

Coskewness and Cokurtosis

John W. Fowler

July 9, 2005

The concept of a covariance matrix can be extended to higher moments; of particular interest are the third and fourth moments. A very common application of covariance matrices is the error covariance matrix, which is defined as the matrix of expectation values for pairs of zero-mean random variables representing errors. A minimum of two random variables is required in order for nontrivial covariance to be manifested; any number of random variables greater than one can be accommodated in principle. For concreteness and brevity, we will consider five random variables denoted ϵ_i , $i = 1$ to 5. The covariance matrix is comprised of the expectation values of all possible pairs of these random variables:

$$\Omega \equiv \begin{pmatrix} \langle \epsilon_1 \epsilon_1 \rangle & \langle \epsilon_1 \epsilon_2 \rangle & \langle \epsilon_1 \epsilon_3 \rangle & \langle \epsilon_1 \epsilon_4 \rangle & \langle \epsilon_1 \epsilon_5 \rangle \\ \langle \epsilon_2 \epsilon_1 \rangle & \langle \epsilon_2 \epsilon_2 \rangle & \langle \epsilon_2 \epsilon_3 \rangle & \langle \epsilon_2 \epsilon_4 \rangle & \langle \epsilon_2 \epsilon_5 \rangle \\ \langle \epsilon_3 \epsilon_1 \rangle & \langle \epsilon_3 \epsilon_2 \rangle & \langle \epsilon_3 \epsilon_3 \rangle & \langle \epsilon_3 \epsilon_4 \rangle & \langle \epsilon_3 \epsilon_5 \rangle \\ \langle \epsilon_4 \epsilon_1 \rangle & \langle \epsilon_4 \epsilon_2 \rangle & \langle \epsilon_4 \epsilon_3 \rangle & \langle \epsilon_4 \epsilon_4 \rangle & \langle \epsilon_4 \epsilon_5 \rangle \\ \langle \epsilon_5 \epsilon_1 \rangle & \langle \epsilon_5 \epsilon_2 \rangle & \langle \epsilon_5 \epsilon_3 \rangle & \langle \epsilon_5 \epsilon_4 \rangle & \langle \epsilon_5 \epsilon_5 \rangle \end{pmatrix}$$

An arbitrary element of the matrix is denoted Ω_{ij} . Because multiplication is commutative, the matrix is symmetric, i.e., $\Omega_{ij} = \Omega_{ji}$. The number of independent elements is $N(N+1)/2$, where N is the number of elements in a row or column, in this case 5. The diagonal elements are the variances of the random variables, commonly denoted $\sigma_i^2 = \Omega_{ii}$.

The skewness of a random variable is defined as $S = \langle \epsilon^3 \rangle / \sigma^3$, and the kurtosis is $K = \langle \epsilon^4 \rangle / \sigma^4$. Sometimes 3 is subtracted from the kurtosis; when this is done, it is better to refer to the *excess kurtosis*; the relevance of 3 is that it is the value of the kurtosis for any Gaussian random variable (i.e., the excess kurtosis of a Gaussian random variable is zero). From here on we will ignore the normalization by powers of σ and concentrate on the higher moments.

The generalization of skewness and kurtosis to coskewness and cokurtosis will involve the expectation values $\langle \epsilon_i \epsilon_j \epsilon_k \rangle$ and $\langle \epsilon_i \epsilon_j \epsilon_k \epsilon_m \rangle$, respectively, where we will use the subscript m rather than l to avoid confusion with the literal number 1. Note that the presence of up to four indices does not imply that more than two random variables are involved; each index runs from 1 to N , and N need only be at least 2 for the concepts of coskewness and cokurtosis to be nontrivial. Thus there are two coskewnesses for any two random variables, e.g., $\langle \epsilon_1 \epsilon_1 \epsilon_2 \rangle$ and $\langle \epsilon_1 \epsilon_2 \epsilon_2 \rangle$, and one for any three random variables, e.g., $\langle \epsilon_1 \epsilon_2 \epsilon_3 \rangle$. Similar remarks apply for the cokurtosis except that more distinct combinations occur.

While the covariance matrix is a covariant tensor of rank 2, extending the matrix concept to these higher moments will require tensors of rank 3 and 4 for coskewness and cokurtosis, respectively. We will define the elements as $S_{ijk} = \langle \epsilon_i \epsilon_j \epsilon_k \rangle$ and $K_{ijkm} = \langle \epsilon_i \epsilon_j \epsilon_k \epsilon_m \rangle$. These are also strictly

symmetric tensors. With N elements per row/column, the number of independent elements in a strictly symmetric tensor of rank M is

$$L = \frac{(M + N - 1)!}{M!(N - 1)!}$$

This is just the number of ways to take N things M at a time with repetition. For $M = 2$, this reduces to the common expression for an $N \times N$ matrix

$$\begin{aligned} L &= \frac{(2 + N - 1)!}{2!(N - 1)!} = \frac{(N + 1)!}{2(N - 1)!} = \frac{(N + 1)N(N - 1)!}{2(N - 1)!} \\ &= \frac{N(N + 1)}{2} \end{aligned}$$

For $M = 3$,

$$\begin{aligned} L &= \frac{(3 + N - 1)!}{3!(N - 1)!} = \frac{(N + 2)!}{6(N - 1)!} = \frac{(N + 2)(N + 1)N(N - 1)!}{6(N - 1)!} \\ &= \frac{N(N + 1)(N + 2)}{6} \end{aligned}$$

And for $M = 4$,

$$\begin{aligned} L &= \frac{(4 + N - 1)!}{4!(N - 1)!} = \frac{(N + 3)!}{24(N - 1)!} = \frac{(N + 3)(N + 2)(N + 1)N(N - 1)!}{24(N - 1)!} \\ &= \frac{N(N + 1)(N + 2)(N + 3)}{24} \end{aligned}$$

For example, in the case $M = 3$, $N = 5$, the 35 independent elements are those with indices

#1: 111	#13: 144	#25: 255
#2: 112	#14: 145	#26: 333
#3: 113	#15: 155	#27: 334
#4: 114	#16: 222	#28: 335
#5: 115	#17: 223	#29: 344
#6: 122	#18: 224	#30: 345
#7: 123	#19: 225	#31: 355
#8: 124	#20: 233	#32: 444
#9: 125	#21: 234	#33: 445
#10: 133	#22: 235	#34: 455
#11: 134	#23: 244	#35: 555
#12: 135	#24: 245	

All other elements have indices which are permutations of those shown, and any such element is equal to the element with corresponding permuted indices above. In general, any indices can be permuted into an ascending or descending sequence, located in a list such as that above, and thereby mapped into a linear sequence usable for efficient storage in computer memory. The position in the linear sequence will be indicated by the index p . Numerical applications involve mapping in both directions between p and the set of indices of an element in the symmetric matrix. For $M = 2$, these mappings are simple, as shown in Appendix A. The remainder of this paper will be concerned with developing mappings for $M > 2$ based on table lookup.

For a tensor of rank M , each element has M indices I_n , $n = 1$ to M (we will consider only covariant indices, since this keeps the covariance-matrix analogy intact, and there is no need herein to distinguish between covariant and contravariant indices). For example, for $p = 22$ (i.e., item #22) in the list above, we have $I_1 = 2$, $I_2 = 3$, $I_3 = 5$. There are L such index sets, i.e., p runs from 1 to L . We will use the doubly subscripted I_{np} to denote the n^{th} index value in the p^{th} index set, i.e., in the p^{th} position in the sorted list, where the list will be generated in a manner that produces the sort order we want in both in dimensions.

For a given p , a number Q_p in base N can be defined such that each index defines a digit of Q_p with $I_{1p} - 1$ being the least significant digit, $I_{2p} - 1$ is the next least significant, and so on up to the most significant digit $I_{Mp} - 1$ (we subtract 1 because zero never occurs as an index value, so for the number base to be N rather than $N + 1$, we must subtract 1 as indicated):

$$Q_p = \sum_{n=1}^M (I_{np} - 1) N^{n-1}$$

Q_p is useful as a monotonically increasing function of p for table lookup purposes. Some such number is needed, and Q_p is less likely to present integer overflow problems in typical applications than, for example, a decimal number simply constructed with digits equal to the indices themselves. For example, the case $M = 3$, $N = 1000$ has maximum values of $p = 167,167,000$, $Q_p = 999,999,999$; a number constructed from the indices would have a maximum value of 100,010,001,000, which overflows a four-byte IEEE integer variable, whereas the maximum Q_p value does not. The following algorithm will generate the list such that Q_p increases with p and I_{np} decreases with n . Thus if we read n as *increasing from right to left* in the example, the algorithm will generate the list as shown. The example of $p = 22$ above then becomes $I_1 = 5$, $I_2 = 3$, $I_3 = 2$.

The general algorithm will use a set of symbolic tokens T_i , $i = 1$ to N , where the tokens may be general objects (e.g., “cat”, “dog”, “goldfish”, etc.). Here, $T_i = i$. For the example above, this yields the set of tokens $\{1, 2, 3, 4, 5\}$. The algorithm then consists of generating the list of sets corresponding to N things taken M at a time with repetition; this will result in L sets, each consisting of M tokens, generated in order of increasing Q_p . The algorithm uses a procedure which we will call RollOver, because the index sets are generated in a manner similar to an odometer increasing from 111...1 to $NNN...N$ except that when a digit rolls over, it does not roll over to 1 but rather to the digit to its left. The procedure is as follows, where R is an array of M elements which serve as pointers into the array of tokens (in our case, each token is just equal to its pointer, so we have no specific need for the token array, but it is useful for more general cases).

```

Procedure RollOver (R,M,N)
  i = 0
  repeat
    i ← i + 1
    if R(i) < N then begin
      R(i) ← R(i) + 1
      for j = 1 to i-1 do R(j) = R(i)
    exit
  end
  until i = M
end procedure

```

It is assumed that if $i = 1$, the loop from $j = 1$ to $i-1$ does not execute. The program that uses this procedure is as follows, where the I array corresponds to I_{np} above.

```

for k = 2 to M do R(k) = 1
R(1) = 0
P = 0

repeat
  RollOver (R,M,N)
  rsum = 0
  for k = 1 to M do rsum ← rsum + R(k)
  P ← P + 1
  for k = 1 to M do I(k,P) = R(k)
until rsum = M*N

```

The repeat loop ends when the sum of the indices in R is equal to $M \times N$, i.e., when all the index values in the set are N . At this point the L index sets have been generated. For any one set contained in R, Q_p can be computed by the following function.

```

Function Qp (R,M,N)
  Itmp = 0
  N2k = 1
  for k = 1 to M do begin
    Itmp ← Itmp + (R(k)-1) * N2k
    N2k ← N2k * N
  end
  Qp = Itmp
end function

```

Note that Q_p increments with occasional jumps because when the “odometer” rolls over, the digit rolling over does not start again at 1. An example with $N = 3$, $M = 5$ is shown below, where Q_p is shown as a decimal (not base 3) number.

P	R	Q_p	P	R	Q_p
1	11111	0	11	12222	40
2	11112	1	12	12223	41
3	11113	2	13	12233	44
4	11122	4	14	12333	53
5	11123	5	15	13333	80
6	11133	8	16	22222	121
7	11222	13	17	22223	122
8	11223	14	18	22233	125
9	11233	17	19	22333	134
10	11333	26	20	23333	161
			21	33333	242

The mapping from p to the index set is straightforward; for example, if one wishes to know the indices for $p = 10$, one simply looks in the 10th location in the table and reads them out.

The inverse mapping requires a table search; we assume that binary search is as good a method as any. The first step is to sort the indices in an ascending sequence. Given an index set I , for example, $(3,1,2,3,2)$, this is sorted into $(1,2,2,3,3)$. Then Q_p is computed; the algorithm results in $Q_p = 44$. A binary search of the table will then find this value at $p = 13$. The value stored in the one-dimensional array at location 13 is then the desired element in the symmetric rank-5 tensor.

In principle, since Q_p is a monotonic function of p , an invertible functional form $Q_p(p)$ could be fit to sufficient accuracy for the inverse to be used directly instead of resorting to table lookup. This is the case for $M = 2$, as discussed in the appendix. It is not clear that this would be computationally more efficient, however, given the complications involved in inverting polynomials of order higher than 2 and the fact that in most applications N is not a gigantic number. The example mentioned above of $M = 3$, $N = 1000$, a coskewness tensor with 1000 variables, involves 167,167,000 independent elements; once the table is generated, any element could be found by a binary search in at most 28 steps with relatively simple code.

Appendix: Index Mapping for Symmetric 2-Dimensional Matrices

The case of 2-dimensional matrices is the most frequently encountered in most applications involving symmetric tensors (e.g., error covariance matrices). For this case, table lookup is not required. The forward mapping, which has the nice feature of not depending on the size of the matrix, was first pointed out to the author by E. L. Kopan (1974, private communication); the inverse mapping simply employs the quadratic formula.

Given the index pair (i, j) , swap values if necessary to force $j \geq i$. The one-dimensional index p is then given by $p = j(j-1)/2 + i$. One may notice the presence of the formula for the number of elements in an upper triangular matrix of size j in this expression, which is no coincidence.

The inverse mapping is based on the idea that for a given value of p , j is the largest number that yields a $j(j-1)/2$ less than p . So we can set i temporarily to 1 and solve the expression above for j . We have $p = j(j-1)/2 + 1$. Since p is known, the value of j can be obtained via the quadratic formula, ignoring the solution with the negative square root because it yields negative results for j . The resulting algorithm is

$$j = \left\{ \frac{\sqrt{8p - 7} + 1}{2} \right\}_{truncated}$$
$$i = p - \frac{j(j-1)}{2}$$