous random variable $X$ with probability density function $f(x)$, x∈[$a$,$b$], the moment generating function is $M(t) = \int_{a}^{b} e^{tx} f(x)\mathrm{d}x$. $M(t)$ can then be expressed in the form:

$$M(t) = 1 + t\left[\frac{\mathrm{d}M(t)}{\mathrm{d}t}\right]_{t=0} + \frac{t^2}{2!}\left[\frac{\mathrm{d}^2 M(t)}{\mathrm{d}t^2}\right]_{t=0} + O\left(t^3\right) = 1 + \mu t + \left(\sigma^2 + \mu^2\right)\frac{t^2}{2!} + O\left(t^3\right).$$

The proof is then set up as follows.

Let $Z_i = (Y_i - m)/s$, so $\mu = \overline{Z}_i = 0$ and $\sigma^2 = \mathrm{var}(Z_i) = 1$.

The moment generating function of each $Z_i$ can then be written as $MZ(t) = 1 + \frac{t^2}{2!} + \frac{t^3}{3!}R(t)$ where $R(t)$ is a power series in $t$ which comprises moments of $Z_i$ of degree 3 or more. This is the starting point for the DERIVE session, and we can use the non-explicit function $r(t)$, exactly as in a paper-and pen proof. We then define $U_n$:

$$U_n = \frac{\overline{Y} - \mu}{\sigma/\sqrt{n}} = \frac{\sqrt{n}}{n}\left(\frac{\sum Y_i - n\mu}{\sigma}\right) = \sum \frac{Z_i}{\sqrt{n}}$$

Since the $Y_i$ are independent, so are the $Z_i$. Hence, the moment generating function of $U_n$ is the product of the moment generating functions of the $Z_i/\sqrt{n}$ (the expression $\lim_{n\to\infty} MZ1(t)$, below). This provides the penultimate step of the DERIVE session. The last expression is the moment generating function of a Normal(0,1) random variable, and this result is established as a subsidiary result. Given that the moment generating function

for any given random variable is unique, this shows that $U_n$ has a Normal(0, 1) distribution. Hence, the non-standardised random variable, $\bar{Y}$, has a Normal($m$, $s^2/n$) distribution. DERIVE is of great value here in obtaining the limit. Without simplification of this step in the proof, enough algebra is involved to detract from the overall thrust of the proof.

## DERIVE Implementation

The following code shows the above formulation and implementation.

```
R(t) :=

                 2
                t       3
MZ(t) := 1 + — — — —  + t ·R(t)

                2


MZ1(t) :=  lim   MZ(t)

           t~t/‹n

           n
lim MZ1(t)
n~–

  2
 t /2
ê
```

The DERIVE code below establishes the required result for the moment generating function of a Normal(0,1) random variable. The need to explicitly declare a range for $t$ is useful because it is only too easy to ignore this type of condition in a pen-and -paper proof, even though it is essential for the proof to be valid.

```
–
^          2
¦      - x /2
¦     ê          t·x
¦   — — — — — — — — ·ê    dx

‰      ‹(2·¹)
 --

t :- Real (0, –)

  2
 t /2
ê
```

The method of coding the substitution of $t$ by $t/\sqrt{n}$ ( $MZ1(t) := \lim_{t \to \frac{t}{\sqrt{n}}} MZ(t)$ ) in the above code should be noted. It works here because continuity is not violated, but will not work if target functions are not continuous. Substitutions have to be done in this way using DERIVE, and it can give the impression that such substitutions always work.

Although this proof works, it depends on the power series *R(t)*, upon which no conditions need be placed in order to obtain the correct result. This can cause further problems. The method fails if $t^3$ *R(t)* in *MZ(t)* is replaced by an apparently simpler, and more explicit, term, *R1*(*t*). The simplification of the revised expression $\lim_{n \to \infty} MZ1(t)$ cannot then be done.

```
R1(t) :=

    •     •  t ,2          ,n
    ¦     ¦— — — — ¦          ¦
    ¦     • ‹n f            ¦
lim ¦1 + — — — — — — — — —  + R1(t)¦

n~– •        2             f

?
```

Furthermore, if the *R*(*t*) term is set to zero we still recover a correct result (as below). However, the heavy truncation involved makes the result somewhat dubious.

```
    •     •  t ,2 ,n
    ¦     ¦— — — —¦  ¦
    ¦     • ‹n f  ¦
lim ¦1 + — — — — — — — — ¦

n~– •        2    f

  2
 t /2
ê
```

These results indicate that great care must be taken when employing DERIVE, or any other symbolic manipulator, in a proof such as this.

Unfortunately, this method fails completely in DERIVE version 2: expressions with the arbitrary power series *R*(*t*) simplify to an incorrect result, 0. In this case it is necessary to approximate *R*(*t*) by a linear polynomial $a + bt$. To justify this we use Taylor's Formula with Remainder, which Apostol (Apostol 1957) presents as an extension to the First Mean Value Theorem. There exists *s* in (0,*t*) such that $R(t) = R(0) + t\,R'(s)$ for *s* in (0, *t*). Then

set $a = R(0)$ and $b = R'(s)$. The expression for $MZ(t)$ is then $MZ(t) = 1 + \frac{t^2}{2} + (a + bt)t^3$ and the method proceeds as before.

## Momement Generating Functions: Some Warnings

Being able to calculate the moment generating function of a Normal(0,1) random variable, as in the previous section, turns out to be something of a luxury. This cannot be done for many other distributions as DERIVE cannot simplify defining expressions, as given by the sums or integrals discussed previously, for their moment generating functions. The same problem sometimes occurs when attempting to find means and variances directly from probability density functions. However, once a result for the moment generating function, $M(t)$, of a random variable $X$ has been found, it is easy to use it to compute the mean and variance of $X$, using the formulae $\mu = M'(0)$, $\sigma^2 = M''(0) - \mu^2$. These can be defined in DERIVE in a generic way as follows.

```
                    d
MEAN(mgf, t) := lim — —  mgf
                t~0 dt

                •     •d ,2    ,                  2
VARIANCE(mgf, t) := ¦lim ¦— — ¦  mgf¦ - MEAN(mgf, t)
                •t~0 •dtf     f
```

Instantiating an actual moment generating function then yields the required results. For example, using the continuous $\chi^2(n)$ random variable, which has moment generating function $(1-2t)^{-n/2}$, we obtain $\mu = n$ and $\sigma^2 = 2n$.

```
                              - n/2
MGFCHI2(t, n) := (1 - 2·t)

MEAN(MGFCHI2(t, n), t)

n

VARIANCE(MGFCHI2(t, n), t)

2·n
```

Similarly, the discrete Geometric($p$) random variable has moment generating function $\frac{pe^t}{1-(1-p)e^t}$, from which we obtain $\mu = 1/p$ and $\sigma^2 = (1-p)/p^2$.

```
                             t
                          p·ê
MGFGEOM(t, p) := — — — — — — — — — — — — — — —
```

$$1 - (1 - p)\cdot e^{t}$$

```
MEAN(MGFGEOM(t, p), t)

 1
— — —
 p

VARIANCE(MGFGEOM(t, p), t)

 1 - p
— — — — — — —
    2
   p
```

## Conclusion

A convenient user interface made it easy to perform tedious algebraic manipulations and construct a relatively simple proof. Using DERIVE is not without its problems and in some cases, careful prior preparation must be done by hand because of theoretical considerations which DERIVE cannot cope with. Building theory from scratch, which has to be done if a computer algebra package is used, is advantageous because it does not encourage mere use (or misuse!) of techniques without understanding. Programming in DERIVE would be easier if memory variables were available, and I encourage the development of this feature.

A number of other tasks are performed faster and with simpler programming by other packages. In particular, dedicated statistical packages and spreadsheets can generate random numbers, transform them, classify them and graph the results very quickly. The aim here was to achieve more than that: algebraic proof and the flexibility to simulate trials from an (almost) arbitrary probability distribution.

## REFERENCES

**Etchells, T.** (1992) Investigating Probability Distributions with DERIVE, In Teaching Mathematics with DERIVE (J Böhm, ed.) Chartwell-Bratt

**Mitic, P. and Thomas, P.** (1995) From the Binomial to the Normal: A Computerised Proof, In *Maths and Stats*, Nov 1995, CMI Birmingham.

**Mitic, P.** (1995) Sampling Distributions for Random Data, In *Mathematica in Education and Research*, **4**, 3. (P.Wellin, ed.). Telos.

**Mitic, P. and Thomas, P.** (1996) The Central Limit Theorem - Visualised, In *Teaching Mathematics and its Applications*. **15**, 2. OUP.

**Mitrani, L.** (1982) Simulation techniques for discrete event systems. CUP.

**Mendenhall, W., Schaeffer, R. and Wackerly, D.** (1986) Mathematical Statistics with Applications. PWS, Boston.

**Apostol, T.** (1957) Mathematical Analysis Addison-Wesley.