Formulas for SU(3) Matrix Generators

Richard Shurtleff *

September 30, 2024

Abstract

The Lie algebra of a Lie group is a set of commutation relations, equations satisfied by the group's generators. For SU(2) and many other Lie groups, the equations have been solved and matrix generators are realized as algebraic expressions suitable for further investigation or numerical evaluation. This article presents formulas that give a set of matrix generators for any irreducible representation of the group SU(3), the group of unimodular unitary three-dimensional complex matrices with matrix multiplication. To assist in calculating the matrix generators, a Mathematica computer program and a Fortran 90 program are included.

1 Introduction

When working with a group, it is sometimes convenient to have numerical matrix representations. For some Lie groups, obtaining matrix representations is as simple as substituting a numerical value in a set of algebraic formulas. Having formulas for a group's matrix generators invites programming, so that coding into computer software allows computers to do the arithmetic. With formulas, one can explore the group's representations with algebraic and analytic manipulations.

For example, consider the Lie group of unitary unimodular 2×2 complex matrices with the usual dot product, the Lie group SU(2).[1] An irreducible representation, an 'irrep', has three spin matrix generators, denoted $T^{\pm}$ and T^3. Each irrep is determined by its spin s, a value that when doubled is a nonnegative integer. Matrices can be written down by substituting s into well-known formulas. For example, one set of formulas gives the T^+ matrix as[1]

$$T^+_{\alpha\beta} = \sqrt{s(s+1) - \alpha(\alpha-1)}\,\delta_{\beta,\alpha+1} \quad ,$$

where $\alpha, \beta = s, s-1, ..., -s$. Given an integer or half-integer s, one can write the matrix. Indeed, the formulas for $T^{\pm}$ and T^3 are useful in this project and the SU(2) formulas appear as Eqs. (16) in Sec. 3 below.

*affiliation and mailing address: Department of Applied Mathematics and Sciences, Wentworth Institute of Technology, 550 Huntington Avenue, Boston, MA, USA, ZIP 02115, e-mail addresses: shurtleffr@wit.edu, momentummatrix@yahoo.com

Other Lie groups with formulas for matrix generators include the group SO(3) of rotations in three-dimensional space,[1, 2, 3] the Lorentz group SL(2,C) in four-dimensional spacetime,[4] and the Poincare group $R^{1,3}\times$ SL(2,C) [5, 6] also in spacetime.

It may be that the algebra of SU(3) has been solved and formulas for its generators exist. However, none have come to my attention. In any case, this article contributes a set of formulas for the generators of the Lie group SU(3).

Alternatively, other procedures for obtaining matrix generators of SU(3) have been developed and are widely available, see, for example, Refs. [7, 8]. As the dimension of the matrices increases, these other procedures become more cumbersome.

In lieu of a proof, we provide computer programs in Secs. 6 and 7 to verify the formulas. I have run the software which produced sets of generators for many irreps, with the Lie algebra and some other requirements tested for each irrep. In the terminology defined below, the (p, q) irreps have been checked for $p+q = 1, 2, ..., 15$. Success with the tested irreps does not prove that the formulas will work for untested irreps. A proof, or supporting calculations showing that the formulas work in general, may be presented elsewhere.

For discussions of SU(3) in language like that adopted here, see, for example, Refs. [9, 10], two popular elementary particle physics texts.

The commutation relations of the algebra of the Lie group SU(3) are derived in Sec. 2. Those commutation relations are the equations that the TUV matrix generators obey. The solutions involve a block structure for the matrices and indirect formulas that are functions of parameters. This entails parameter spaces, block index formulas and component formulas, a system much more complicated than is needed for SU(2). So, we devote Sec. 3 to discuss the structure of the solutions. The formulas for the TUV matrix generators are presented in Sec. 4. Some remarks are collected in Sec. 5. Finally, the computer programs take up Secs. 6 and 7.

2 Equations

The set of unimodular, unitary 3×3 matrices with complex components combined with matrix multiplication form a representation of the topological group SU(3). To obtain the Lie algebra of SU(3), consider a set of group elements G that are close to the identity matrix $\mathbf{1}$. In terms of a 3×3 matrix A with the absolute value of each of its components bounded by some small value ϵ, we have

$$G = \mathbf{1} + A \quad , \tag{1}$$

Since G is unitary, we have $GG^\dagger = \mathbf{1}$, and

$$(\mathbf{1} + A)(\mathbf{1} + A)^\dagger = (\mathbf{1} + A + A^\dagger) + O(\epsilon^2) = \mathbf{1},$$

which implies that

$$A + A^\dagger = O(\epsilon^2) \quad . \tag{2}$$

Thus, A is skew-hermitian, $A = iH$, where H is hermitian, $H = H^\dagger$. In this, the dagger ($\dagger$) indicates the combination of the complex conjugation of the matrix components and the matrix transpose, which together make the hermitian conjugate.

For G to be unimodular, it's determinant is unity, $\det G = 1$. With the antisymmetric symbol ϵ^{ijk}, where $i, j, k = 1, 2, 3$ and $\epsilon^{123} = 1$, one has

$$1 = \det G = \epsilon^{ijk} G^1_i G^2_j G^3_k = \epsilon^{ijk}(\delta^1_i + A^1_i)(\delta^2_j + A^2_j)(\delta^3_k + A^3_k) = \epsilon^{123} + \epsilon^{i23}A_i + \epsilon^{1j3}A_j + \epsilon^{12k}A_k \ ,$$

which implies that

$$A^1_1 + A^2_2 + A^3_3 = O(\epsilon^2) \quad . \tag{3}$$

Repeated indices are summed. The delta function δ^i_j is unity when $i = j$ and vanishes otherwise. Hence, the trace of A vanishes to second order in ϵ. Since $A = iH$, the trace of H must vanish as well.

By Eqs. (2) and (3), the 3×3 matrix H is both hermitian and traceless. With the help of matrix exponentiation, $\exp itH$, where t is real, one can show that the matrix exponent of the skew-hermitian matrix itH is a unimodular, unitary 3×3 matrix with components that are not necessarily small because t can be large. By rescaling H and t with some positive real factor, we can drop the requirement that the components of H are small. The rescaled H is hermitian and traceless. We have

$$G = e^{itH} \quad , \tag{4}$$

where t is real. Any group element G can be constructed this way. The set of traceless hermitian matrices H forms the 'generators' of G.

A basis for the set of generators H is the set of eight matrices F^i, where[9, 10]

$$F^1 = \frac{1}{2}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad F^2 = \frac{1}{2}\begin{pmatrix} 0 & -i & 0 \\ i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad F^3 = \frac{1}{2}\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad , \tag{5}$$

$$F^4 = \frac{1}{2}\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad F^5 = \frac{1}{2}\begin{pmatrix} 0 & 0 & -i \\ 0 & 0 & 0 \\ i & 0 & 0 \end{pmatrix} \quad F^6 = \frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad ,$$

$$F^7 = \frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -i \\ 0 & i & 0 \end{pmatrix} \quad F^8 = \frac{1}{2\sqrt{3}}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{pmatrix} \quad .$$

The matrices $2F^i$ are called the 'Gell-Mann' matrices. By inspection, the F^i are hermitian and traceless.

Since the matrices F^2, F^5, F^7 have pure imaginary components and the F^1, F^3, F^4, F^8 are real, one can have matrices with real valued components by defining suitable linear combinations of the F^i matrices. The 'TUV matrices' are the following linear combinations of the F^i,

$$T^\pm = F^1 \pm iF^2 \quad ; \quad T^3 = F^3 \quad ; \quad V^\pm = F^4 \pm iF^5 \quad ; \tag{6}$$

$$U^{\pm} = F^6 \pm iF^7 \quad ; \quad U^3 = -\frac{1}{2}F^3 + \frac{\sqrt{3}}{2}F^8 \quad .$$

With the F^i in Eq. (5) , the TUV matrices are eight 3×3 real basis generators $\{T^+, T^-, T^3, U^+, U^-, U^3, V^+, V^-\}$.

Once the eight 3×3 TUV matrices are calculated from Eq. (5) and (6), their commutation relations can be found. One has

$$[T^+, T^-] = 2\,T^3 \quad ; \quad [T^3, T^{\pm}] = \pm\, T^{\pm} \quad ; \tag{7}$$

$$[U^+, U^-] = 2\,U^3 \quad ; \quad [U^3, U^{\pm}] = \pm\, U^{\pm} \quad ; \tag{8}$$

$$[V^+, V^-] = 2U^3 + 2T^3 \quad ; \quad [U^3, V^{\pm}] = \pm\,\frac{1}{2}V^{\pm} \quad ; \quad [T^3, V^{\pm}] = \pm\,\frac{1}{2}V^{\pm} \quad ; \tag{9}$$

$$[T^3, U^{\pm}] = \mp\,\frac{1}{2}U^{\pm} \quad ; \quad [T^3, V^{\pm}] = \pm\,\frac{1}{2}V^{\pm} \quad ; \quad [U^3, T^{\pm}] = \mp\,\frac{1}{2}T^{\pm} \quad ; \tag{10}$$

$$[U^3, U^{\pm}] = \pm\, U^{\pm} \quad ; \quad [U^3, V^{\pm}] = \pm\,\frac{1}{2}V^{\pm} \quad ; \quad [T^3, U^3] = 0 \quad ; \tag{11}$$

$$[T^{\pm}, U^{\mp}] = [T^{\pm}, V^{\pm}] = 0 \quad ; \quad [T^{\pm}, U^{\pm}] = \pm\, V^{\pm} \quad ; \quad [T^{\pm}, V^{\mp}] = \mp\, U^{\mp} \quad . \tag{12}$$

$$[V^{\pm}, U^{\mp}] = \pm\, T^{\pm} \quad ; \quad [U^{\pm}, V^{\pm}] = 0 \quad . \tag{13}$$

These are the commutation relations of the specific set of 3×3 TUV matrices calculated from Eq. (5) and (6). The commutation relations form the '$\mathfrak{su}(3)$ algebra.'

There are many solutions to the $\mathfrak{su}(3)$ algebra, not just the specific set of 3×3 TUV matrices calculated from Eq. (5) and (6). Formulas to obtain matrices that satisfy the $\mathfrak{su}(3)$ algebra are in Secs. 3 and 4.

Those solutions include a set of eight 3×3 TUV matrices that are equivalent to the TUV matrices calculated from Eq. (5) and (6), as well as other sets of TUV matrices that satisfy the $\mathfrak{su}(3)$ algebra, Eqs. (7) – (12). Despite the terminology conflict, the name 'TUV matrices' applies to any and all solutions of the Lie algebra and the label 'F^i' includes all sets of matrices found by solving Eq. (6) for the F^i with any given set of TUV matrices.

It is known that there is an irrep of the $\mathfrak{su}(3)$ algebra for each pair of nonnegative integers (p, q). The TUV matrix solutions in Secs. 3 and 4 require $p \geq q$. This can be seen, for example, by inspecting the solutions' parameter spaces in Sec. 4.

The TUV matrices for the other irreps, those with $p < q$, are the negative transpose of the TUV matrices with $p > q$. Thus, irreps with $p < q$ can be found from irreps with $p > q$ by adding a couple of extra steps to the solution. The extra steps appear at the end of Sec. 4. In this way, the irreps of the $\mathfrak{su}(3)$ algebra for any nonnegative integer pair (p, q) can be calculated from the TUV matrix formulas in Secs. 3 and 4.

3 Structure of the Solutions

This section contains a description of the structure of the solutions in Sec. 4. To prepare for the more complicated situation with SU(3), we first consider the group SU(2), which is the group of unitary unimodular 2×2 complex matrices combined with matrix multiplication.

The commutation relations of a basis for the matrix generators of SU(2), the $\mathfrak{su}(2)$ algebra, are [1, 2, 3]

$$[T^i, T^j] = i\epsilon^{ijk}\, T^k \quad , \tag{14}$$

where $i, j, k = 1, 2, 3$ are indices for Cartesian components of three-dimensional space. The excuse for using the letter 'T' to label the SU(2) matrix generators can be seen by defining $T^{\pm} = T^1 \pm iT^2$. The commutation relations are now

$$[T^+, T^-] = 2\, T^3 \quad ; \quad [T^3, T^{\pm}] = \pm\, T^{\pm} \quad , \tag{15}$$

which coincide with the SU(3) commutation relations of the T-matricies, Eq. (7). So we use the letter 'T' here as well.

Formulas for the T-matrices of an SU(2) irrep are well known. The irrep for spin s, where $2s$ is a nonnegative integer, has T-matrix components[4]

$$T^{\pm}_{\alpha\beta} = \sqrt{(s \pm \alpha)(1 + s \mp \alpha)}\, \delta_{\beta,\alpha\mp 1} \quad ; \quad T^3_{\alpha\beta} = \alpha\, \delta_{\alpha,\beta} \quad , \tag{16}$$

where $\alpha, \beta = s, s-1, ..., -s$. The allowed values of the 'spin component' indices α and β are determined by the irrep's 'spin' s and the functions for the components are functions of the spin s.

To connect with the language for SU(3), we say that the SU(2) T-matrices are "determined by the value of parameter s in the parameter space of the SU(2) irrep." The parameter space is discrete, with just one allowed value, the spin s. The parameter spaces of SU(3), by contrast, have discrete values of two parameters and more than one pair. See Fig. 1.

It has been noted that the three SU(3) commutation relations involving T-matrices, Eq. (7), replicate the commutation relations of the Lie algebra of the group SU(2). The TUV matrix solutions in Sec. 4 take advantage of this. The T-matrices in the solutions are reduced to block diagonal form, with the T-matrices of SU(2) irreps in Eq. (16) along the diagonal. This is basic to the structure of the TUV matrix solutions.

The list of T-spins for the (p, q) irrep and the list of U^3 eigenvalues, often called 'weights', can be found in, for example, Ref. [11]. The number of T-spins N_T is $N_T = (p+1)(q+1)$. It follows that the TUV matrices are $N_T \times N_T$ arrays of blocks. The dimension of a block with T-spin s is $2s+1$. Knowing the T-spin list and the number of rows and columns of components in each block, allows one to calculate the dimension d of a TUV matrix,[9, 10]

$$d = \frac{1}{2}(p+1)(q+1)(p+q+2) \quad . \tag{17}$$

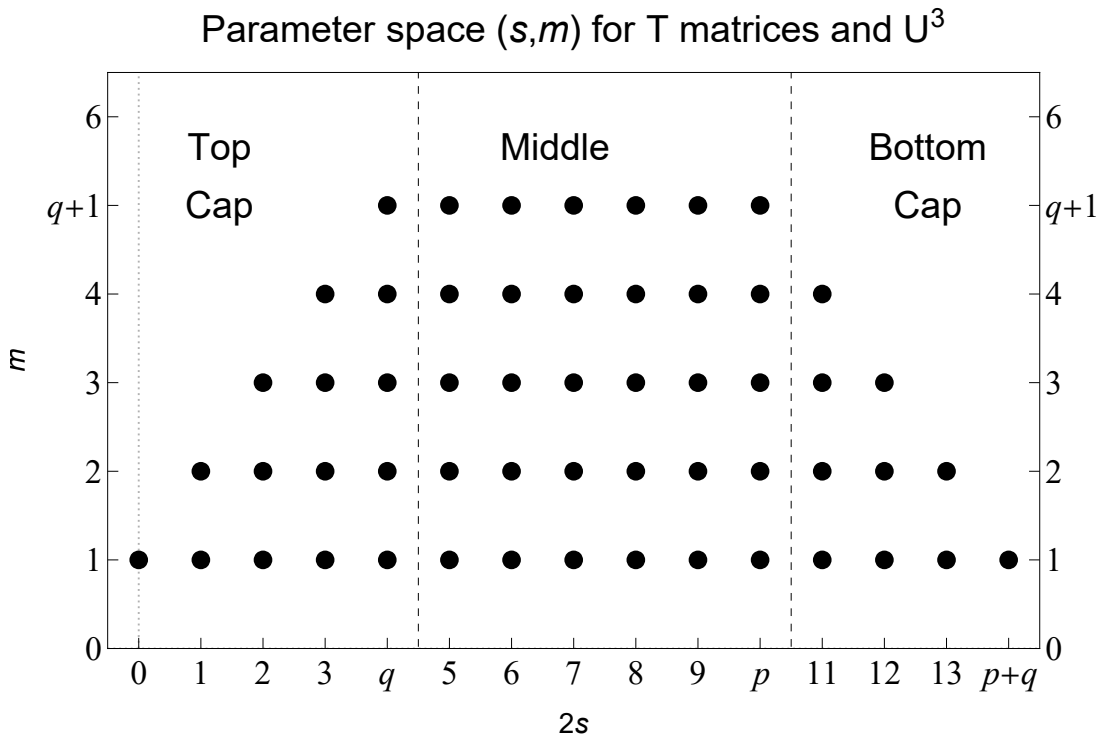

Figure 1: *The allowed values of the spin parameter s and its multiplicity index m for the matrices $T^{\pm}$, T^3, U^3 in Collection I. The $(p,q) = (10,4)$ irrep is shown.* The parameters s and m appear in the formulas for the (i,j) block addresses and the block's (α,β) components in Collection I in Eqs. (21) to (29). The collection of matrices $T^{\pm}$, T^3, U^3 are one of five such subsets of TUV matrices each with its own parameter space (s,m), its own (i,j) formulas and its own component formulas. Any one collection's parameter space is similar, but distinct, from the space of any another.

Clearly, the number of components, d^2, in a TUV matrix for the (p,q) irrep grows quickly with p and q.

Each TUV matrix has rows and columns of blocks, and each block has rows and columns of components. The block structure can be illustrated for the T^+ matrix. The same block structure applies to all the TUV matrices.

Consider the matrix T^+ of an SU(3) irrep. We put block indices in parentheses, (i,j), so the address of a block in the ith row and the jth column is written $T^+_{(i,j)}$.

Each block is a matrix with an array of components. The component in the ath row and the bth column of the (i,j) block of the T^+ matrix is written

$$T^+_{(i,j)_{a,b}} \quad \text{or} \quad T^+_{(i,j)_{\alpha,\beta}} \quad , \tag{18}$$

where α, β are spin component indices like the SU(2) matrix indices that appear in Eq. (16). Indeed, each diagonal block, say $T^+_{(i,i)}$ is the matrix in Eq. (16) for some spin s_i, the spin of the SU(2) irrep for the $T^+_{(i,i)}$ block. It follows that there are $2s_i+1$ rows and columns in the block $T^+_{(i,i)}$.

The pattern of blocks in the entire T^+ matrix is fixed by the blocks $T^+_{(i,i)}$ along the diagonal. Thus, the row and column component indices in a block $T^+_{(i,j)}$, whether on-diagonal $(i=j)$ or off-diagonal $(i \neq j)$, take the values $a = 1, 2, ..., 2s_i+1$ and $b = 1, 2, ..., 2s_j+1$. We choose to put the spin component indices in descending order, so they take the values $\alpha = s_i, s_i-1, ..., -s_i$ and $\beta = s_j, s_j-1, ..., -s_j$. With the two equations

$$a = s_i - \alpha + 1 \quad \text{and} \quad b = s_j - \beta + 1 \quad , \tag{19}$$

one can go from the spin component indices (α, β) to the corresponding row/column indices (a,b), and back.

The component $T^+_{(i,j)_{a,b}}$ in the a^{th} row and the b^{th} column of the (i,j) block of T^+ is in some row r and some column c of the T^+ matrix. One finds that

$$r = a + \sum_{i'=1}^{i-1} (2s_{i'}+1) \quad \text{and} \quad c = b + \sum_{j'=1}^{j-1} \left(2s_{j'}+1\right) \quad , \tag{20}$$

The row index r combines all the rows in the blocks preceeding the i^{th} block plus the a^{th} row in the i^{th} block. Similarly for the columns. The same formula applies to all TUV matrices.

Like SU(2), the components of the TUV matrices of SU(3) are functions of a set of allowed values in a parameter space. Like SU(2), the parameter spaces for SU(3) are discrete, consisting of a finite number of values. Unlike SU(2), there are five parameter spaces for the many SU(3) formulas. Each allowed value (s,m) of the parameters in an SU(3) parameter space has an integer or half-integer s and an integer m. We continue to call s the 'spin', while m is a 'multiplicity index' that distinguishes the copies of each spin value s in the SU(3) parameter space. See Fig. 1.

We turn now to the presentation of the formulas for the TUV matrices.

4 Solutions

The TUV matrices can be calculated with the formulas in this section. The solutions are sorted by shared properties into five collections of matrices or parts of matrices.

The four block-diagonal matrices have much in common and are presented in the first collection. Thus, the first collection consists of the three T-matrices and U^3. The other four matrices, U^+, U^-, V^+, V^-, have components that vanish in the diagonal blocks and are nonzero only in off-diagonal blocks. These four matrices have nonzero components in a line of blocks above the diagonal, called 'Upper', and in a line of blocks below the diagonal, called 'Lower.'

Thus, there are five collections: the collection of the three T-matrices and U^3 that have nonzero components in diagonal blocks, two collections for the Upper and Lower sections of U^+ and V^+ and two more collections for the Upper and Lower sections of U^- and V^-.

Each collection has (A) the (s, m) parameter space, (B) the row and column indices (i, j) of those blocks that may have nonzero components, and (C) the formulas for the potentially nonzero components in the blocks of (B).

For each (i, j) block in (B), the spin component indices α, β take values in descending order, $\alpha = s_i, s_i - 1, ..., -s_i$, and $\beta = s_j, s_j - 1, ..., -s_j$, where s_i and s_j are the T-spins of the (i, j) block. To find the component row/column indices a, b where $a = 1, 2, ..., 2s_i + 1$ and $b = 1, 2, ..., 2s_j + 1$, make use of Eq. 19.

Collection I: $T^3, U^3, T^\pm$.

Top Cap:

(A) Parameters: $2s = 0, ..., q$, each with multiplicity $m = 1, ..., 2s + 1$

(B) Block: $i = s\,(2s + 1) + m$, $j = i$, with T-spins: $(s_i, s_j) = (s, s)$

(C) Components:

$$T^\pm_{(i,j)_{\alpha\beta}} = \sqrt{(s \pm \alpha)(1 + s \mp \alpha)}\delta_{\alpha\mp1,\beta} \tag{21}$$

$$T^3_{(i,j)_{\alpha\beta}} = \alpha\delta_{\alpha,\beta} \quad , \tag{22}$$

$$U^3_{(i,j)_{\alpha\beta}} = \frac{1}{2}\left(-3 - p + q + 3m - 3s - \alpha\right)\delta_{\alpha\beta} \quad , \tag{23}$$

Middle:

(A) Parameters: $2s = q + 1, ..., p$, each with multiplicity $m = 1, ..., q + 1$

(B) Block: $i = \frac{1}{2}\,(4s - q)\,(q + 1) + m$, $j = i$, with T-spins: $(s_i, s_j) = (s, s)$

(C) Components:

$$T^\pm_{(i,j)_{\alpha\beta}} = \sqrt{(s \pm \alpha)(1 + s \mp \alpha)}\delta_{\alpha\mp1,\beta} \tag{24}$$

$$T^3_{(i,j)_{\alpha\beta}} = \alpha\delta_{\alpha,\beta} \quad , \tag{25}$$

$$U^3_{(i,j)_{\alpha\beta}} = \frac{1}{2}\left(-3 - p - 2q + 3m + 3s - \alpha\right)\delta_{\alpha\beta} \quad , \tag{26}$$

Bottom Cap:
(A) Parameters: $2s = p+1, ..., p+q$, each with multiplicity $m = 1, ..., , p+q-2s+1$
(B) Block: $i = \frac{1}{2}\left[(4s-q)(q+1)+(4s-p)(p+1)\right] - s(2s+1) + m$, $j = i$, with T-spins: $(s_i, s_j) = (s, s)$
(C) Components:

$$T^{\pm}_{(i,j)_{\alpha\beta}} = \sqrt{(s \pm \alpha)(1+s \mp \alpha)}\delta_{\alpha \mp 1, \beta} \tag{27}$$

$$T^{3}_{(i,j)_{\alpha\beta}} = \alpha \delta_{\alpha,\beta} \quad , \tag{28}$$

$$U^{3}_{(i,j)_{\alpha\beta}} = \frac{1}{2}\left(-3-p-2q+3m+3s-\alpha\right)\delta_{\alpha\beta} \quad , \tag{29}$$

Collection II: Upper $(i < j)$ blocks of U^+ and V^+.
Top Cap:
(A) Parameters: $2s = 0, ..., q-2$ each with multiplicity $m = 1, ..., 2s+1$
(B) Block: $i = s(2s+1)+m$, $j = i+2s+1$, with T-spins: $(s_i, s_j) = (s, s+1/2)$
(C) Components:

$$U^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{(2-m+2s)(3+p-m+2s)(-1+q+m-2s)(1+s+\alpha)}{2(1+s)(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2,\beta} \quad . \tag{30}$$

$$V^{+}_{(i,j)_{\alpha\beta}} = -\left[\frac{(2-m+2s)(3+p-m+2s)(-1+q+m-2s)(1+s-\alpha)}{2(1+s)(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2,\beta} \quad , \tag{31}$$

Middle:
(A) Parameters: $2s = q-1, ..., p$ each with multiplicity $m = 1, ..., q$
(B) Block: $i = \frac{1}{2}(4s-q)(q+1)+m+1$, $j = i+q$, with T-spins: $(s_i, s_j) = (s, s+1/2)$
(C) Components:

$$U^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{m(1+q-m)(2+p+q-m)(1+s+\alpha)}{2(1+s)(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2,\beta} \tag{32}$$

$$V^{+}_{(i,j)_{\alpha\beta}} = -\left[\frac{m(1+q-m)(2+p+q-m)(1+s-\alpha)}{2(1+s)(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2,\beta} \quad , \tag{33}$$

Bottom Cap:
(A) Parameters: $2s = p+1, ..., p+q-1$ each with multiplicity $m = 1, ..., p+q-2s$
(B) Block: $i = \frac{1}{2}\left[(4s-q)(q+1)+(4s-p)(p+1)\right] - s(2s+1) + m + 1$, $j = i+p+q-2s$, with T-spins: $(s_i, s_j) = (s, s+1/2)$

(C) Components:

$$U^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{m\,(1+q-m)\,(2+p+q-m)\,(1+s+\alpha)}{2\,(1+s)\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2\,,\beta} \tag{34}$$

$$V^{+}_{(i,j)_{\alpha\beta}} = -\left[\frac{m\,(1+q-m)\,(2+p+q-m)\,(1+s-\alpha)}{2\,(1+s)\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2\,,\beta} \quad , \tag{35}$$

Collection III: Lower $(i > j)$ blocks of U^{+} and V^{+}.

Top Cap:

(A) Parameters: $2s = 1, ..., q$ each with multiplicity $m = 1, ..., 2s$

(B) Block: $i = s\,(2s+1) + m + 1$, $j = i - 2s - 1$, with T-spins: $(s_i, s_j) = (s, s-1/2)$

(C) Components:

$$U^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{m\,(1+p-m)\,(1+q+m)\,(s-\alpha)}{2s\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2\,,\beta} \quad , \tag{36}$$

$$V^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{m\,(1+p-m)\,(1+q+m)\,(s+\alpha)}{2s\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2\,,\beta} \quad , \tag{37}$$

Middle:

(A) Parameters: $2s = q+1, ..., p$ each with multiplicity $m = 1, ..., q+1$

(B) Block: $i = \frac{1}{2}\,(4s-q)\,(q+1) + m$, $j = i - q - 1$, with T-spins: $(s_i, s_j) = (s, s-1/2)$

(C) Components:

$$U^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{(m+2s)\,(-1-q+m+2s)\,(2+p+q-m-2s)\,(s-\alpha)}{2s\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2\,,\beta} \quad , \tag{38}$$

$$V^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{(m+2s)\,(-1-q+m+2s)\,(2+p+q-m-2s)\,(s+\alpha)}{2s\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2\,,\beta} \quad , \tag{39}$$

Bottom Cap:

(A) Parameters: $2s = p+1, ..., p+q$ each with multiplicity $m = 1, ..., p+q-2s+1$

(B) Block: $i = \frac{1}{2}\,[(4s-q)\,(q+1) + (4s-p)\,(p+1)] - s\,(2s+1) + m$, $j = i - p - q + 2s - 2$, with T-spins: $(s_i, s_j) = (s, s-1/2)$

(C) Components:

$$U^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{(m+2s)\,(-1-q+m+2s)\,(2+p+q-m-2s)\,(s-\alpha)}{2s\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2\,,\beta} \quad , \tag{40}$$

$$V^{+}_{(i,j)_{\alpha\beta}} = \left[\frac{(m+2s)\,(-1-q+m+2s)\,(2+p+q-m-2s)\,(s+\alpha)}{2s\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2\,,\beta} \quad , \tag{41}$$

Collection IV: Upper $(i<j)$ blocks of U^{-} and V^{-}.

Top Cap:

(A) Parameters: $2s = 0, ..., q-1$ each with multiplicity $m = 1, ..., 2s+1$

(B) Block: $i = m + s + 2s^2$, $j = i + 2 + 2s$, with T-spins: $(s_i, s_j) = (s, s+1/2)$

(C) Components:

$$U^{-}_{(i,j)_{\alpha\beta}} = \left[\frac{m\,(1+p-m)\,(1+q+m)\,(1+s-\alpha)}{2\,(1+s)\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2\,,\beta} \quad , \tag{42}$$

$$V^{-}_{(i,j)_{\alpha\beta}} = \left[\frac{m\,(1+p-m)\,(1+q+m)\,(1+s+\alpha)}{2\,(1+s)\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2\,,\beta} \quad , \tag{43}$$

Middle:

(A) Parameters: $2s = q, ..., p-1$ each with multiplicity $m = 1, ..., q+1$

(B) Block: $i = m - \frac{q}{2} - \frac{q^2}{2} + 2\,(q+1)\,s$, $j = i + 1 + q$, with T-spins: $(s_i, s_j) = (s, s+1/2)$

(C) Components:

$$U^{-}_{(i,j)_{\alpha\beta}} = \left[\frac{(1+m+2s)\,(-q+m+2s)\,(1+p+q-m-2s)\,(1+s-\alpha)}{2\,(1+s)\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2\,,\beta} \quad , \tag{44}$$

$$V^{-}_{(i,j)_{\alpha\beta}} = \left[\frac{(1+m+2s)\,(-q+m+2s)\,(1+p+q-m-2s)\,(1+s+\alpha)}{2\,(1+s)\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2\,,\beta} \quad , \tag{45}$$

Bottom Cap:

(A) Parameters: $2s = p, ..., p+q-1$ each with multiplicity $m = 1, ..., p+q-2s$

(B) Block: $i = m - \frac{p}{2} - \frac{p^2}{2} - \frac{q}{2} - \frac{q^2}{2} + (2p+2q+3)\,s - 2s^2$, $j = i + 1 + p + q - 2s$, with T-spins: $(s_i, s_j) = (s, s+1/2)$

(C) Components:

$$U^{-}_{(i,j)_{\alpha\beta}} = \left[\frac{(1+m+2s)\,(-q+m+2s)\,(1+p+q-m-2s)\,(1+s-\alpha)}{2\,(1+s)\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha-1/2\,,\beta} \quad , \tag{46}$$

$$V^{-}_{(i,j)_{\alpha\beta}} = \left[\frac{(1+m+2s)\,(-q+m+2s)\,(1+p+q-m-2s)\,(1+s+\alpha)}{2\,(1+s)\,(1+2s)}\right]^{\frac{1}{2}} \delta_{\alpha+1/2\,,\beta} \quad , \tag{47}$$

Collection V: Lower ($i > j$) blocks of U^- and V^-.

Top Cap:

(A) Parameters: $2s = 1, ..., q$ each with multiplicity $m = 1, ..., 2s$

(B) Block: $i = m + s + 2s^2$, $j = i - 2s$, with T-spins: $(s_i, s_j) = (s, s - 1/2)$

(C) Components:

$$U^-_{(i,j)_{\alpha\beta}} = \left[\frac{(1 - m + 2s)\,(2 + p - m + 2s)\,(q + m - 2s)\,(s + \alpha)}{2s\,(1 + 2s)}\right]^{\frac{1}{2}} \delta_{\alpha - 1/2\,,\beta} \quad , \tag{48}$$

$$V^-_{(i,j)_{\alpha\beta}} = -\left[\frac{(1 - m + 2s)\,(2 + p - m + 2s)\,(q + m - 2s)\,(s - \alpha)}{2s\,(1 + 2s)}\right]^{\frac{1}{2}} \delta_{\alpha + 1/2\,,\beta} \quad , \tag{49}$$

Middle:

(A) Parameters: $2s = q + 1, ..., p$ each with multiplicity $m = 1, ..., q$

(B) Block: $i = m - \frac{q}{2} - \frac{q^2}{2} + 2\,(q + 1)\,s$, $j = i - q$, with T-spins: $(s_i, s_j) = (s, s - 1/2)$

(C) Components:

$$U^-_{(i,j)_{\alpha\beta}} = \left[\frac{m\,(1 + q - m)\,(2 + p + q - m)\,(s + \alpha)}{2s\,(1 + 2s)}\right]^{\frac{1}{2}} \delta_{\alpha - 1/2\,,\beta} \quad , \tag{50}$$

$$V^-_{(i,j)_{\alpha\beta}} = -\left[\frac{m\,(1 + q - m)\,(2 + p + q - m)\,(s - \alpha)}{2s\,(1 + 2s)}\right]^{\frac{1}{2}} \delta_{\alpha + 1/2\,,\beta} \quad , \tag{51}$$

Bottom Cap:

(A) Parameters: $2s = p + 1, ..., p + q$ each with multiplicity $m = 1, ..., p + q - 2s + 1$

(B) Block: $i = m - \frac{p}{2} - \frac{p^2}{2} - \frac{q}{2} - \frac{q^2}{2} + (2p + 2q + 3)\,s - 2s^2$, $j = i - p - q + 2s - 1$, with T-spins: $(s_i, s_j) = (s, s - 1/2)$

(C) Components:

$$U^-_{(i,j)_{\alpha\beta}} = \left[\frac{m\,(1 + q - m)\,(2 + p + q - m)\,(s + \alpha)}{2s\,(1 + 2s)}\right]^{\frac{1}{2}} \delta_{\alpha - 1/2\,,\beta} \quad , \tag{52}$$

$$V^-_{(i,j)_{\alpha\beta}} = -\left[\frac{m\,(1 + q - m)\,(2 + p + q - m)\,(s - \alpha)}{2s\,(1 + 2s)}\right]^{\frac{1}{2}} \delta_{\alpha + 1/2\,,\beta} \quad , \tag{53}$$

where $\alpha = s, s - 1, ..., -s$, in descending order.

The (i, j) blocks in Collection I are square matrices aligned along the diagonal. That implies $i = j$ and $s_i = s_j$. The other collections have (i, j) blocks that are rectangular, with T-spins s_i and s_j differing by a half, $s_j = s_i \pm 1/2$. By comparing $j - i$ with the maximum multiplicity, max m, one sees that the difference $s_j - s_i = \pm 1/2$ is built into the formulas.

Recall that the formulas in Collections I – V are valid for those (p, q) irreps with $p \geq q$. Their use for the irreps with $p < q$ requires extra steps.

Irreps with $p < q$.

For a (p, q) irrep with $p < q$, the TUV matrices can be found from the preceding process by taking advantage of the relationship between (q, p) and (p, q) irreps: The negative transpose of the TUV matrices for a (q, p) irrep gives a representation of the (p, q) irrep. [11]

Thus, to get an irrep with $p < q$, we first have p and q exchange places. Define $p' = q$ and q' $= p$ so that $p' > q'$. Proceed as above to obtain the T′U′V′ matrices for the (p', q') irrep, which has $p' > q'$. By the relationship between (p, q) and (q, p) irreps, the TUV matrices for (p, q), with $p < q$, are the negative transposes of the $p' > q'$ T′U′V′matrices. In this way, we can obtain the TUV matrices for the $p < q$ irreps.

We have shown how to calculate a set of TUV matrices for the (p, q) irrep of the $\mathfrak{su}(3)$ algebra for any pair of nonnegative integers (p, q). That completes the work.

5 Discussion

As presented here, the validity of the process relies on calculating sets of TUV matrices and showing that they obey the $\mathfrak{su}(3)$ algebra. This has been done for (p, q) irreps with $p+q \leq 15$. An algebraic proof of validity for any (p, q) irrep would be preferable. Such proof may be the subject of another article.

It should be mentioned, however, that four of the eight matrix solutions in Sec. 4 need not be derived, because they can be considered as already given in the literature. The four given matrices are the matrices $T^3, U^3, T^{\pm}$ in Collection I. The list of T-spins for the (p, q) irrep and the list of U^3 eigenvalues, often called 'weights', can be found in, for example, Ref. [11]. Since U^3 is a diagonal matrix, its eigenvalues make the matrix. Given the T-spin list and formulas Eq. (16), the T-matrices of the $\mathfrak{su}(3)$ algebra are quickly constructed. We, therefore, consider the matrices $T^3, U^3, T^{\pm}$ as given in the literature. Thus, an algebraic derivation or verification of the formulas for the matrices U^+, U^-, V^+, and V^- is lacking.

As a practical matter, those using the formulas are advised to apply several checks. At minimum, ensure the TUV matrices satisfy the commutation relations. For other suggested checks, consider calculating the trace and hermiticity of the matrices F^i and compare the Casimir invariant with its well-known formula. Some of those checks are coded in the computer programs in Sec. 6 and Sec. 7.

6 Mathematica Notebook

The Mathematica [12] notebook copied below calculates the eight basis matrices $T^{\pm}$, T^3, $U^{\pm}$, U^3, and $V^{\pm}$ for the (p,q) irrep of the $\mathfrak{su}(3)$ algebra. The notebook checks that the eight matrices obey the $\mathfrak{su}(3)$ algebra and the quadratic Casimir invariance equation. The eight traceless, hermitian generator matrices F^i, $i \in \{1, 2, ..., 8\}$, are also calculated.

Please note that the TUV matrices for $(p, q) = (1, 0)$ calculated with the computer program gives eight 3×3 complex matrices F^i that are equivalent to the matrices in Eq. (5). The two equivalent sets of matrices are related by a similarity transformation and the two sets obey the same commutation relations.

The Mathematica notebook was copied verbatim into Latex, which turns σ into '[Sigma]', for example. Exporting the pdf to make a ready-to-run notebook may require some adjustments to be successful. Such issues may be avoided by following links to the ready-to-run file that are provided in Refs. [13] and [14].

The Mathematica Notebook

```
(*Options*)
(*font: Courier New, 9 point*)
SetOptions[EvaluationNotebook[], ShowCellLabel -> False];
SetOptions[EvaluationNotebook[], PageWidth -> 540];
Unprotect[Up];

(*SU(3) Matrices, a notebook by R. Shurtleff, August 2024,*)(*
Department of Applied Math and Sciences,Wentworth Institute of \
Technology,Boston MA 02115, shurtleffr@wit.edu, \
momentummatrix@yahoo.com *)
(*Links to this notebook in a ready-to-run file:
https://www.wolframcloud.com/obj/shurtleffr/Published/\
SU3MatricesMMA2024.nb ,
https://www.dropbox.com/scl/fi/ey0wk7cvhupeqeeeowr1v/MMAforPAPER3.nb?\
rlkey=7jhigi9m8lnxo862c78jdznw6&st=hkuoc12g&dl=0*)

(*This Mathematica notebook [1,2] replaces previous such notebooks. \
[3]*)


(*Contents*)
(*Part 0. Introduction*)
(*Part I. Formulas for TUV matrices.*)
(*Part II. A Numerical Example.*)
(*References*)

(*Introduction
Part I has the formulas for the 8 basis matrices \
```

```
{Tp,Tm,T3,Up,Um,U3,Vp,Vm} for the (p,q) irrep of the su(3) algebra. \
The commutation relations of the algebra
can be read from the Print statement at the end of Part II.*)
(* Tp,Tm,T3 are SU(2) matrix generators,reduced with nonzero \
components confined to blocks along the matrix diagonal; each block \
is a matrix. All TUV matrices share the block structure of the \
Tp,Tm,T3 matrices.
Each set of formulas has a parameter space (s,m), formulas for the \
row and column indices (i,j) of the blocks and formulas for the \
components of those blocks. *)
(*Notes: 1. 'p' in Tp, Up, Vp,...  stands for +(plus) and m stands \
for -(minus).*)
(*2. The symbol 'p' also stands for one of the two integers (p,q)
that identifies an irrep. *)
(*3. The symbol 'm' also denotes the index m that counts the many \
copies
of the spin s in the parameter space (s,m). *)
(*4. The notation mimics standard particle physics texts, see e.g. \
Ref. 4,5.*)

(*In Part II, the formulas of Part I are used to construct the \
matrices. The notebook calculates the 8 basis matrices \
{Tp,Tm,T3,Up,Um,U3,Vp,Vm} for the (p,q) irrep of the su(3) algebra.  *)
(*The values of p and q are entered at the start of Part II.*)
(*The eight F^i traceless, hermitian matrix generators are calculated
 at the end of Part II.*)
(*Towards the end of Part II there are checks that the calculated TUV \
matrices obey the su(3) algebra and that they give the correct \
quadratic Casimer operator.*)


(*Tp,Tm,T3 are reducible SU(2) matrix generators, reduced to su(2) \
irreps in blocks along the matrix diagonal.*)
(* Tspins-list of SU(2) spins runs from small to large, s=0 to s=p+q.*)
(*Tspin matrix irreps determines the block structure of the TUV \
matrices.*)(*The T-spin list has structure: {top cap,middle,bottom \
cap} named by*)
(*the location in the TUV matrix.   *)
(*The ij block is a matrix with 2Subscript[s, i]+1 rows and \
2Subscript[s, j]+1 columns where
```

```
s_i is the ith T-spin and s_j is the jth T-spin.*)

(*The T matrices and U3 are block diagonal matrices, i=j.*)
(*The Up,Um,Vp,Vm matrices have blocks above and below the diagonal,
'Upper'(i<j) and 'Lower'(i>j).*)

(*Part 1.  Formulas for TUV matrices. T3, U3, Tp, Tm, Up, Um, Vp, Vm*)


(*The formulas for (i,j) block indices and the (\[Alpha],\[Beta]) \
block components
are parametric functions of individual sets of spins s and an index m \
of copies of s for each value of s.*)
(*The 'Top Cap' blocks appear in the upper left portion of the \
matrix.*)
(*The 'Middle' section of the (s,m) parameter space has blocks along \
the middle diagonal region of the TUV matrix.*)
(*The 'Bottom Cap' blocks are located along the lower right diagonal.*)


(*Definitions:
  1. s2 - double the spin s, s2 = 2s, always an integer
2. (irow,jcol) - indices for the block rows and columns
3. f - formula for components in a block
4. (pp,qq) - variable version of the irrep designation (p,q)
5. (ss2,mt) - variable version of the (s,m) parameters
*)


(*x = 1 indicates matrix X = T3. *)
(*T3      1 -  Top Cap:*)
s2Min[1, 1, qq_, pp_] := 0; s2Max[1, 1, qq_, pp_] := qq;
mMin[1, 1, qq_, pp_, s2_] := 1; mMax[1, 1, qq_, pp_, s2_] := s2 + 1;
irow[1, 1, qq_, pp_, s2_, m_] := s2 (s2 + 1)/2 + m;
jcol[1, 1, qq_, pp_, s2_, m_] := irow[1, 1, qq, pp, s2, m];
f[1, 1, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (\[Alpha] KroneckerDelta[\[Beta] - \
\[Alpha]]);

(*T3       2-Middle:*)
```

```
s2Min[1, 2, qq_, pp_] := qq + 1; s2Max[1, 2, qq_, pp_] := pp;
mMin[1, 2, qq_, pp_, s2_] := 1; mMax[1, 2, qq_, pp_, s2_] := qq + 1;
irow[1, 2, qq_, pp_, s2_, m_] := (1/2) (2 s2 - qq) (qq + 1) + m;
jcol[1, 2, qq_, pp_, s2_, m_] := irow[1, 2, qq, pp, s2, m];
f[1, 2, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (\[Alpha] KroneckerDelta[\[Beta] - \
\[Alpha]]);

(*T3       3-Bottom Cap*)
s2Min[1, 3, qq_, pp_] := pp + 1; s2Max[1, 3, qq_, pp_] := pp + qq;
mMin[1, 3, qq_, pp_, s2_] := 1;
mMax[1, 3, qq_, pp_, s2_] := pp + qq - s2 + 1;
irow[1, 3, qq_, pp_, s2_,
   m_] := (1/2) ((2 s2 - qq) (qq + 1) + (2 s2 - pp) (pp + 1) -
      s2 (s2 + 1)) + m;
jcol[1, 3, qq_, pp_, s2_, m_] := irow[1, 3, qq, pp, s2, m];
f[1, 3, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (\[Alpha] KroneckerDelta[\[Beta] - \
\[Alpha]]);

(*x = 2 indicates matrix X = U3. *)
(*U3       1-Top Cap:*)
s2Min[2, 1, qq_, pp_] := 0; s2Max[2, 1, qq_, pp_] := qq;
mMin[2, 1, qq_, pp_, s2_] := 1; mMax[2, 1, qq_, pp_, s2_] := s2 + 1;
irow[2, 1, qq_, pp_, s2_, m_] := s2 (s2 + 1)/2 + m;
jcol[2, 1, qq_, pp_, s2_, m_] := irow[2, 1, qq, pp, s2, m];
f[2, 1, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((-3 - pp + qq + 3 m -
       3 s2/2 - \[Alpha])/2 KroneckerDelta[\[Beta] - \[Alpha]]);

(*U3       2-Middle:*)
s2Min[2, 2, qq_, pp_] := qq + 1; s2Max[2, 2, qq_, pp_] := pp;
mMin[2, 2, qq_, pp_, s2_] := 1; mMax[2, 2, qq_, pp_, s2_] := qq + 1;
irow[2, 2, qq_, pp_, s2_, m_] := (1/2) (2 s2 - qq) (qq + 1) + m;
jcol[2, 2, qq_, pp_, s2_, m_] := irow[2, 2, qq, pp, s2, m];
f[2, 2, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((-3 - pp - 2 qq + 3 m +
        3 s2/2 - \[Alpha])/2) KroneckerDelta[\[Beta] - \[Alpha]]);

(*U3       3-Bottom Cap*)
```

```
s2Min[2, 3, qq_, pp_] := pp + 1; s2Max[2, 3, qq_, pp_] := pp + qq;
mMin[2, 3, qq_, pp_, s2_] := 1;
mMax[2, 3, qq_, pp_, s2_] := pp + qq - s2 + 1;
irow[2, 3, qq_, pp_, s2_,
   m_] := (1/2) ((2 s2 - qq) (qq + 1) + (2 s2 - pp) (pp + 1) -
      s2 (s2 + 1)) + m;
jcol[2, 3, qq_, pp_, s2_, m_] := irow[2, 3, qq, pp, s2, m];
f[2, 3, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((-3 - pp - 2 qq + 3 m +
        3 s2/2 - \[Alpha])/2) KroneckerDelta[\[Beta] - \[Alpha]]);

(*x = 3 indicates matrix X = Tp. *)
(*Tp       1-Top Cap:*)
s2Min[3, 1, qq_, pp_] := 0; s2Max[3, 1, qq_, pp_] := qq;
mMin[3, 1, qq_, pp_, s2_] := 1; mMax[3, 1, qq_, pp_, s2_] := s2 + 1;
irow[3, 1, qq_, pp_, s2_, m_] := s2 (s2 + 1)/2 + m;
jcol[3, 1, qq_, pp_, s2_, m_] := irow[3, 1, qq, pp, s2, m];
f[3, 1, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((s2/2 + \[Alpha]) (1 +
        s2/2 - \[Alpha]))^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] - 1)]);

(*Tp       2-Middle:*)
s2Min[3, 2, qq_, pp_] := qq + 1; s2Max[3, 2, qq_, pp_] := pp;
mMin[3, 2, qq_, pp_, s2_] := 1; mMax[3, 2, qq_, pp_, s2_] := qq + 1;
irow[3, 2, qq_, pp_, s2_, m_] := (1/2) (2 s2 - qq) (qq + 1) + m;
jcol[3, 2, qq_, pp_, s2_, m_] := irow[3, 2, qq, pp, s2, m];
f[3, 2, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((s2/2 + \[Alpha]) (1 +
        s2/2 - \[Alpha]))^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] - 1)]);

(*Tp       3-Bottom Cap*)
s2Min[3, 3, qq_, pp_] := pp + 1; s2Max[3, 3, qq_, pp_] := pp + qq;
mMin[3, 3, qq_, pp_, s2_] := 1;
mMax[3, 3, qq_, pp_, s2_] := pp + qq - s2 + 1;
irow[3, 3, qq_, pp_, s2_,
   m_] := (1/2) ((2 s2 - qq) (qq + 1) + (2 s2 - pp) (pp + 1) -
      s2 (s2 + 1)) + m;
jcol[3, 3, qq_, pp_, s2_, m_] := irow[3, 3, qq, pp, s2, m];
```

```
f[3, 3, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((s2/2 + \[Alpha]) (1 +
        s2/2 - \[Alpha]))^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] - 1)]);

(*x = 4 indicates matrix X = Tm. *)
(*Tm       1-Top Cap:*)
s2Min[4, 1, qq_, pp_] := 0; s2Max[4, 1, qq_, pp_] := qq;
mMin[4, 1, qq_, pp_, s2_] := 1; mMax[4, 1, qq_, pp_, s2_] := s2 + 1;
irow[4, 1, qq_, pp_, s2_, m_] := s2 (s2 + 1)/2 + m;
jcol[4, 1, qq_, pp_, s2_, m_] := irow[4, 1, qq, pp, s2, m];
f[4, 1, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((s2/2 - \[Alpha]) (1 +
        s2/2 + \[Alpha]))^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] + 1)]);


(*Tm       2-Middle:*)
s2Min[4, 2, qq_, pp_] := qq + 1; s2Max[4, 2, qq_, pp_] := pp;
mMin[4, 2, qq_, pp_, s2_] := 1; mMax[4, 2, qq_, pp_, s2_] := qq + 1;
irow[4, 2, qq_, pp_, s2_, m_] := (1/2) (2 s2 - qq) (qq + 1) + m;
jcol[4, 2, qq_, pp_, s2_, m_] := irow[4, 2, qq, pp, s2, m];
f[4, 2, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] :=  (((s2/2 - \[Alpha]) (1 +
        s2/2 + \[Alpha]))^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] + 1)]);

(*Tm       3-Bottom Cap*)
s2Min[4, 3, qq_, pp_] := pp + 1; s2Max[4, 3, qq_, pp_] := pp + qq;
mMin[4, 3, qq_, pp_, s2_] := 1;
mMax[4, 3, qq_, pp_, s2_] := pp + qq - s2 + 1;
irow[4, 3, qq_, pp_, s2_,
   m_] := (1/2) ((2 s2 - qq) (qq + 1) + (2 s2 - pp) (pp + 1) -
      s2 (s2 + 1)) + m;
jcol[4, 3, qq_, pp_, s2_, m_] := irow[4, 3, qq, pp, s2, m];
f[4, 3, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((s2/2 - \[Alpha]) (1 +
        s2/2 + \[Alpha]))^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] + 1)]);
```

```
(*x = 5 indicates matrix X = Up. *)
(*Up       1-Upper Top Cap:*)
s2Min[5, 1, qq_, pp_] := 0; s2Max[5, 1, qq_, pp_] := qq - 2;
mMin[5, 1, qq_, pp_, s2_] := 1; mMax[5, 1, qq_, pp_, s2_] := s2 + 1;
irow[5, 1, qq_, pp_, s2_, m_] := s2 (s2 + 1)/2 + m ;
jcol[5, 1, qq_, pp_, s2_, m_] := irow[5, 1, qq, pp, s2, m] + s2 + 1;
f[5, 1, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((2 - m + s2) (3 + pp - m +
        s2) (-1 + qq + m - s2)/(2 + s2))^(1/2)
     Sqrt[ (1 + s2/2 + \[Alpha])/(1 + s2)]
     KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Up       2-Upper Middle:*)
s2Min[5, 2, qq_, pp_] := qq - 1; s2Max[5, 2, qq_, pp_] := pp;
mMin[5, 2, qq_, pp_, s2_] := 1; mMax[5, 2, qq_, pp_, s2_] := qq;
irow[5, 2, qq_, pp_, s2_, m_] := (1/2) (2 s2 - qq) (qq + 1) + m + 1 ;
jcol[5, 2, qq_, pp_, s2_, m_] := irow[5, 2, qq, pp, s2, m] + qq;
f[5, 2, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((m (1 + qq -
        m) (2 + pp + qq - m)/(2 + s2) )^(1/2)
     Sqrt[ (1 + s2/2 + \[Alpha])/(1 + s2)]
     KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Up       3-Upper Bottom Cap:*)
s2Min[5, 3, qq_, pp_] := pp + 1;
s2Max[5, 3, qq_, pp_] := pp + qq - 1;
mMin[5, 3, qq_, pp_, s2_] := 1;
mMax[5, 3, qq_, pp_, s2_] := pp + qq - s2;
irow[5, 3, qq_, pp_, s2_,
   m_] := (1/2) ((2 s2 - qq) (qq + 1) + (2 s2 - pp) (pp + 1) -
      s2 (s2 + 1)) + m + 1 ;
jcol[5, 3, qq_, pp_, s2_, m_] :=
  irow[5, 3, qq, pp, s2, m] + pp + qq - s2;
f[5, 3, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((m (1 + qq -
        m) (2 + pp + qq - m)/(2 + s2) )^(1/2)
     Sqrt[ (1 + s2/2 + \[Alpha])/(1 + s2)]
     KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Up       4-Lower Top Cap:*)
```

```
s2Min[5, 4, qq_, pp_] := 1; s2Max[5, 4, qq_, pp_] := qq;
mMin[5, 4, qq_, pp_, s2_] := 1; mMax[5, 4, qq_, pp_, s2_] := s2;
irow[5, 4, qq_, pp_, s2_, m_] := s2 (s2 + 1)/2 + m + 1;
jcol[5, 4, qq_, pp_, s2_, m_] := irow[5, 4, qq, pp, s2, m] - s2 - 1;
f[5, 4, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((m (1 + pp -
        m) (1 + qq + m)/(s2 (1 + s2))  )^(1/2) Sqrt[s2/2 - \[Alpha]]
     KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Up       5-Lower Middle:*)
s2Min[5, 5, qq_, pp_] := qq + 1; s2Max[5, 5, qq_, pp_] := pp;
mMin[5, 5, qq_, pp_, s2_] := 1; mMax[5, 5, qq_, pp_, s2_] := qq + 1;
irow[5, 5, qq_, pp_, s2_, m_] := (1/2) (2 s2 - qq) (qq + 1) + m;
jcol[5, 5, qq_, pp_, s2_, m_] := irow[5, 5, qq, pp, s2, m] - qq - 1;
f[5, 5, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((m + s2) (-1 - qq + m +
        s2) (2 + pp + qq - m - s2)/(s2 (1 + s2)) )^(1/2) Sqrt[
    s2/2 - \[Alpha]] KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Up       6-Lower Bottom Cap:*)
s2Min[5, 6, qq_, pp_] := pp + 1; s2Max[5, 6, qq_, pp_] := pp + qq;
mMin[5, 6, qq_, pp_, s2_] := 1;
mMax[5, 6, qq_, pp_, s2_] := pp + qq - s2 + 1;
irow[5, 6, qq_, pp_, s2_,
   m_] := (1/2) ((2 s2 - qq) (qq + 1) + (2 s2 - pp) (pp + 1) -
      s2 (s2 + 1)) + m;
jcol[5, 6, qq_, pp_, s2_, m_] :=
  irow[5, 6, qq, pp, s2, m] - pp - qq + s2 - 2;
f[5, 6, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((m + s2) (-1 - qq + m +
        s2) (2 + pp + qq - m - s2)/(s2 (1 + s2)) )^(1/2) Sqrt[
    s2/2 - \[Alpha]] KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*x = 6 indicates matrix X = Um. *)
(*Um       1-Upper Top Cap:*)
s2Min[6, 1, qq_, pp_] := 0; s2Max[6, 1, qq_, pp_] := qq - 1;
mMin[6, 1, qq_, pp_, s2_] := 1; mMax[6, 1, qq_, pp_, s2_] := s2 + 1;
irow[6, 1, qq_, pp_, s2_, m_] := (s2 + s2^2)/2 + m;
jcol[6, 1, qq_, pp_, s2_, m_] := irow[6, 1, qq, pp, s2, m] + 2 + s2;
f[6, 1, qq_, pp_, s2_,
```

```
   m_, \[Alpha]_, \[Beta]_] := ((m (1 + pp -
        m) (1 + qq + m)/((2 + s2) (1 + s2))  )^(1/2) Sqrt[
    1 + s2/2 - \[Alpha]] KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Um       2-Upper Middle:*)
s2Min[6, 2, qq_, pp_] := qq; s2Max[6, 2, qq_, pp_] := pp - 1;
mMin[6, 2, qq_, pp_, s2_] := 1; mMax[6, 2, qq_, pp_, s2_] := qq + 1;
irow[6, 2, qq_, pp_, s2_, m_] := -((qq + qq^2)/2) + s2 (qq + 1) + m;
jcol[6, 2, qq_, pp_, s2_, m_] := irow[6, 2, qq, pp, s2, m] + qq + 1;
f[6, 2, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((1 + m + s2) (-qq + m +
        s2) (1 + pp + qq - m - s2)/((2 + s2) (1 + s2)) )^(1/2) Sqrt[
    1 + s2/2 - \[Alpha]] KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Um      3-Upper Bottom Cap:*)
s2Min[6, 3, qq_, pp_] := pp; s2Max[6, 3, qq_, pp_] := pp + qq - 1;
mMin[6, 3, qq_, pp_, s2_] := 1;
mMax[6, 3, qq_, pp_, s2_] := pp + qq - s2;
irow[6, 3, qq_, pp_, s2_,
   m_] := -(1/2) (pp + pp^2 + qq + qq^2 + s2^2) +
   s2 (pp + qq + 3/2) + m;
jcol[6, 3, qq_, pp_, s2_, m_] :=
  irow[6, 3, qq, pp, s2, m] + 1 + pp + qq - s2;
f[6, 3, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((1 + m + s2) (-qq + m +
        s2) (1 + pp + qq - m - s2)/((2 + s2) (1 + s2)) )^(1/2) Sqrt[
    1 + s2/2 - \[Alpha]] KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Um       4-Lower Top Cap:*)
s2Min[6, 4, qq_, pp_] := 1; s2Max[6, 4, qq_, pp_] := qq;
mMin[6, 4, qq_, pp_, s2_] := 1; mMax[6, 4, qq_, pp_, s2_] := s2;
irow[6, 4, qq_, pp_, s2_, m_] := (1/2) (s2 + s2^2) + m ;
jcol[6, 4, qq_, pp_, s2_, m_] := irow[6, 4, qq, pp, s2, m] - s2;
f[6, 4, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((1 - m + s2) (2 + pp - m +
        s2) (qq + m - s2)/(1 + s2))^(1/2) Sqrt[ (s2/2 + \[Alpha])/s2]
     KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Um       5-Lower Middle:  *)
s2Min[6, 5, qq_, pp_] := qq + 1; s2Max[6, 5, qq_, pp_] := pp;
```

```
mMin[6, 5, qq_, pp_, s2_] := 1; mMax[6, 5, qq_, pp_, s2_] := qq;
irow[6, 5, qq_, pp_, s2_, m_] := -(1/2) (qq + qq^2) + s2 (qq + 1) +
   m;
jcol[6, 5, qq_, pp_, s2_, m_] := irow[6, 5, qq, pp, s2, m] - qq;
f[6, 5, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((m (1 + qq -
        m) (2 + pp + qq - m)/(1 + s2) )^(1/2)
     Sqrt[ (s2/2 + \[Alpha])/s2]
     KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Um       6-Lower Bottom Cap:*)
s2Min[6, 6, qq_, pp_] := pp + 1; s2Max[6, 6, qq_, pp_] := pp + qq;
mMin[6, 6, qq_, pp_, s2_] := 1;
mMax[6, 6, qq_, pp_, s2_] := pp + qq - s2 + 1;
irow[6, 6, qq_, pp_, s2_,
   m_] := -(1/2) (pp + pp^2 + qq + qq^2 + s2^2) +
   s2 (pp + qq + 3/2) + m;
jcol[6, 6, qq_, pp_, s2_, m_] :=
  irow[6, 6, qq, pp, s2, m] - 1 - pp - qq + s2;
f[6, 6, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((m (1 + qq -
        m) (2 + pp + qq - m)/(1 + s2) )^(1/2)
     Sqrt[ (s2/2 + \[Alpha])/s2]
     KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*x = 7 indicates matrix X = Vp. *)
(*Vp       1-Upper Top Cap:*)
s2Min[7, 1, qq_, pp_] := 0; s2Max[7, 1, qq_, pp_] := qq - 2;
mMin[7, 1, qq_, pp_, s2_] := 1; mMax[7, 1, qq_, pp_, s2_] := s2 + 1;
irow[7, 1, qq_, pp_, s2_, m_] := s2 (s2 + 1)/2 + m ;
jcol[7, 1, qq_, pp_, s2_, m_] := irow[7, 1, qq, pp, s2, m] + s2 + 1;
f[7, 1, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (-(((2 - m + s2) (3 + pp - m +
           s2) (-1 + qq + m - s2) (1 + s2/2 - \[Alpha]))/((2 +
           s2) (1 + s2)))^(1/2)
      KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(* Vp      2 - Upper Middle:*)
s2Min[7, 2, qq_, pp_] := qq - 1; s2Max[7, 2, qq_, pp_] := pp;
mMin[7, 2, qq_, pp_, s2_] := 1; mMax[7, 2, qq_, pp_, s2_] := qq;
```

```
irow[7, 2, qq_, pp_, s2_, m_] := (1/2) (2 s2 - qq) (qq + 1) + m + 1 ;
jcol[7, 2, qq_, pp_, s2_, m_] := irow[7, 2, qq, pp, s2, m] + qq;
f[7, 2, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (-(m (1 + qq - m) (2 + pp + qq -
         m) (1 + s2/2 - \[Alpha])/((2 + s2) (1 + s2)) )^(1/2)
      KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Vp       3-Upper Bottom Cap:*)
s2Min[7, 3, qq_, pp_] := pp + 1;
s2Max[7, 3, qq_, pp_] := pp + qq - 1;
mMin[7, 3, qq_, pp_, s2_] := 1;
mMax[7, 3, qq_, pp_, s2_] := pp + qq - s2;
irow[7, 3, qq_, pp_, s2_,
   m_] := (1/2) ((2 s2 - qq) (qq + 1) + (2 s2 - pp) (pp + 1) -
      s2 (s2 + 1)) + m + 1 ;
jcol[7, 3, qq_, pp_, s2_, m_] :=
  irow[7, 3, qq, pp, s2, m] + pp + qq - s2;
f[7, 3, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] :=  (-(m (1 + qq - m) (2 + pp + qq -
         m) (1 + s2/2 - \[Alpha])/((2 + s2) (1 + s2)) )^(1/2)
      KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Vp       4-Lower Top Cap:*)
s2Min[7, 4, qq_, pp_] := 1; s2Max[7, 4, qq_, pp_] := qq;
mMin[7, 4, qq_, pp_, s2_] := 1; mMax[7, 4, qq_, pp_, s2_] := s2;
irow[7, 4, qq_, pp_, s2_, m_] := s2 (s2 + 1)/2 + m + 1;
jcol[7, 4, qq_, pp_, s2_, m_] := irow[7, 4, qq, pp, s2, m] - s2 - 1;
f[7, 4, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((m (1 + pp - m) (1 + qq +
        m) (s2/2 + \[Alpha])/(s2 (1 + s2))  )^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Vp       5-Lower Middle:*)
s2Min[7, 5, qq_, pp_] := qq + 1; s2Max[7, 5, qq_, pp_] := pp;
mMin[7, 5, qq_, pp_, s2_] := 1; mMax[7, 5, qq_, pp_, s2_] := qq + 1;
irow[7, 5, qq_, pp_, s2_, m_] := (1/2) (2 s2 - qq) (qq + 1) + m;
jcol[7, 5, qq_, pp_, s2_, m_] := irow[7, 5, qq, pp, s2, m] - qq - 1;
f[7, 5, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((m + s2) (-1 - qq + m + s2) (2 + pp +
         qq - m - s2) (s2/2 + \[Alpha])/(s2 (1 + s2)) )^(1/2)
```

```
     KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*Vp       6-Lower Bottom Cap:*)
s2Min[7, 6, qq_, pp_] := pp + 1; s2Max[7, 6, qq_, pp_] := pp + qq;
mMin[7, 6, qq_, pp_, s2_] := 1;
mMax[7, 6, qq_, pp_, s2_] := pp + qq - s2 + 1;
irow[7, 6, qq_, pp_, s2_,
   m_] := (1/2) ((2 s2 - qq) (qq + 1) + (2 s2 - pp) (pp + 1) -
      s2 (s2 + 1)) + m;
jcol[7, 6, qq_, pp_, s2_, m_] :=
  irow[7, 6, qq, pp, s2, m] - pp - qq + s2 - 2;
f[7, 6, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (((m + s2) (-1 - qq + m + s2) (2 + pp +
         qq - m - s2) (s2/2 + \[Alpha])/(s2 (1 + s2)) )^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] - 1/2)]);

(*x = 8 indicates matrix X = Vm. *)
(*Vm       1-Upper Top Cap:*)
s2Min[8, 1, qq_, pp_] := 0; s2Max[8, 1, qq_, pp_] := qq - 1;
mMin[8, 1, qq_, pp_, s2_] := 1; mMax[8, 1, qq_, pp_, s2_] := s2 + 1;
irow[8, 1, qq_, pp_, s2_, m_] := (s2 + s2^2)/2 + m;
jcol[8, 1, qq_, pp_, s2_, m_] := irow[8, 1, qq, pp, s2, m] + 2 + s2;
f[8, 1, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((m (1 + pp - m) (1 + qq +
        m) (1 + s2/2 + \[Alpha])/((2 + s2) (1 + s2))  )^(1/2)
     KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Vm       2-Upper Middle:*)
s2Min[8, 2, qq_, pp_] := qq; s2Max[8, 2, qq_, pp_] := pp - 1;
mMin[8, 2, qq_, pp_, s2_] := 1; mMax[8, 2, qq_, pp_, s2_] := qq + 1;
irow[8, 2, qq_, pp_, s2_, m_] := -(qq + qq^2)/2 + s2 (qq + 1) + m;
jcol[8, 2, qq_, pp_, s2_, m_] := irow[8, 2, qq, pp, s2, m] + qq + 1;
f[8, 2, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((((1 + m + s2) (-qq + m + s2) (1 +
          pp + qq - m - s2) (1 + s2/2 + \[Alpha]))/((2 + s2) (1 +
          s2)) )^(1/2) KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Vm       3-Upper Bottom Cap:*)
s2Min[8, 3, qq_, pp_] := pp; s2Max[8, 3, qq_, pp_] := pp + qq - 1;
mMin[8, 3, qq_, pp_, s2_] := 1;
```

```
mMax[8, 3, qq_, pp_, s2_] := pp + qq - s2;
irow[8, 3, qq_, pp_, s2_,
   m_] := -(1/2) (pp + pp^2 + qq + qq^2 + s2^2) +
   s2 (pp + qq + 3/2) + m;
jcol[8, 3, qq_, pp_, s2_, m_] :=
  irow[8, 3, qq, pp, s2, m] + 1 + pp + qq - s2;
f[8, 3, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := ((((1 + m + s2) (-qq + m + s2) (1 +
          pp + qq - m - s2) (1 + s2/2 + \[Alpha]))/((2 + s2) (1 +
          s2)) )^(1/2) KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Vm       4-Lower Top Cap:*)
s2Min[8, 4, qq_, pp_] := 1; s2Max[8, 4, qq_, pp_] := qq;
mMin[8, 4, qq_, pp_, s2_] := 1; mMax[8, 4, qq_, pp_, s2_] := s2;
irow[8, 4, qq_, pp_, s2_, m_] := (1/2) (s2 + s2^2) + m ;
jcol[8, 4, qq_, pp_, s2_, m_] := irow[8, 4, qq, pp, s2, m] - s2;
f[8, 4, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (-((1 - m + s2) (2 + pp - m + s2) (qq +
          m - s2) (s2/2 - \[Alpha])/(s2 (1 + s2)))^(1/2)
      KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Vm       5-Lower Middle:*)
s2Min[8, 5, qq_, pp_] := qq + 1; s2Max[8, 5, qq_, pp_] := pp;
mMin[8, 5, qq_, pp_, s2_] := 1; mMax[8, 5, qq_, pp_, s2_] := qq;
irow[8, 5, qq_, pp_, s2_, m_] := -(1/2) (qq + qq^2) + s2 (qq + 1) +
   m;
jcol[8, 5, qq_, pp_, s2_, m_] := irow[8, 5, qq, pp, s2, m] - qq;
f[8, 5, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (-(m (1 + qq - m) (2 + pp + qq -
         m) (s2/2 - \[Alpha])/(s2 (1 + s2)) )^(1/2)
      KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Vm       6-Lower Bottom Cap:*)
s2Min[8, 6, qq_, pp_] := pp + 1; s2Max[8, 6, qq_, pp_] := pp + qq;
mMin[8, 6, qq_, pp_, s2_] := 1;
mMax[8, 6, qq_, pp_, s2_] := pp + qq - s2 + 1;
irow[8, 6, qq_, pp_, s2_,
   m_] := -(1/2) (pp + pp^2 + qq + qq^2 + s2^2) +
   s2 (pp + qq + 3/2) + m ;
jcol[8, 6, qq_, pp_, s2_, m_] :=
```

```
  irow[8, 6, qq, pp, s2, m] - 1 - pp - qq + s2;
f[8, 6, qq_, pp_, s2_,
   m_, \[Alpha]_, \[Beta]_] := (-(m (1 + qq - m) (2 + pp + qq -
         m) (s2/2 - \[Alpha])/(s2 (1 + s2)) )^(1/2)
      KroneckerDelta[\[Beta] - (\[Alpha] + 1/2)]);

(*Part II  A Numerical Example*)
(*Notes:
1. x=1,...,8 for TUV matrices in the order T3,U3,Tp,Tm,Up,Um,Vp,Vm
2. For T3,U3,Tp,Tm: n=1,2,3 indicates Top Cap,Middle,Bottom Cap
3. For Up,Um,Vp,Vm: n=1,2,3,4,5,6 indicates Upper Top Cap,Upper \
Middle,Upper Bottom Cap, Lower Top Cap,Lower Middle,Lower Bottom Cap*)


(* Choose (p,q): *)
p = 5; q = 3;

If[p >= q, pqOK = True,
  pqOK = False];(*For p<q, switch p and q. We need p>=q.*)
If[pqOK, "sweet", p0 = p]; If[pqOK, "sweet", q0 = q];
If[pqOK, "sweet", p = q0]; If[pqOK, "sweet", q = p0];

Tspins =(*The list of SU(2) spins for the matrices T3,Tp,
  Tm.*)(1/2) Flatten[{Table[s2, {s2, 0, q}, {m, s2 + 1}],
     Table[s2, {s2, q + 1, p}, {m, q + 1}],
     Table[s2, {s2, p + 1, p + q}, {m, p + q + 1 - s2}]}];

numTspins = (p + 1) (q + 1);(*the number of Tspins*)
dimREP = (1/2) (p + 1) (q + 1) (p + q +
     2);(*dimension of the TUV matrices*)


(* (r,c) - (row,column) indices, (\[Alpha],\[Beta]) - spin component \
indices*)
TUVmatrices = {};
For[x = 1, x <= 4, x++,
  matrixX = Table[0, {i, dimREP}, {j, dimREP}];
  For[n = 1, n <= 3 , n++,(*matrices T3,U3,Tp,Tm have nMAX = 3*)
   For[s2 = s2Min[x, n, q, p], s2 <= s2Max[x, n, q, p], s2++,
    For[m = mMin[x, n, q, p, s2], m <= mMax[x, n, q, p, s2], m++,
```

```
    Table[matrixX[[(*r*)(s2/2 - \[Alpha] + 1) +
        Sum[2 Tspins[[ii]] + 1, {ii,
          irow[x, n, q, p, s2, m] - 1}],(*c*)(s2/2 - \[Beta] + 1) +
        Sum[2 Tspins[[jj]] + 1, {jj,
          jcol[x, n, q, p, s2, m] - 1}]]] =
      f[x, n, q, p, s2, m, \[Alpha], \[Beta]], {\[Alpha],
      s2/2, -s2/2, -1}, {\[Beta], s2/2, -s2/2, -1}]]];
  AppendTo[TUVmatrices, matrixX]
  ];

For[x = 5, x <= 8, x++,
  matrixX = Table[0, {i, dimREP}, {j, dimREP}];
  For[n = 1, n <= 3, n++,
   For[s2 = s2Min[x, n, q, p], s2 <= s2Max[x, n, q, p], s2++,
    For[m = mMin[x, n, q, p, s2], m <= mMax[x, n, q, p, s2], m++,
     Table[matrixX[[(*r*)(s2/2 - \[Alpha] + 1) +
        Sum[2 Tspins[[ii]] + 1, {ii,
          irow[x, n, q, p, s2, m] - 1}],(*c*)((s2 + 1)/2 - \[Beta] +
          1) + Sum[
         2 Tspins[[jj]] + 1, {jj, jcol[x, n, q, p, s2, m] - 1}]]] =
      f[x, n, q, p, s2, m, \[Alpha], \[Beta]], {\[Alpha],
      s2/2, -s2/2, -1}, {\[Beta], (s2 + 1)/2, -(s2 + 1)/2, -1}]]]];
  For[n = 4, n <= 6, n++,(* matrices Up,Um,Vp,Vm have nMAX = 6.*)
   For[s2 = s2Min[x, n, q, p], s2 <= s2Max[x, n, q, p], s2++,
    For[m = mMin[x, n, q, p, s2], m <= mMax[x, n, q, p, s2], m++,
     Table[matrixX[[(*r*)(s2/2 - \[Alpha] + 1) +
        Sum[2 Tspins[[ii]] + 1, {ii,
          irow[x, n, q, p, s2, m] - 1}],(*c*)((s2 - 1)/2 - \[Beta] +
          1) + Sum[
         2 Tspins[[jj]] + 1, {jj, jcol[x, n, q, p, s2, m] - 1}]]] =
      f[x, n, q, p, s2, m, \[Alpha], \[Beta]], {\[Alpha],
      s2/2, -s2/2, -1}, {\[Beta], (s2 - 1)/2, -(s2 - 1)/2, -1}]]]];
  AppendTo[TUVmatrices, matrixX]
  ];

(*Identify the matrices*)
T3 = TUVmatrices[[1]]; U3 = TUVmatrices[[2]]; Tp = TUVmatrices[[3]];
Tm = TUVmatrices[[4]]; Up = TUVmatrices[[5]]; Um = TUVmatrices[[6]];
Vp = TUVmatrices[[7]]; Vm = TUVmatrices[[8]];
```

```
(*Special steps when q>p :*)
If[pqOK, "sweet", Tp = -Transpose[Tp]]; If[pqOK, "sweet",
 Tm = -Transpose[Tm]];
If[pqOK, "sweet", T3 = -Transpose[T3]];
If[pqOK, "sweet", Up = -Transpose[Up]];
If[pqOK, "sweet", Um = -Transpose[Um]];
If[pqOK, "sweet", U3 = -Transpose[U3]];
If[pqOK, "sweet", Vp = -Transpose[Vp]];
If[pqOK, "sweet", Vm = -Transpose[Vm]];
If[pqOK, "sweet", p = p0]; If[pqOK, "sweet", q = q0];

Print["Check  equations for {p,q} = ", {p, q},
  " . All eqns are satisfied: ", {0} ==
   Union[
    Flatten[Simplify[{T3 . Tp - Tp . T3 - Tp,
       T3 . Tm - Tm . T3 - (-Tm), Tp . U3 - U3 . Tp - (+(1/2) Tp),
       T3 . U3 - U3 . T3, Tm . U3 - U3 . Tm - (-(1/2) Tm),
       T3 . Up - Up . T3 - (-(1/2) Up),
       T3 . Um - Um . T3 - (+(1/2) Um), T3 . Vp - Vp . T3 - (1/2) Vp,
       T3 . Vm - Vm . T3 - (-(1/2) Vm), Tp . Tm - Tm . Tp - 2 T3,
       Tp . Up - Up . Tp - Vp, Tp . Um - Um . Tp, Tp . Vp - Vp . Tp,
       Tp . Vm - Vm . Tp - (-Um), Tm . Up - Up . Tm,
       Tm . Um - Um . Tm - (-Vm), Tm . Vp - Vp . Tm - Up,
       Tm . Vm - Vm . Tm, U3 . Up - Up . U3 - (+ Up),
       U3 . Um - Um . U3 - (- Um), U3 . Vp - Vp . U3 - (+(1/2) Vp),
       U3 . Vm - Vm . U3 - (-(1/2) Vm), Up . Um - Um . Up - (2 U3),
       Up . Vp - Vp . Up, Up . Vm - Vm . Up - (+ Tm),
       Um . Vp - Vp . Um - (- Tp), Um . Vm - Vm . Um,
       Vp . Vm - Vm . Vp - (2 U3 + 2 T3), (1/2) (Tp . Tm + Tm . Tp) +
        T3 . T3 + (1/2) (Vp . Vm + Vm . Vp) + (1/2) (Up . Um +
           Um . Up) + (1/
           3) (2 U3 + T3) . (2 U3 +
            T3) - (((p^2 + p*q + q^2 + 3 p + 3 q)/3) IdentityMatrix[
           dimREP])}]]]];

(*The F^i matrix generators *)
Fi = {(1/2) (Tp + Tm), (-I/2) (Tp - Tm),
   T3, (1/2) (Vp + Vm), (-I/2) (Vp - Vm), (1/2) (Up + Um), (-I/
      2) (Up + (-Um)), (2 U3 + T3)/Sqrt[3]};
```

```
Print["For {p,q} = ", {p, q},
  ", the F^i are traceless: ", {0} ==
   Union[Table[Sum[Fi[[i]][[j, j]], {j, dimREP}], {i, 8}]] ];
Print["and hermitian: ", {0} ==
   Union[Flatten[
     Table[Union[
       Flatten[Table[
         Fi[[i]][[j, k]] - Conjugate[Fi[[i]][[k, j]]], {j,
          dimREP}, {k, dimREP}]]], {i, 8}]]]];

Print["For {p,q} = ", {p, q},
  ", check that the quadratic invariant: F^2 = Sum F^i.F^i = \
(p^2+p*q+q^2+3p+3q)/3, is satisfied: ", {0} ==
   Union[
    Flatten[Simplify[
       Sum[Fi[[i]] . Fi[[i]], {i,
         8}]] - (((p^2 + p*q + q^2 + 3 p + 3 q)/3) IdentityMatrix[
         dimREP])]]];

(*References*)
(**)
(*1. Wolfram Research,Inc.,Mathematica,Version 14.0,Champaign,IL \
(2023)*)
(*2. The present version runs with Mathematica 14.0.0 on a Microsoft \
Windows (64-bit) platform.*)
(*3. R. Shurtleff, arXiv:0908.3864v3[math-ph], 2023. *)
(*4. S.Gasiorowicz,Elementary Particle Physics,(John \
Wiley\&Sons,Inc.,New York,1966),Equations (17.21-24)*)
(*5. W. Greiner and B. M\"{u}ller, Quantum Mechanics,Symmetries,2nd \
revised edition (Springer-Verlag,Berlin,1994).*)
(*Links to this notebook in a ready-to-run file:
https://www.wolframcloud.com/obj/shurtleffr/Published/\
SU3MatricesMMA2024.nb ,
https://www.dropbox.com/scl/fi/ey0wk7cvhupeqeeeowr1v/MMAforPAPER3.nb?\
rlkey=7jhigi9m8lnxo862c78jdznw6&st=hkuoc12g&dl=0*)
```

End of Program

7 Fortran 90 Program

The Fortran 90 program calculates the eight basis matrices $T^{\pm}$, T^3, $U^{\pm}$, U^3, and $V^{\pm}$ for the (p,q) irrep of the $\mathfrak{su}(3)$ algebra. The program checks that the eight matrices obey the $\mathfrak{su}(3)$ algebra and the quadratic Casimir invariance equation.

Those familiar with the Fortran language can alter the code to suit their own purposes. Here, for my convenience with its testing, the program calculates the TUV matrices for all (p, q) irreps with a constant sum $p + q = pPLUSq$. The user must select the value of $pPLUSq$ by finding and changing its value in the code. The program has been successfully tested for $pPLUSq = 1, 2, ..., 15$.

Instead of a 'Readme.txt' file, the program includes a Readme section, Sec. 1 Readme. In short, there are two types of output. A summary of the results is output to the standard console screen. And the eight TUV matrices are saved in a file, one file for each of the (p, q) irreps.

In order to be displayed here, the Fortran 90 program is copied verbatim into Latex. Exporting the pdf Fortran code to plain text may require some manipulation to be run successfully. Such issues may be avoided by following the link to the ready-to-run file that is provided in Ref. [15]. Please download the file as soon as possible to avoid the inevitable extinction of the link.

The Fortran 90 Program

```
!RULER78911234567892123456789312345678941234567895123456789612345678971234567898 1
!
!A FORTRAN 90 program for calculating generators for irreps of the SU(3)
! Lie algebra, by Richard Shurtleff, Wentworth Institute of Technology, Boston,
! MA, USA - retired
! email: shurtleffr(at)wit.edu, momentummatrix(at)yahoo.com
!
!------- CONTENTS -----------------------
!---0. Preamble
!---1. Readme
!---2. Program Start, Interface Blocks --------------------
!---3. Type declarations -----------------------
!---4. Begin -----------------------
!---5. Preliminaries -----------------------
!---6. Make the list of Tspins -----------------------
!---7. Make the eight TUV matrices -----------------------
!---8. Extra steps are needed when p0 < q0 ----------------
!---9. Check for components that are 'Not a Number', 'NaN'-------------
!--10. Check 28 commutator relations and an invariance equation -----------
!--11. Save the results to a file, end program  -----------------------
!--12. External functions, end-of-file-----------------------------
```

```
!
!----------------------------0. Preamble-----------------------------------
!This FORTRAN program implements the calculations in the Main article [1],
! 'Formulas for SU(3) Matrix Generators' by Richard Shurtleff,
!   Main article's Abstract:
! The Lie algebra of a Lie group is a set of commutation relations, equations
! satisfied by the group's generators. For SU(2) and many other Lie groups,
! the equations have been solved and matrix generators are realized as
! algebraic expressions suitable for further investigation or numerical
! evaluation. This article presents formulas that give a set of matrix
! generators for any irreducible representation of the group SU(3), the group
! of unimodular unitary three-dimensional complex matrices with matrix
! multiplication. A computer program to calculate the matrix generators is
! included.
!
!   Acknowledgment
! I would like to express my appreciation for Patrick Koh's debugging of a
! previous program which taught me lessons that I have applied in writing
! this program.
!
!   Copyright: CC BY-SA. Public use and modification of this code are allowed
!   provided that the preprint[1] or any subsequent published version is cited.
!
!   References
!
!   [1] R. Shurtleff, "Formulas for SU(3) Matrix Generators," viXra,
!        2409.0060v1, 2024.
!   [2] This program can be downloaded by following the link:
! https://www.dropbox.com/scl/fi/zia4anqs72e2bqsja5f6t/SU3GENpPLUSq.DP3.f90?
! rlkey=f9mhjm18mnguc8yapow7l3z3s&st=2jby4oyx&dl=0
!
!--------------------------1. Readme-------------------------------------

! This program calculates 8 basis generator matrices for the (p,q) irrep of
! the su(3) Lie agebra.  The equations are implicit in the code in Sec. 10.
! See the article Ref. [1] for details.
!
! This FORTRAN 90 program ran successfully on a Windows 11 computer with a
!  GNU fortran compiler Code::Blocks 20.03, Created: 2010/25/05 11:52
!  Updated: 2010/25/05 11:52, Author: HighTec EDV-Systeme GmbH,
```

```
!  Copyright 2010 HighTec EDV-Systeme GmbH, available at
!  https://gcc.gnu.org/fortran/ and http://www.codeblocks.org
!
! USER GUIDE:
! INPUT your value of p and q in Sec. 4 Begin as p0 and q0.
! Then, if you wish, reset the tolerance for error, MaxErrLimit.
!For double precision, I set MaxErrLimit to 1.D-10. Check the code, to be sure.
!
! OUTPUT to standard screen
! Some data is sent to the standard output. This includes the integers
! p0 and q0 that identify the irrep, the dimension of the irrep's matrices,
! the error tolerance, the maximum error found in 29 equations that the
! generators are required to solve.
!
! OUTPUT to a DATA File named "p#q#SU3gen.dat," where # stands for the
! values of p,q. The DATA File's records are
! Record 1: the integers p, q, and the max error in 29 su(3) algebra
! equations, the 28 commutation relations plus the invariance relation.
! The Format of Record 1: FORMAT(2I3,1E16.7).
!
! Records 2,3,...: the components of the eight BASIS matrices, a total
!  of 8n**2 real numbers, where n is the matrix dimension dimREP, n = dimREP.
!  Each of the 8 TUV matrices X has n**2 components arranged by rows, i.e.
!   X(1,1), X(1,2),X(1,3),...,X(1,n),X(2,1),X(2,2),X(2,3),...,X(n,n).
!  The sequence of TUV matrices is X = T3, U3, Tp, Tm, Up, Um, Vp, Vm.
! The Format of Record 2,3,...: FORMAT(5F16.12),
!Thus the matrix components appear 5 real numbers per line until the last line.
!
! Sample records from an OUTPUT file for the (p,q) = (3,1) irrep:
! Record #1:
! 3  1   0.3552714E-14
! Record #12:
!-0.500000000000 0.000000000000  0.000000000000  0.000000000000  0.000000000000
! Record #541:
!0.000000000000  1.118033988750  0.000000000000  0.000000000000  0.000000000000
! Record #923
! 0.000000000000  0.000000000000  0.000000000000
!
!  Given n = dimREP = 24 for (p,q) = (3,1), the four Records shown tell us that
!   Record 1: p = 3, q = 1, MaxErr = 0.3552714E-14
```

```
!   Record 12: 5*(11-1)+1 = 51 = 2*n + 3 implies T3(3,3) = -0.5
!   Record 541: 5*(540-1)+2 = 4*24**2 + 16*24 + 9 implies Up(17,9) = 1.1180
!   Record 923: 5*(922-1) + 3 = 4608 = 8*24**2 = 8*n**2, Total # of components
!
! Best wishes, may you find the program of use.
!
PROGRAM SU3Generators
!
    IMPLICIT NONE
!------------------------2. Program Start, Interface Blocks -------------------

INTERFACE
    FUNCTION iTC0(s2,m) RESULT  (w)    !
    IMPLICIT NONE
        INTEGER :: w     ! iTC default Top Cap row function
        INTEGER, INTENT(IN) :: s2,m    ! q characterizes the (p,q)-irrep
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION iM0(q,s2,m) RESULT  (w)    !
    IMPLICIT NONE
        INTEGER :: w     ! iM0 starter Middle block row function
        INTEGER, INTENT(IN) :: q,s2,m    !
    END FUNCTION
END INTERFACE

INTERFACE
    FUNCTION iBC0(q,p,s2,m) RESULT  (w)    !
    IMPLICIT NONE
        INTEGER :: w     ! iBC0 starter Bottom Cap block row function
        INTEGER, INTENT(IN) :: q,p,s2,m    !
    END FUNCTION
END INTERFACE
INTERFACE   !The nth block starts at the end of the first n-1 Tspin blocks
    FUNCTION sumTmatrixDims(n,numTspins,Tspins) RESULT  (w)
        INTEGER, INTENT(IN) :: n,numTspins   ! nth spin, number of Tspins
        REAL(8), INTENT(IN) :: Tspins(numTspins)   ! the list of Tspins
        INTEGER :: i,k,w   ! dummy indices, the result is an integer
        REAL(8) :: TspinN    ! dummy spin
```

```
    END FUNCTION
END INTERFACE

INTERFACE               ! f is the formula for a component
FUNCTION f(q,p,s2,m,alpha,beta) RESULT  (w)
    IMPLICIT NONE
    REAL(8) :: w(8,6)
    INTEGER :: q,p,s2,m !
    REAL(8), INTENT(IN) :: alpha,beta
END FUNCTION
END INTERFACE
!
!-----------------------3. Type declarations ----------------------
!
INTEGER ::  p0,q0,p,q,pPLUSq          ! integers p0,q0 identify the su(3) irrep
INTEGER ::  numTspins,dimREP    ! the number of T-spins, TUV matrix dimension
INTEGER  :: x,i,j,k,s2,m,n,s2Min(8,6),s2Max(8,6),mMax(8,6),iTC1,iM1,iBC1
INTEGER  :: iRow(8,6),jCol(8,6),r,c,rTOTAL,cTOTAL,alpha2,beta2,dbeta2(8),xtra
INTEGER  :: s2Min0(4,3)
REAL(8), ALLOCATABLE :: Tspins(:),unitMatrix(:,:)
REAL(8) ::  si,alpha,beta,fHere(8,6),s
! The 8 matrices produced by the program:
REAL(8),ALLOCATABLE :: T3(:,:),U3(:,:),Tp(:,:),Tm(:,:),Up(:,:),Um(:,:)
REAL(8),ALLOCATABLE :: Vp(:,:),Vm(:,:),matrixX1(:,:,:)
REAL(8) :: MaxErr,MaxErrLimit !Max error found in required equations, tolerance
!
! X1X2comm - commutation relation: X1X2comm = ([X1,X2] - expected value)
REAL(8),ALLOCATABLE:: T3Tpcomm(:,:),T3Tmcomm(:,:),T3Upcomm(:,:),T3Umcomm(:,:)
REAL(8),ALLOCATABLE:: T3U3comm(:,:),T3Vpcomm(:,:),T3Vmcomm(:,:),TpTmcomm(:,:)
REAL(8),ALLOCATABLE:: TpUpcomm(:,:),TpUmcomm(:,:),TpU3comm(:,:),TpVpcomm(:,:)
REAL(8),ALLOCATABLE:: TpVmcomm(:,:),TmUpcomm(:,:),TmUmcomm(:,:),TmU3comm(:,:)
REAL(8),ALLOCATABLE:: TmVpcomm(:,:),TmVmcomm(:,:),U3Upcomm(:,:),U3Umcomm(:,:)
REAL(8),ALLOCATABLE:: U3Vpcomm(:,:),U3Vmcomm(:,:),UpUmcomm(:,:),UpVpcomm(:,:)
REAL(8),ALLOCATABLE:: UpVmcomm(:,:),UmVpcomm(:,:),UmVmcomm(:,:),VpVmcomm(:,:)
REAL(8),ALLOCATABLE:: SU3identity(:,:)
LOGICAL:: p0GEq0,noNaNs,MaxErrNotTooBig  ! p0 >= q0?, matrices all numerical?
CHARACTER (len=20) :: file_name         ! file name for output data file
!
!-----------------------4. Begin ----------------------
!
```

```
WRITE(*,*)    'The program calculates the TUV matrices, a basis for the &
            generators of SU(3) for the (p,q) irrep'
WRITE(*,*) 'If successful, the program produces an output file with the &
            values of p and q in the file-name.'
   MaxErrLimit = 1.D-10 ! Set the tolerance, the largest allowed error in
   !                       any component of the 29 eqns
! The irreps (p,q) with p + q = pPLUSq are calculated.
    pPLUSq = 4 !
 DO   p0 = 0,pPLUSq     ! The irrep identifiers (p,q) are called
      q0 = pPLUSq - p0    !  (p0,q0), initially.
!
    dimREP = (p0+1)*(q0+1)*(p0+q0+2)/2 !Matrices are nxn square, n = dimREP

!
!------------------------5. Preliminaries -----------------------
!
  IF (p0 .GE. q0) THEN  ! The formulas are set up for p >= q.
    p0GEq0 = .TRUE.
    p = p0
    q = q0
    ELSE IF (p0 < q0) THEN  !When p < q, we swap p and q for the calculation
        p0GEq0 = .FALSE.    ! and take the negative transpose.
        p = q0
        q = p0
  END IF
 ALLOCATE( unitMatrix(dimREP,dimREP)) ! unit matrix, a.k.a. identity matrix
    DO  i = 1,dimREP
        DO  j = 1,dimREP
    unitMatrix(i,j) = 0.0_8  ! the unit matrix has zeros everywhere,except
        END DO
            unitMatrix(i,i) = 1.0_8    ! ones along the diagonal.
    END DO
!
! (*Part 1.  Formulas for TUV matrices. T3, U3, Tp, Tm, Up, Um, Vp, Vm*)
!
!-----------------6. Make the list of Tspins -----------------------
! The diagonal blocks of the T-matrices are SU(2) irreps. Tspins is the list &
! of those SU(2) spins.
    numTspins = (p+1)*(q+1)         ! numTspins - the number of Tspins
ALLOCATE( Tspins(numTspins))
```

```
n=1                                   ! Make the list of Tspins
    DO s2=1,q         !
      DO m = 1, s2
         Tspins(n) = dble(s2-1)*0.5_8  !
         n=n+1
      END DO
    END DO
    DO s2=q,p
      DO m = 1, q+1
         Tspins(n) = dble(s2)*0.5_8
         n=n+1
      END DO
    END DO
    DO s2=p+1,p+q
      DO m = 1,q+1-s2+p
         Tspins(n) = dble(s2)*0.5_8
         n=n+1
      END DO
    END DO
!   WRITE(*,*) 'The Tspins si are si = ', Tspins
!  WRITE(*,*) 'The Tspins are often doubled, with s2 = 2*si = ', NINT(2*Tspins)
!   WRITE(*,*) 'Check that the number of Tspins agrees with the formula:&
!               numTspins, SIZE(Tspins) ', numTspins,'?=?', SIZE(Tspins)
!
!-----------------7. Make the eight TUV matrices -----------------------
!
!                                Dummy index scheme
!     x - TUV matrix index, 1(T3),2(U3),3(Tp),4(Tm),5(Up),6(Um),7(Vp),8(Vm)
!     n - 1.Top Cap, 2.Middle, 3.Bottom Cap for x = 1,2,3,4
!     n - 1.Upper Top Cap, 2.Upper Middle, 3.Upper Bottom Cap for x = 5,6,7,8
!     n - 4.Lower Top Cap, 5.Lower Middle, 6.Lower Bottom Cap for x = 5,6,7,8
!
!               s2 MIN
! s2Min = min value of 2*s for each TUV matrix x and each Tspin range n.
      s2Min = TRANSPOSE(RESHAPE((/0*1,q+1,p+1,0,0,0,0*2,q+1,p+1,0,0,0,&
      0*3,q+1,p+1,0,0,0,0*4,q+1,p+1,0,0,0,0*5,q-1,p+1,1,q+1,p+1,&
      0*6,q,p,1,q+1,p+1,0*7,q-1,p+1,1,q+1,p+1,0*8,q,p,1,q+1,p+1/),&
      SHAPE(TRANSPOSE(s2Min))))
!
!
```

```
!               s2 MAX
! s2Max = max value of 2*s for each TUV matrix x and each Tspin range n.
      s2Max = TRANSPOSE(RESHAPE((/0*1+q,p,p+q,0,0,0,0*2+q,p,p+q,0,0,0,&
      0*3+q,p,p+q,0,0,0,0*4+q,p,p+q,0,0,0,0*5+q-2,p,p+q-1,q,p,p+q,&
      0*6+q-1,p-1,p+q-1,q,p,p+q,0*7+q-2,p,p+q-1,q,p,p+q,&
      0*8+q-1,p-1,p+q-1,q,p,p+q/),SHAPE(TRANSPOSE(s2Max))))
!
      dbeta2=(/0,0,-2,+2,+1,-1,-1,+1/) ! 2*beta = 2*alpha + dbeta2
 ! The difference dbeta2 depends only on which TUV matrix x it is for, &
!   so there are 8 values of dbeta2, one for each x.
!

ALLOCATE( matrixX1(8,dimREP,dimREP))
    DO x = 1,8        ! Initially, all TUV matrix components vanish
                DO  i = 1,dimREP
                    DO  j = 1,dimREP
                                matrixX1(x,i,j) = 0._8
                    END DO
                END DO
    END DO
!               Make T3, U3, Tp, Tm
  DO x = 1,4        ! x = 1(T3),2(U3),3(Tp),4(Tm)
        xtra=x  ! for unknown reasons, sometimes xtra works when x doesn't work
    DO n = 1,3  ! spin parameter sections n = 1(TopCap),2(Middle),3(Bottom Cap)
          DO s2 = s2Min(x,n),s2Max(x,n)     ! s2 = twice the spin, s2 = 2*s
                                si = dble(s2)/2._8   ! spin si = half of s2
!
!               m MAX depends on s2, so put it in the s2 DO loop
      mMax = TRANSPOSE(RESHAPE((/0*1+s2+1,q+1,p+q-s2+1,0,0,0,&
      0*2+s2+1,q+1,p+q-s2+1,0,0,0,0*3+s2+1,q+1,p+q-s2+1,0,0,0,&
      0*4+s2+1,q+1,p+q-s2+1,0,0,0,0*5+s2+1,q,p+q-s2,s2,q+1,p+q-s2+1,&
      0*6+s2+1,q+1,p+q-s2,s2,q,p+q-s2+1,0*7+s2+1,q,p+q-s2,s2,q+1,p+q-s2+1,&
      0*8+s2+1,q+1,p+q-s2,s2,q,p+q-s2+1/),SHAPE(TRANSPOSE(mMax))))
!
            Do m = 1,mMax(x,n)
 !      Define convenient, repeatedly used row constants
                iTC1=iTC0(s2,m)     ! Top Cap
                iM1=iM0(q,s2,m)     ! Middle
                iBC1=iBC0(q,p,s2,m) ! Bottom Cap
 ! There is a value of the block row index i for each (x,n):
```

```
     iRow = TRANSPOSE(RESHAPE((/0*1+iTC1,iM1,iBC1,0,0,0,& ! (i,j) Block
      0*2+iTC1,iM1,iBC1,0,0,0,  0*3+iTC1,iM1,iBC1,0,0,0,&
      0*4+iTC1,iM1,iBC1,0,0,0,  0*5+iTC1,iM1+1,iBC1+1,iTC1+1,iM1,iBC1,&
      0*6+iTC1,iM1,iBC1,iTC1,iM1,iBC1,&
      0*7+iTC1,iM1+1,iBC1+1,iTC1+1,iM1,iBC1,&
      0*8+iTC1,iM1,iBC1,iTC1,iM1,iBC1/),SHAPE(TRANSPOSE(iRow))))
!   Same for j, iRow,jCol are 8x6 matrices, one entry for each (x,n)
     jCol = TRANSPOSE(RESHAPE((/0*1+iTC1,iM1,iBC1,0,0,0,& ! (i,j) Block
      0*2+iTC1,iM1,iBC1,0,0,0,  0*3+iTC1,iM1,iBC1,0,0,0,&
      0*4+iTC1,iM1,iBC1,0,0,0,&
      0*5+iTC1+s2+1,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q-1,iBC1-p-q+s2-2,&
      0*6+iTC1+s2+2,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q,iBC1-p-q+s2-1,&
      0*7+iTC1+s2+1,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q-1,iBC1-p-q+s2-2,&
      0*8+iTC1+s2+2,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q,iBC1-p-q+s2-1/),&
      SHAPE(TRANSPOSE(jCol))))
 ! WRITE(*,*)    '(x,n,s2,m,(iRow(x,n),jCol(x,n)),si) = ',&
 !         (/x,n,s2,m,iRow(x,n),jCol(x,n)/),' si = ',si
             DO alpha2 = s2,-s2,-2       ! twice the spin component alpha
            alpha=dble(alpha2)/2._8     ! alpha is an integer or half-integer
             beta2=alpha2+dbeta2(x)  ! one beta per alpha,
             beta=dble(beta2)/2._8   !
        r=NINT(si-alpha)+1      ! row in block, MMA: (s2/2 - \[Alpha] + 1)
        c=NINT(si-beta)+1       ! col in block, MMA: (s2/2 - \[Beta] + 1)
    rTOTAL=r+sumTmatrixDims(iRow(x,n),numTspins,Tspins)  !row in the matrix
    cTOTAL=c+sumTmatrixDims(jCol(x,n),numTspins,Tspins)  !col in the matrix
        IF ((r==0).OR.(c==0)) THEN    ! r=0 or c=0 means the row or column is
            GOTO 100              ! in the wrong block, i.e. the previous block
        END IF
    IF ((rTOTAL>dimREP).OR.(cTOTAL>dimREP)) THEN    ! r>max or c>max means the
            GOTO 100                       ! row or column is not in the matrix
    END IF                                  ! If true, then skip this alpha2.
            fHere=f(q,p,s2,m,alpha,beta)  ! the array f(x,n) of the block
!  component with row/col indices alpha and beta, parameterized by s2,m
! For matrix x, output the row/col indices and the component's value f
 ! WRITE(*,*)'matrix x, row,col,f =',x,rTOTAL,cTOTAL,fHere(x,n)
! output the DO loop values x,n,s2,m,alpha2 plus beta2,r,c,f:
!WRITE(*,*)'x,n,s2,m,alpha2,beta2,r,c,f =',x,n,s2,m,alpha2,beta2,r,c,fHere(x,n)
!     The component of matrix x at row/col rTOTAL,cTOTAL:
                  matrixX1(x,rTOTAL,cTOTAL)= fHere(x,n) !X(rTOTAL,cTOTAL) = f
    100 CONTINUE   !Bad r,c. We skip alpha2 when r,c=0 or rTOTAL,cTOTAL>dimREP.
```

```
                END DO  ! alpha2
              End Do  ! m
            End Do    ! s2
          End Do  ! n
  End Do    ! x
!
! Make Up, Um, Vp, Vm, Upper and Lower blocks
  DO x = 5,8        ! x = 5(Up), 6(Um), 7(Vp), 8(Vm)
        xtra=x  !for unknown reasons, sometimes xtra works when x doesn't work
 ! n = 1,2,3     'Upper' blocks are above the diagonal for matrices x = 5,6,7,8
    DO n = 1,3  ! spin parameter sections n = 1(TopCap),2(Middle),3(Bottom Cap)
          DO s2 = s2Min(x,n),s2Max(x,n)     ! s2 = twice the spin, s2 = 2s
   !         Write(*,*) 'Vp(8,1) = ',matrixX1(7,8,1)
            si = dble(s2)/2._8   ! spin si = half of s2
!
!               m MAX depends on s2, so put it in the s2 DO loop
      mMax = TRANSPOSE(RESHAPE((/0*1+s2+1,q+1,p+q-s2+1,0,0,0,&
      0*2+s2+1,q+1,p+q-s2+1,0,0,0,0*3+s2+1,q+1,p+q-s2+1,0,0,0,&
      0*4+s2+1,q+1,p+q-s2+1,0,0,0,0*5+s2+1,q,p+q-s2,s2,q+1,p+q-s2+1,&
      0*6+s2+1,q+1,p+q-s2,s2,q,p+q-s2+1,0*7+s2+1,q,p+q-s2,s2,q+1,p+q-s2+1,&
      0*8+s2+1,q+1,p+q-s2,s2,q,p+q-s2+1/),SHAPE(TRANSPOSE(mMax))))
!
            Do m = 1,mMax(x,n)
 !      Define convenient, repeatedly used row constants
                iTC1=iTC0(s2,m)     ! Top Cap
                iM1=iM0(q,s2,m)     ! Middle
                iBC1=iBC0(q,p,s2,m) ! Bottom Cap
     iRow = TRANSPOSE(RESHAPE((/0*1+iTC1,iM1,iBC1,0,0,0,&
      0*2+iTC1,iM1,iBC1,0,0,0,  0*3+iTC1,iM1,iBC1,0,0,0,&
      0*4+iTC1,iM1,iBC1,0,0,0,  0*5+iTC1,iM1+1,iBC1+1,iTC1+1,iM1,iBC1,&
      0*6+iTC1,iM1,iBC1,iTC1,iM1,iBC1,&
      0*7+iTC1,iM1+1,iBC1+1,iTC1+1,iM1,iBC1,&
      0*8+iTC1,iM1,iBC1,iTC1,iM1,iBC1/),SHAPE(TRANSPOSE(mMax))))
!
     jCol = TRANSPOSE(RESHAPE((/0*1+iTC1,iM1,iBC1,0,0,0,&
      0*2+iTC1,iM1,iBC1,0,0,0,  0*3+iTC1,iM1,iBC1,0,0,0,&
      0*4+iTC1,iM1,iBC1,0,0,0,&
      0*5+iTC1+s2+1,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q-1,iBC1-p-q+s2-2,&
      0*6+iTC1+s2+2,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q,iBC1-p-q+s2-1,&
      0*7+iTC1+s2+1,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q-1,iBC1-p-q+s2-2,&
```

```
      0*8+iTC1+s2+2,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q,iBC1-p-q+s2-1/),&
      SHAPE(TRANSPOSE(mMax))))
   !             WRITE(*,*)    '(x,n,(s2,m),(iRow(x,n),jCol(x,n)),si) = ',&
    !                        (/x,n,s2,m,iRow(x,n),jCol(x,n)/),' si = ',si
                DO alpha2 = s2,-s2,-2       !
     !               Write(*,*) 'Vp(8,1) = ',matrixX1(7,8,1)
                    alpha=dble(alpha2)/2._8
                      beta2=alpha2+dbeta2(x)
                      beta=dble(beta2)/2._8
        r=NINT(si-alpha)+1      ! row in block, MMA: (s2/2 - \[Alpha] + 1)
        c=NINT(si+0.5_8-beta)+1 ! col in block, MMA: ((s2+1)/2 - \[Beta] + 1)
    rTOTAL=r+sumTmatrixDims(iRow(x,n),numTspins,Tspins)  !row in the matrix
    cTOTAL=c+sumTmatrixDims(jCol(x,n),numTspins,Tspins)  !col in the matrix
        IF ((r==0).OR.(c==0)) THEN    ! r=0 or c=0 means the row or column is
            GOTO 200              ! in the wrong block, i.e. the previous block
        END IF                        ! If true, skip this alpha2
    IF ((rTOTAL>dimREP).OR.(cTOTAL>dimREP)) THEN    ! r>Max or c>Max means the
            GOTO 200                    ! row or column is not in the matrix
    END IF                              ! If true, then skip this alpha2.
            fHere=f(q,p,s2,m,alpha,beta)  ! the array f(x,n) of the block
 !  component with row/col indices alpha and beta, parameterized by s2,m
! For matrix x, output the row/col indices and the component's value f
 ! WRITE(*,*)'matrix x, row,col,f =',x,rTOTAL,cTOTAL,fHere(x,n)
! output the DO loop values x,n,s2,m,alpha2 plus beta2,r,c,f:
!WRITE(*,*)'x,n,s2,m,alpha2,beta2,r,c,f =',x,n,s2,m,alpha2,beta2,r,c,fHere(x,n)
!     The component of matrix x at row/col rTOTAL,cTOTAL:
                  matrixX1(x,rTOTAL,cTOTAL)= fHere(x,n)
    200 CONTINUE   !Bad r,c. We skip alpha2 when r,c=0 or rTOTAL,cTOTAL>dimREP.
                END DO  ! alpha2
              End Do  ! m
            End Do    ! s2
        End Do  ! n
!      n = 4,5,6     'Lower' blocks are below the diagonal for x = 5,6,7,8
    DO n = 4,6  ! spin parameter sections n = 4(TopCap),5(Middle),6(Bottom Cap)
          DO s2 = s2Min(x,n),s2Max(x,n)     ! s2 = twice the spin, s2 = 2*s
            si = dble(s2)/2._8   ! spin si = half of s2
!
!               m MAX depends on s2, so put it in the s2 DO loop
      mMax = TRANSPOSE(RESHAPE((/0*1+s2+1,q+1,p+q-s2+1,0,0,0,&
      0*2+s2+1,q+1,p+q-s2+1,0,0,0,0*3+s2+1,q+1,p+q-s2+1,0,0,0,&
```

```
      0*4+s2+1,q+1,p+q-s2+1,0,0,0,0*5+s2+1,q,p+q-s2,s2,q+1,p+q-s2+1,&
      0*6+s2+1,q+1,p+q-s2,s2,q,p+q-s2+1,0*7+s2+1,q,p+q-s2,s2,q+1,p+q-s2+1,&
      0*8+s2+1,q+1,p+q-s2,s2,q,p+q-s2+1/),SHAPE(TRANSPOSE(mMax))))
!
            Do m = 1,mMax(x,n)
 !      Define convenient, repeatedly used row constants
               iTC1=iTC0(s2,m)     ! Top Cap
               iM1=iM0(q,s2,m)     ! Middle
               iBC1=iBC0(q,p,s2,m) ! Bottom Cap
     iRow = TRANSPOSE(RESHAPE((/0*1+iTC1,iM1,iBC1,0,0,0,&
      0*2+iTC1,iM1,iBC1,0,0,0,  0*3+iTC1,iM1,iBC1,0,0,0,&
      0*4+iTC1,iM1,iBC1,0,0,0,  0*5+iTC1,iM1+1,iBC1+1,iTC1+1,iM1,iBC1,&
      0*6+iTC1,iM1,iBC1,iTC1,iM1,iBC1,&
      0*7+iTC1,iM1+1,iBC1+1,iTC1+1,iM1,iBC1,&
      0*8+iTC1,iM1,iBC1,iTC1,iM1,iBC1/),SHAPE(TRANSPOSE(mMax))))
!
     jCol = TRANSPOSE(RESHAPE((/0*1+iTC1,iM1,iBC1,0,0,0,&
      0*2+iTC1,iM1,iBC1,0,0,0,  0*3+iTC1,iM1,iBC1,0,0,0,&
      0*4+iTC1,iM1,iBC1,0,0,0,&
      0*5+iTC1+s2+1,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q-1,iBC1-p-q+s2-2,&
      0*6+iTC1+s2+2,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q,iBC1-p-q+s2-1,&
      0*7+iTC1+s2+1,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q-1,iBC1-p-q+s2-2,&
      0*8+iTC1+s2+2,iM1+q+1,iBC1+p+q-s2+1,iTC1-s2,iM1-q,iBC1-p-q+s2-1/),&
      SHAPE(TRANSPOSE(mMax))))
      !          WRITE(*,*)    '(x,n,(s2,m),(iRow(x,n),jCol(x,n))) = ',&
       !                      (/x,n,s2,m,iRow(x,n),jCol(x,n)/),' si = ',si
                DO alpha2 = s2,-s2,-2       !
                    alpha=dble(alpha2)/2._8
                      beta2=alpha2+dbeta2(x)
                      beta=dble(beta2)/2._8
       r=NINT(si-alpha)+1      ! row in block, MMA: (s2/2 - \[Alpha] + 1)
       c=NINT(si-0.5_8-beta)+1 ! col in block, MMA: ((s2-1)/2 - \[Beta] + 1)
    rTOTAL=r+sumTmatrixDims(iRow(x,n),numTspins,Tspins)  !row in the matrix
    cTOTAL=c+sumTmatrixDims(jCol(x,n),numTspins,Tspins)  !col in the matrix
       IF ((r==0).OR.(c==0)) THEN    ! r=0 or c=0 means the row or column is
           GOTO 300              ! in the wrong block, i.e. the previous block
       END IF                        ! If true, skip this alpha2
    IF ((rTOTAL>dimREP).OR.(cTOTAL>dimREP)) THEN    ! r>max or c>max means the
           GOTO 300                     ! row or column is not in the matrix
    END IF                              ! If true, then skip this alpha2.
```

```
            fHere=f(q,p,s2,m,alpha,beta)  ! the array f(x,n) of the block
 !        component with row/col indices alpha and beta, parameterized by s2,m
! For matrix x, output the row/col indices and the component's value f
 ! WRITE(*,*)'matrix x, row,col,f =',x,rTOTAL,cTOTAL,fHere(x,n)
! output the DO loop values x,n,s2,m,alpha2 plus beta2,r,c,f :
!WRITE(*,*)'x,n,s2,m,alpha2,beta2,r,c,f =',x,n,s2,m,alpha2,beta2,r,c,fHere(x,n)
!     The component of matrix x at row/col rTOTAL,cTOTAL:
                  matrixX1(x,rTOTAL,cTOTAL)= fHere(x,n)
!
    300 CONTINUE   !Bad r,c. We skip alpha2 when r,c=0 or rTOTAL,cTOTAL>dimREP.
                END DO  ! alpha2
              End Do  ! m
            End Do    ! s2
          End Do  ! n
  End Do    ! x
!
ALLOCATE( T3(dimREP,dimREP)) !Get ready to rename the matrixX1s
ALLOCATE( U3(dimREP,dimREP))
ALLOCATE( Tp(dimREP,dimREP))
ALLOCATE( Tm(dimREP,dimREP))
ALLOCATE( Up(dimREP,dimREP))
ALLOCATE( Um(dimREP,dimREP))
ALLOCATE( Vp(dimREP,dimREP))
ALLOCATE( Vm(dimREP,dimREP))
! Identify matrixX solutions as TUV matrices T3, U3, Tp, Tm, Up, Um, Vp, Vm
T3=matrixX1(1,:,:)
U3=matrixX1(2,:,:)
Tp=matrixX1(3,:,:)
Tm=matrixX1(4,:,:)
Up=matrixX1(5,:,:)
Um=matrixX1(6,:,:)
Vp=matrixX1(7,:,:)
Vm=matrixX1(8,:,:)
!
!-----------------8. Extra steps are needed when p < q ---------------------
!
! If the given irrep integers p0 and q0 satisfy  p0 >= q0, then the
! 8 matrices found above make the (p0,q0) irrep.
! For  p0 < q0, we calculated with p = q0 and q = p0, so p > q. The negative
! transpose of the 8 matrices of the (p,q)-irrep make the (p0,q0) irrep.
```

```
! For p0 < q0, we need one more step to get the matrices.
  IF (.NOT.p0GEq0) THEN !  needed for p0 < q0
    Tp = -TRANSPOSE(Tp)
    Tm = -TRANSPOSE(Tm)
    T3 = -TRANSPOSE(T3)
    Up = -TRANSPOSE(Up)
    Um = -TRANSPOSE(Um)
    U3 = -TRANSPOSE(U3)
    Vp = -TRANSPOSE(Vp)
    Vm = -TRANSPOSE(Vm)
  END IF
!
! The 8 matrices Tp,Tm,T3,Up,Um,U3,Vp,Vm have been calculated. Check them.
!
!------9. Check for components that are 'Not a Number', 'NaN'-------------
! Before checking for NANs, start the output here so it looks good
WRITE(*,*) !output a blank line
WRITE(*,*)  'p,q  = ',(/p0,q0/)
! WRITE(*,*)   'To produce an output file, the absolute value of the error in &
  !  any component of the 29 eqns must be less than the MaxErrLimit = ',&
   !                 MaxErrLimit, 'which is set in Sec. 4 "Begin".'
 WRITE(*,*)    'The dimension of the TUV matrices is dimREP = ',dimREP, ' .'
 !      Now, the NaN check:
    k=0     ! k - keep count of the number of NaNs in the TUV matrices
Do x=1,8  !  Check all 8 TUV matrices for NaN entries, NaN = Not A Number
    DO i=1,dimREP   !
        DO j = 1,dimREP
            IF (matrixX1(x,i,j) /= matrixX1(x,i,j)) THEN
                    WRITE(*,*) 'Suspected NaN: x,i,j',x,i,j
                    k=k+1
            END IF
        END DO
    END DO
END DO
        IF (k==0) THEN
                noNaNs=.TRUE.
               WRITE(*,*) 'TUV matrices are filled with numbers, &
                           no Not a Number components, none are "NaN": T'
        ELSE IF (k>0) THEN
                noNaNs=.FALSE.
```

```
            WRITE(*,*) 'TUV matrices have at least one nonnumerical component.'
        END IF
!
! ------10. Check 28 commutator relations and an invariance equation ----------
!
n = dimREP  !n was something else above, now it's the dimension of the matrices
ALLOCATE(T3Tpcomm(n,n),T3Tmcomm(n,n),T3Upcomm(n,n),T3Umcomm(n,n))
ALLOCATE(T3U3comm(n,n),T3Vpcomm(n,n),T3Vmcomm(n,n),TpTmcomm(n,n))
ALLOCATE(TpUpcomm(n,n),TpUmcomm(n,n),TpU3comm(n,n),TpVpcomm(n,n))
ALLOCATE(TpVmcomm(n,n),TmUpcomm(n,n),TmUmcomm(n,n),TmU3comm(n,n))
ALLOCATE(TmVpcomm(n,n),TmVmcomm(n,n),U3Upcomm(n,n),U3Umcomm(n,n))
ALLOCATE(U3Vpcomm(n,n),U3Vmcomm(n,n),UpUmcomm(n,n),UpVpcomm(n,n))
ALLOCATE(UpVmcomm(n,n),UmVpcomm(n,n),UmVmcomm(n,n),VpVmcomm(n,n))
ALLOCATE(SU3identity(n,n))
        T3Tpcomm = MATMUL(T3,Tp) - MATMUL(Tp,T3) - Tp                    ! 1
        MaxErr = MAXVAL(ABS( T3Tpcomm ))
            T3Tmcomm = MATMUL(T3,Tm) - MATMUL(Tm,T3) + Tm
            MaxErr = MAXVAL(ABS( (/MaxErr, T3Tmcomm/) ))
                T3Upcomm = MATMUL(T3,Up) - MATMUL(Up,T3) + Up/2._8
                MaxErr = MAXVAL(ABS( (/MaxErr, T3Upcomm/) ))
                    T3Umcomm = MATMUL(T3,Um) - MATMUL(Um,T3) - Um/2._8
                    MaxErr = MAXVAL(ABS( (/MaxErr, T3Umcomm/) ))
                        T3U3comm = MATMUL(T3,U3) - MATMUL(U3,T3)         ! 5
                        MaxErr = MAXVAL(ABS( (/MaxErr, T3U3comm/) ))
                            T3Vpcomm = MATMUL(T3,Vp) - MATMUL(Vp,T3) - Vp/2.
                            MaxErr = MAXVAL(ABS( (/MaxErr, T3Vpcomm/) ))
                              T3Vmcomm = MATMUL(T3,Vm) - MATMUL(Vm,T3) + Vm/2.
                                MaxErr = MAXVAL(ABS( (/MaxErr, T3Vmcomm/) ))
        TpTmcomm = MATMUL(Tp,Tm) - MATMUL(Tm,Tp) - 2._8*T3
        MaxErr = MAXVAL(ABS( (/MaxErr, TpTmcomm/) ))
            TpUpcomm = MATMUL(Tp,Up) - MATMUL(Up,Tp) - Vp
            MaxErr = MAXVAL(ABS( (/MaxErr, TpUpcomm/) ))
                TpUmcomm = MATMUL(Tp,Um) - MATMUL(Um,Tp)                 ! 10
                MaxErr = MAXVAL(ABS( (/MaxErr, TpUmcomm/) ))
                    TpU3comm = MATMUL(Tp,U3) - MATMUL(U3,Tp) - Tp/2._8
                    MaxErr = MAXVAL(ABS( (/MaxErr, TpU3comm/) ))
                        TpVpcomm = MATMUL(Tp,Vp) - MATMUL(Vp,Tp)
                        MaxErr = MAXVAL(ABS( (/MaxErr, TpVpcomm/) ))
                            TpVmcomm = MATMUL(Tp,Vm) - MATMUL(Vm,Tp) + Um
                            MaxErr = MAXVAL(ABS( (/MaxErr, TpVmcomm/) ))
```

```
        TmUpcomm = MATMUL(Tm,Up) - MATMUL(Up,Tm)
        MaxErr = MAXVAL(ABS( (/MaxErr, TmUpcomm/) ))
            TmUmcomm = MATMUL(Tm,Um) - MATMUL(Um,Tm) + Vm                ! 15
            MaxErr = MAXVAL(ABS( (/MaxErr, TmUmcomm/) ))
                TmU3comm = MATMUL(Tm,U3) - MATMUL(U3,Tm) + Tm/2._8
                MaxErr = MAXVAL(ABS( (/MaxErr, TmU3comm/) ))
                    TmVpcomm = MATMUL(Tm,Vp) - MATMUL(Vp,Tm) - Up
                    MaxErr = MAXVAL(ABS( (/MaxErr, TmVpcomm/) ))
                        TmVmcomm = MATMUL(Tm,Vm) - MATMUL(Vm,Tm)
                        MaxErr = MAXVAL(ABS( (/MaxErr, TmVmcomm/) ))
        U3Upcomm = MATMUL(U3,Up) - MATMUL(Up,U3) - Up
        MaxErr = MAXVAL(ABS( (/MaxErr, U3Upcomm/) ))
            U3Umcomm = MATMUL(U3,Um) - MATMUL(Um,U3) + Um                ! 20
            MaxErr = MAXVAL(ABS( (/MaxErr, U3Umcomm/) ))
                U3Vpcomm = MATMUL(U3,Vp) - MATMUL(Vp,U3) - Vp/2._8
                MaxErr = MAXVAL(ABS( (/MaxErr, U3Vpcomm/) ))
                    U3Vmcomm = MATMUL(U3,Vm) - MATMUL(Vm,U3) + Vm/2._8
                    MaxErr = MAXVAL(ABS( (/MaxErr, U3Vmcomm/) ))
        UpUmcomm = MATMUL(Up,Um) - MATMUL(Um,Up) - 2._8*U3
        MaxErr = MAXVAL(ABS( (/MaxErr, UpUmcomm/) ))
            UpVpcomm = MATMUL(Up,Vp) - MATMUL(Vp,Up)
            MaxErr = MAXVAL(ABS( (/MaxErr, UpVpcomm/) ))
                UpVmcomm = MATMUL(Up,Vm) - MATMUL(Vm,Up) - Tm            ! 25
                MaxErr = MAXVAL(ABS( (/MaxErr, UpVmcomm/) ))
        UmVpcomm = MATMUL(Um,Vp) - MATMUL(Vp,Um) + Tp
        MaxErr = MAXVAL(ABS( (/MaxErr, UmVpcomm/) ))
            UmVmcomm = MATMUL(Um,Vm) - MATMUL(Vm,Um)
            MaxErr = MAXVAL(ABS( (/MaxErr, UmVmcomm/) ))
        VpVmcomm = MATMUL(Vp,Vm) - MATMUL(Vm,Vp) - 2._8*U3 - 2._8*T3       ! 28
        MaxErr = MAXVAL(ABS( (/MaxErr, VpVmcomm/) ))
!
        SU3identity = (MATMUL(Tp,Tm)+MATMUL(Tm,Tp)+MATMUL(Up,Um)+ &
            MATMUL(Um,Up)+MATMUL(Vp,Vm)+MATMUL(Vm,Vp))/2.+MATMUL(T3,T3)+ &
            MATMUL(2._8*U3+T3,2._8*U3+T3)/3._8-(dble(p**2+p*q+q**2)/3._8+ &
            dble(p + q))*unitMatrix                                      ! 29
        MaxErr = MAXVAL(ABS( (/MaxErr, SU3identity/) ))
!
    WRITE(*,*)  'The 28 commutation relations plus one quadratic invariance &
                expression are obeyed: ',(MaxErr.LE.MaxErrLimit)
WRITE(*,*) 'The largest error in the 29 equations is ',MaxErr,','
```

```
WRITE(*,*)  'which should be smaller than MaxErrLimit = ',MaxErrLimit,' which &
               is the allowed tolerance.'
    WRITE(*,*)      ! a blank line
!
    IF(MaxErr.LE.MaxErrLimit) THEN    ! Is the largest error within tolerance?
        MaxErrNotTooBig = .TRUE.        ! TRUE means Yes.
    ELSE IF (MaxErr >MaxErrLimit)    THEN
        MaxErrNotTooBig = .FALSE.       ! FALSE means No, not within tolerance.
    END IF
!
!
!----------11. Save the results to a file, end program -----------------------
!
 !MaxErrNotTooBig = .FALSE.    ! Use this statement to test the failure option
 !noNaNs = .FALSE.             ! Use this statement to test the failure option
 ! MaxErrNotTooBig = .TRUE.    ! Use this statement to make an output
 ! noNaNs = .TRUE.             ! Use this statement to make an output
!
    IF ((noNaNs).AND.(MaxErrNotTooBig)) THEN            ! No Failure(s) found
        WRITE (file_name,"('p',i0,'q',i0,'SU3gen.dat')")p0,q0
        OPEN(Unit=5,file=file_name)
        WRITE(5,3000) p0,q0, MaxErr   ! Start the data file with p0,q0,MaxErr.
        CLOSE(5)    ! Next, the 8*dimREP**2 components of the 8 basis matrices
    OPEN(Unit=5,file=file_name,STATUS='OLD', POSITION='APPEND') ! Append T,U,V
    WRITE(5,4000) TRANSPOSE(T3),TRANSPOSE(U3),TRANSPOSE(Tp), TRANSPOSE(Tm),&
                  TRANSPOSE(Up), TRANSPOSE(Um), TRANSPOSE(Vp), TRANSPOSE(Vm)
      CLOSE(5)
    END IF
!
  3000 FORMAT(2I3,1E16.7) !FORMAT statement for two integers and a real number
  4000 FORMAT(5F16.12)!The FORMAT statement for the matrices,5 numbers per line
!
IF(ALLOCATED(unitMatrix)) DEALLOCATE(unitMatrix)
IF(ALLOCATED(Tspins)) DEALLOCATE(Tspins)
IF(ALLOCATED(T3)) DEALLOCATE(T3)    ! The program calculates irreps for
IF(ALLOCATED(U3)) DEALLOCATE(U3)    ! many different (p,q). That means the
IF(ALLOCATED(Tp)) DEALLOCATE(Tp)    ! space in memory changes for many of the
IF(ALLOCATED(Tm)) DEALLOCATE(Tm)    ! variables. So they must be DEALLOCATED.
IF(ALLOCATED(Up)) DEALLOCATE(Up)
IF(ALLOCATED(Um)) DEALLOCATE(Um)
```

```
IF(ALLOCATED(Vp)) DEALLOCATE(Vp)
IF(ALLOCATED(Vm)) DEALLOCATE(Vm)
IF(ALLOCATED(matrixX1)) DEALLOCATE(matrixX1)
IF(ALLOCATED(T3Tpcomm)) DEALLOCATE(T3Tpcomm)
IF(ALLOCATED(T3Tmcomm)) DEALLOCATE(T3Tmcomm)
IF(ALLOCATED(T3Upcomm)) DEALLOCATE(T3Upcomm)
IF(ALLOCATED(T3Umcomm)) DEALLOCATE(T3Umcomm)
IF(ALLOCATED(T3U3comm)) DEALLOCATE(T3U3comm)
IF(ALLOCATED(T3Vpcomm)) DEALLOCATE(T3Vpcomm)
IF(ALLOCATED(T3Vmcomm)) DEALLOCATE(T3Vmcomm)
IF(ALLOCATED(TpTmcomm)) DEALLOCATE(TpTmcomm)
IF(ALLOCATED(TpUpcomm)) DEALLOCATE(TpUpcomm)
IF(ALLOCATED(TpUmcomm)) DEALLOCATE(TpUmcomm)
IF(ALLOCATED(TpU3comm)) DEALLOCATE(TpU3comm)
IF(ALLOCATED(TpVpcomm)) DEALLOCATE(TpVpcomm)
IF(ALLOCATED(TpVmcomm)) DEALLOCATE(TpVmcomm)
IF(ALLOCATED(TmUpcomm)) DEALLOCATE(TmUpcomm)
IF(ALLOCATED(TmUmcomm)) DEALLOCATE(TmUmcomm)
IF(ALLOCATED(TmU3comm)) DEALLOCATE(TmU3comm)
IF(ALLOCATED(TmVpcomm)) DEALLOCATE(TmVpcomm)
IF(ALLOCATED(TmVmcomm)) DEALLOCATE(TmVmcomm)
IF(ALLOCATED(U3Upcomm)) DEALLOCATE(U3Upcomm)
IF(ALLOCATED(U3Umcomm)) DEALLOCATE(U3Umcomm)
IF(ALLOCATED(U3Vpcomm)) DEALLOCATE(U3Vpcomm)
IF(ALLOCATED(U3Vmcomm)) DEALLOCATE(U3Vmcomm)
IF(ALLOCATED(UpUmcomm)) DEALLOCATE(UpUmcomm)
IF(ALLOCATED(UpVpcomm)) DEALLOCATE(UpVpcomm)
IF(ALLOCATED(UpVmcomm)) DEALLOCATE(UpVmcomm)
IF(ALLOCATED(UmVpcomm)) DEALLOCATE(UmVpcomm)
IF(ALLOCATED(UmVmcomm)) DEALLOCATE(UmVmcomm)
IF(ALLOCATED(VpVmcomm)) DEALLOCATE(VpVmcomm)
IF(ALLOCATED(SU3identity)) DEALLOCATE(SU3identity)
!
 END DO
        END PROGRAM Su3Generators
!
!-----------------12. External functions, end-of-file-----------------------
!   iTC0,iM0,iBC0 appear in the block row arrays iRow  and jCol
FUNCTION iTC0(s2,m) RESULT  (w)    !iTC0 - Top Cap block row function
    IMPLICIT NONE                  !
```

```
    INTEGER :: w
    INTEGER, INTENT(IN) :: s2,m
            w = s2*(s2+1)/2 + m
END FUNCTION

FUNCTION iM0(q,s2,m) RESULT  (w)    !iM0 - Middle block row function
    IMPLICIT NONE
    INTEGER :: w     !
    INTEGER, INTENT(IN) :: q,s2,m    !
            w = (2*s2-q)*(q+1)/2 + m
END FUNCTION

FUNCTION iBC0(q,p,s2,m) RESULT  (w)    !iBC0 Bottom Cap block row function
    IMPLICIT NONE
    INTEGER :: w     !
    INTEGER, INTENT(IN) :: q,p,s2,m    !
            w = ((2*s2-q)*(q+1)+(2*s2-p)*(p+1))/2 - s2*(s2+1)/2 + m
END FUNCTION
!The nth block starts at the end of the first n-1 Tspin blocks
FUNCTION sumTmatrixDims(n,numTspins,Tspins) RESULT  (w)
  IMPLICIT NONE     ! sumTmatrixDims - sum the sizes of the first n-1 blocks
    INTEGER, INTENT(IN) :: n,numTspins   !  nth spin, total number of Tspins
    REAL(8), INTENT(IN) :: Tspins(numTspins)   ! the list of Tspins
    INTEGER :: i,k,w        ! i,k - dummy indices, the result w is an integer
    REAL(8) :: TspinN            ! dummy spin, the ith Tspin
                k = 0            ! initialize k
    DO i=1,n-1            ! sum the sizes of the first n-1 Tspin matrices
                TspinN = Tspins(i)  ! TspinN - the ith Tspin
                k = k + NINT(2.*TspinN)+1  ! the number of spin components in
                                          !a Tspin block is 2*s+1
    END DO
                w=k          ! the result is the sum of Tspin block sizes
END FUNCTION
! The values of the components are found with the functions 'f'
FUNCTION f(q,p,s2,m,alpha,beta) RESULT  (w)
    IMPLICIT NONE           !The functions f form an 8x6 array, with 8 choices
    REAL(8) :: w(8,6),si,z ! for the matrixX1 and 6 choices for spin section.
    INTEGER :: q,p,s2,m !
    REAL(8), INTENT(IN) :: alpha,beta
    si = dble(s2)/2._8
```

```
   z=0.0_8     ! I use z to keep track of the eight matrices.
  w=TRANSPOSE(RESHAPE((/&
z*1._8+alpha,alpha,alpha,z,z,z,&  ! z = 0 so z*1 = 0, z*2 = 0,...
z*2._8+(DBLE(-3 - p + q + 3*m) - 3._8*si - alpha)/2._8,&
  (DBLE(-3 - p -2*q + 3*m) + 3._8*si - alpha)/2._8,&
  (DBLE(-3 - p -2*q + 3*m) + 3._8*si - alpha)/2._8,z,z,z,&
z*3._8+DSQRT((si + alpha)*(1._8 + si - alpha)),&
   DSQRT((si + alpha)*(1._8 + si - alpha)),&
   DSQRT((si + alpha)*(1._8 + si - alpha)),z,z,z,&
z*4._8+DSQRT((si - alpha)*(1._8 + si + alpha)),&
   DSQRT((si - alpha)*(1._8 + si + alpha)),&
   DSQRT((si - alpha)*(1._8 + si + alpha)),z,z,z,&
z*5._8+DSQRT(DBLE((2 - m + s2)*(3 + p - m + s2)*(-1 + q + m - s2))*&
         (1._8 + si + alpha)/DBLE((1 + s2)*(2 + s2))),&
   DSQRT(DBLE(m*(1 + q - m)*(2 + p + q - m))*(1._8 + si + alpha)/&
           DBLE((1 + s2)*(2 + s2))),&
   DSQRT(DBLE(m*(1 + q - m)*(2 + p + q - m))*(1._8 + si + alpha)/&
           DBLE((1 + s2)*(2 + s2))),&
   DSQRT(DBLE(m*(1 + p - m)*(1 + q + m))*(si - alpha)/DBLE(s2*(1 + s2))),&
   DSQRT(DBLE((m + s2)*(-1 - q + m + s2)*(2 + p + q - m - s2))*(si - alpha)/&
           DBLE(s2*(1 + s2))),&
   DSQRT(DBLE((m + s2)*(-1 - q + m + s2)*(2 + p + q - m - s2))*(si - alpha)/&
           DBLE(s2*(1 + s2))),&
z*6._8+DSQRT(DBLE(m*(1 + p - m)*(1 + q + m))*(1._8 + si - alpha)/&
           DBLE((2 + s2)*(1 + s2))),&
   DSQRT(DBLE((1 + m + s2)*(-q + m + s2)*(1 + p + q - m - s2))*&
           (1._8 + si - alpha)/DBLE((2 + s2)*(1 + s2))),&
   DSQRT(DBLE((1 + m + s2)*(-q + m + s2)*(1 + p + q - m - s2))*&
           (1._8 + si - alpha)/DBLE((2 + s2)*(1 + s2))),&
   DSQRT(DBLE((1 - m + s2)*(2 + p - m + s2)*(q + m - s2))*&
           ((si + alpha))/DBLE(s2*(1 + s2))) ,&
   DSQRT(DBLE(m*(1 + q - m)*(2 + p + q - m))*(si + alpha)/DBLE(s2*(1 + s2))),&
   DSQRT(DBLE(m*(1 + q - m)*(2 + p + q - m))*(si + alpha)/DBLE(s2*(1 + s2))),&
z*7._8-DSQRT(DBLE((2 - m + s2)*(3 + p - m + s2)*(-1 + q + m - s2))*&
           (1._8 + si - alpha)/DBLE((2 + s2)*(1 + s2))),&
  -DSQRT(DBLE(m*(1 + q - m)*(2 + p + q - m))*(1._8 + si - alpha)/&
           DBLE((2 + s2)*(1 + s2))),&
  -DSQRT(DBLE(m*(1 + q - m)*(2 + p + q - m))*(1._8 + si - alpha)/&
           DBLE((2 + s2)*(1 + s2))),&
   DSQRT(DBLE(m*(1 + p - m)*(1 + q + m))*(si + alpha)/DBLE(s2*(1 + s2))),&
```

```
   DSQRT(DBLE((m + s2)*(-1 - q + m + s2)*(2 + p + q - m - s2))*(si + alpha)/&
            DBLE(s2*(1 + s2))),&
   DSQRT(DBLE((m + s2)*(-1 - q + m + s2)*(2 + p + q - m - s2))*(si + alpha)/&
            DBLE(s2*(1 + s2))),&
 z*8._8+DSQRT(DBLE(m*(1 + p - m)*(1 + q + m))*(1._8 + si + alpha)/&
            DBLE((2 + s2)*(1 + s2))),&
   DSQRT(DBLE((1 + m + s2)*(-q + m + s2)*(1 + p + q - m - s2))*&
            (1._8 + si + alpha)/DBLE((2 + s2)*(1 + s2))),&
   DSQRT(DBLE((1 + m + s2)*(-q + m + s2)*(1 + p + q - m - s2))*&
            (1._8 + si + alpha)/DBLE((2 + s2)*(1 + s2))),&
  -DSQRT(DBLE((1 - m + s2)*(2 + p - m + s2)*(q + m - s2))*(si - alpha)/&
            DBLE(s2*(1 + s2))),&
  -DSQRT(DBLE(m*(1 + q - m)* (2 + p + q - m))*(si - alpha)/DBLE(s2*(1 + s2))),&
  -DSQRT(DBLE(m*(1 + q - m)*(2 + p + q - m))*(si - alpha)/&
            DBLE(s2*(1 + s2)))/),SHAPE(TRANSPOSE(w))))
END FUNCTION
! end-of-file
```

8 Author Declarations and License Information

The author has no conflicts to disclose. The computer programs and this article are openly licensed under CC BY-SA 4.0.[16]

References

[1] Wu-Ki Tung, *Group Theory in Physics*, (World Scientific, Philadelphia, 1985)

[2] A. R. Edmonds, *Angular Momentum in Quantum Mechanics*, 2nd edition (Princeton University Press, Princeton, 1960).

[3] Morris E. Rose, *Elementary Theory of Angular Momentum*, 1st edition (John Wiley & Sons, Inc., New York, 1957).

[4] Steven Weinberg, *The Quantum Theory of Fields, Vol. I*, 1st edition (Cambridge University Press, Cambridge, 1995), Sec. 5.6.

[5] R. Shurtleff, arXiv:math-ph/0401002, (2004).

[6] G. Ya. Lyubarskii, transl. by S. Dedijer, *The Applications of Group Theory in Physics*(Pergamon Press, Oxford, 1960), Chap. XVI.

[7] Brian C. Hall, *Lie Groups, Lie Algebras, and Representations*, 2nd edition (Springer Cham, N.Y., 2015) and references therein.

[8] G. E. Baird and L. C. Biedenharn, "On the Representations of the Semisimple Lie Groups. II," J. Math. Phys. **4**, 1449–1466 (1963).

[9] Stephan Gasiorowicz, *Elementary Particle Physics*, (John Wiley & Sons, Inc., New York, 1966), Ch. 17.

[10] Walter Greiner and Berndt Müller, *Quantum Mechanics, Symmetries*, 2nd revised edition (Springer-Verlag, Berlin, 1994).

[11] V. B. Mandel'tsveig, "Irreducible Representations of the SU_3 Group," Soviet Physics JETP **20**, 1237–1243 (1965).

[12] Wolfram Research, Inc., Mathematica, Version 14.0, Champaign, IL (2023).

[13] link to the Mathematica program,
https://www.wolframcloud.com/obj/shurtleffr/Published/SU3MatricesMMA2024.nb

[14] link to the Mathematica program,
https://www.dropbox.com/scl/fi/ey0wk7cvhupeqeeeowr1v/MMAforPAPER3.nb?rlkey=
7jhigi9m8lnxo862c78jdznw6&dl=0

[15] link to the Fortran 90 program,
https://www.dropbox.com/scl/fi/zia4anqs72e2bqsja5f6t/SU3GENpPLUSq.DP3.f90?rlkey=
f9mhjm18mnguc8yapow7l3z3s&st=ad61hj3m&dl=0