# Simple Recurrence Relations Between Cumulants and Moments of Univariate Distributions

Grant B. Bunker

June 2, 2024

Cumulants[1, 2] can be viewed as specific nonlinear functions of the power moments of a probability or other distribution[1]. Here we confine our attention to cumulants of univariate distributions. Cumulants possess unusual and very useful properties, among them: the cumulants $C_n$ of order $n > 1$ are origin-independent, and the cumulants of convoluted distributions are additive. A commonplace example of this use, sometimes known as "adding in quadrature", is that of adding the variances (second cumulants) of convoluted Gaussian distributions.

In 1983, in a condensed-matter physics paper, we derived[3] very simple recurrence relations between cumulants and power moments, ones that appear not to have been discovered elsewhere. Understandably, these relations, coming from outside the mathematics and statistics literature, have not received much attention apart from its original application. Their derivation involved the use of un-normalized probability distributions, which conventionally are not much used in statistics, and differentiation with respect to an auxiliary variable. Our recurrence relations may have been overlooked for those reasons. However, they are simple, useful, and computationally reasonably efficient, so we think they should be better known – whence this brief note.

Here we wish to demonstrate their utility, and give simple implementations using Mathematica[4]. Using our coupled recurrence relations, in about a minute of computation time, we can generate all expressions up to $n = 50$ that relate the nth order cumulants to all moments with $m \leq n$, or alternatively, relate all nth moments up to $n = 50$ to all cumulants with $m \leq n$. $C_{50}$ has 204226 terms, and the total number of computed terms in the hierarchy is nearly 1.3 million (1295972). These are posted as compressed zip files at `http://gbxafs.iit.edu/cumulants/`.

Calculating all such relations up to $n = 60$ takes about five minutes on our laptop computer, and the data structure containing those relations takes about 5.6 GB to store, with close to a million (966467) terms. It is straightforward to continue the recurrence for

[1]Some online resources can be found in `http://www.scholarpedia.org/article/Cumulants`, `https://en.wikipedia.org/wiki/Cumulant`, and `https://www.stat.uchicago.edu/~pmcc/courses/stat306/2013/cumulants.pdf`.

the $n+1$th cumulant and beyond, knowing only the expression for the $n$th in terms of the internal representation in terms of un-normalized moments.

It is also helpful to have expressions for the moments in terms of the cumulants. For low orders, as is typically done, the equations can be inverted to find those expressions, but its rapidly becomes impractical to do so higher orders. The execution time increases by a factor of about 1.57 for each increase in $n$ by 1. In contrast our recursion relation can be used to directly generate such expressions – the time for those calculations increases roughly by a factor of 1.24 for increase in $n$ by 1. The differences of their rates of exponential growth are quite significant. These recurrence relations and Mathematica[4] code for generating those expressions can be found below, as are some possibilities for ways to compute them efficiently to much higher orders.

# 1 Cumulants in terms of Moments

The recurrence relations[3] specifically are

$$C_{n+1}(\gamma) = \frac{dC_n(\gamma)}{d\gamma}; \qquad P_{n+1}(\gamma) = \frac{dP_n(\gamma)}{d\gamma}; \qquad C_0(\gamma) = \ln P_0(\gamma),$$

with $\gamma$ a formal parameter of no significance here. These give

$$C_1 = \frac{dC_0}{d\gamma} = \frac{d\ln P_0(\gamma)}{d\gamma} = \frac{1}{P_0(\gamma)}\frac{dP_0(\gamma)}{d\gamma} = \frac{P_1(\gamma)}{P_0(\gamma)} = p_1,$$

where $p_n$ are the moments of the *normalized* distribution, and $P_n$ refer to the un-normalized distribution. Similarly

$$C_2 = \frac{dC_1}{d\gamma} = \frac{d}{d\gamma}\frac{P_1}{P_0} = \frac{P_0P_1' - P_1P_0'}{P_0^2} = \frac{P_0P_2 - P_1P_1}{P_0^2} = p_2 - p_1^2,$$

where we have suppressed the dependence on the formal parameter $\gamma$, and the prime indicates differentiation with respect to $\gamma$. Expressions for cumulants of all orders can be generated this way, which is relatively fast and easy up to order $\approx 60$ and more using modern computer algebra systems, or potentially other methods described below for higher orders.

This can be implemented very simply in Mathematica as:

```
nmax = 50; nextn = D[#, γ]/.{D[P[n_, γ], γ] → P[n + 1, γ]}&;
AbsoluteTiming[tableau = NestList[nextn, Log[P[0, γ]], nmax]; ].
cumulants = Drop[tableau//.{P[n_, γ] → p[n], p[0] → 1}, 1];
```

Up to fifth order (so as to fit on the page) they are:

$$
\begin{aligned}
c_1 &= p_1 \\
c_2 &= p_2 - p_1^2 \\
c_3 &= 2p_1^3 - 3p_2p_1 + p_3 \\
c_4 &= -6p_1^4 + 12p_2p_1^2 - 4p_3p_1 - 3p_2^2 + p_4 \\
c_5 &= 24p_1^5 - 60p_2p_1^3 + 20p_3p_1^2 + 30p_2^2p_1 - 5p_4p_1 - 10p_2p_3 + p_5 \\
&\dots
\end{aligned}
$$

Choosing the origin so that $p_1 = 0$ we have:

$$
\begin{aligned}
c_1 &= p_1 = 0 \\
c_2 &= p_2 \\
c_3 &= p_3 \\
c_4 &= p_4 - 3p_2^2 \\
c_5 &= p_5 - 10p_2p_3 \\
c_6 &= 30p_2^3 - 15p_4p_2 - 10p_3^2 + p_6 \\
c_7 &= 210p_3p_2^2 - 21p_5p_2 - 35p_3p_4 + p_7 \\
c_8 &= -630p_2^4 + 420p_4p_2^2 + 560p_3^2p_2 - 28p_6p_2 - 35p_4^2 - 56p_3p_5 + p_8 \\
c_9 &= -7560p_3p_2^3 + 756p_5p_2^2 + 2520p_3p_4p_2 - 36p_7p_2 + 560p_3^3 - 126p_4p_5 - 84p_3p_6 + p_9 \\
&\dots
\end{aligned}
$$

## 2 Moments in terms of Cumulants

Expressions for the moments in terms of the cumulants can be generated in a similar way:

$$P_1 = \frac{dP_0}{d\gamma} = \frac{d(e^{C_0})}{d\gamma} = e^{C_0}\frac{dC_0}{d\gamma} = e^{C_0}C_1 = P_0C_1$$

giving $p_1 = P_1/P_0 = C_1$. Continuing, we have

$$P_2 = \frac{d(e^{C_0}C_1)}{d\gamma} = e^{C_0}C_0'C_1 + e^{C_0}C_1' = e^{C_0}(C_1^2 + C_2) = P_0(C_1^2 + C_2)$$

giving $p_2 = P_2/P_0 = C_1^2 + C_2$, and so on. Here the prime denotes differentiation with respect to $\gamma$.

Choosing the origin at the centroid of the distribution vastly simplifies the expressions. The full forms can be reconstituted from the simplified forms if needed. The recurrence can be implemented very simply in Mathematica as:

```
nmax = 50; nextn = D[#, γ]/.{D[c[n_, γ], γ] → c[n + 1, γ]}&;
AbsoluteTiming[tableau = NestList[nextn, Exp[c[0, γ]], nmax]; ].
moments = Drop[tableau//.{c[n_, γ] → c[n], c[0] → 0}, 1];
```

For arbitrary origin we have (up to sixth order, to fit on the page):

$$
\begin{aligned}
p_1 &= c_1 \\
p_2 &= c_1^2 + c_2 \\
p_3 &= c_1^3 + 3c_2c_1 + c_3 \\
p_4 &= c_1^4 + 6c_2c_1^2 + 4c_3c_1 + 3c_2^2 + c_4 \\
p_5 &= c_1^5 + 10c_2c_1^3 + 10c_3c_1^2 + 15c_2^2c_1 + 5c_4c_1 + 10c_2c_3 + c_5 \\
p_6 &= c_1^6 + 15c_2c_1^4 + 20c_3c_1^3 + 45c_2^2c_1^2 + 15c_4c_1^2 + 60c_2c_3c_1 + 6c_5c_1 + 15c_2^3 + 10c_3^2 + 15c_2c_4 + c_6 \\
&\dots
\end{aligned}
$$

However, if we choose the origin to be located at the centroid of the distribution $p_1 = 0$, and many terms vanish:

$$
\begin{aligned}
p_1 &= 0 \\
p_2 &= c_2 \\
p_3 &= c_3 \\
p_4 &= 3c_2^2 + c_4 \\
p_5 &= 10c_2c_3 + c_5 \\
p_6 &= 15c_2^3 + 15c_4c_2 + 10c_3^2 + c_6 \\
p_7 &= 105c_3c_2^2 + 21c_5c_2 + 35c_3c_4 + c_7 \\
p_8 &= 105c_2^4 + 210c_4c_2^2 + 280c_3^2c_2 + 28c_6c_2 + 35c_4^2 + 56c_3c_5 + c_8 \\
p_9 &= 1260c_3c_2^3 + 378c_5c_2^2 + 1260c_3c_4c_2 + 36c_7c_2 + 280c_3^3 + 126c_4c_5 + 84c_3c_6 + c_9 \\
&\dots
\end{aligned}
$$

For $c_{50}$, 85% of the terms vanish. They can be reconstituted however. For example, setting $p_n = \langle (x - \bar{x})^n \rangle$; expanding the terms, and averaging regenerates the full expressions. Storing expressions in the $p_1 \to 0$ form might be beneficial if storage of large expressions for very large n were to become onerous. This can give a more than 7-fold reduction in required memory.

## 3 Discussion and Conclusion

The recurrence relations and Mathematica code implementations allow computation of expressions for the cumulants in terms of the moments, and vice versa for fairly large systems. The need for this is unclear, but there may be applications in statistical or combinatorial mathematics. At minimum the recurrence relations work very nicely for modest sized systems as well.

Explicitly solving for the moments in terms of the cumulants rapidly becomes impractical and time consuming for large $n$, even using sophisticated computer algebra systems, so having this direct way to compute the moments in terms of the cumulants is useful.

If one were interested in carrying out the recurrence to much higher order, there is an alternative to explicitly taking the derivatives. Because the recurrence relations only involve

taking derivatives of sums of simple products of moments to various integer powers (including negative powers of $P_0$), the operation can be trivially parallelized over the terms in the sum, because the derivative is a linear operation. Although ParallelMap in Mathematica is quite effective, it should be feasible to parallel-process these operations even on GPU cores, in which each core takes the derivative of a single term or a batch of terms in a cumulant expression. Only integer addition (subtraction) of exponents is required to calculate the derivatives. Using Feynman's trick for differentiating a product of expressions taken to various powers, one can quickly evaluate derivatives of the individual terms, because the computation really just amounts to keeping track of the integer exponents of the various moment factors, and some weights. Specifically, if an individual term $T = c\,\Pi_{j=0}^{n} P_j^{m_j}$ then $\log T = \log c + \sum_{j=0}^{n} m_j \log P_j$ and $T^{'} = T(\sum_{j=0}^{n} m_j P_j^{'}/P_j) = T(\sum_{j=0}^{n} m_j P_{j+1}/P_j)$, using the recurrence relation. Each term can be described simply by an overall multiplicative signed-integer factor, and a list of signed-integer-valued powers of the various un-normalized moments. Multiplying through by $T$ simply increments the powers of the corresponding terms by various integers, so no floating point operations are required. The sum over all the terms is then multiplied by a signed-integer (possibly large) scale factor. Such arithmetical bookkeeping is simple enough it easily could be done at the individual GPU or TPU kernel level, with the total expressions summed at the end. A GPU with thousands of cores should make short work of such calculations, depending on operational details such as bandwidth to memory etc, if such calculations were needed. We have implemented this scheme in Mathematica, and it works correctly, without explicitly taking derivatives, as advertised.

## References


[1] M. G. Kendall and A. Stuart, *The Advanced Theory of Statistics. Vol. 2* (Charles Griffin, 1961).

[2] P. McCullagh and J. Kolassa, Scholarpedia **4**, 4699 (2009).

[3] G. Bunker, Nuclear Instruments and Methods in Physics Research **207**, 437 (1983).

[4] Mathematica, Version 14.0, Wolfram Research, Inc., Champaign, IL.