JOURNAL OF COMPUTATIONAL PHYSICS 1, 397-405 (1967)

Computational Problems Associated with Racah Algebra[1]

J. STEIN

The Hebrew University, Jerusalem, Istael

ABSTRACT

In a program written by the author, for performing various calculations in the Racah algebra, some new calculational methods were used; these methods are described in this paper. They include: (1) a representation of numbers alternative to the Rotenberg form; (2) a new algorithm for calculating the greatest common devisor of two odd integers, which was developed for reducing the calculating time in the new representation; (3) methods for shortening the computation time of 3-j and 6-j symbols.

INTRODUCTION

In atomic spectroscopy, it is often necessary to calculate matrices, the elements of which are functions of 6-j or 3-j coefficients. These matrices can be calculated in two ways: one is by using floating-point numbers; the other [1], [2] by exact calculation, using the fact that these numbers can be expressed in the form $\pm (a/b)^{1/2}$ where a and b are integers [3].

Most of the large computers are capable of doing fast floating-point calculations with a high order of precision (hardware or software), so that the first way is accurate enough, simpler than the second, and can be more easily used by FORTRAN programming. Nevertheless many physicists prefer the second way because of two reasons. (1) these matrices are standard matrices, calculated "once and for ever" and are used for many applications where various degrees of preci-

[1] This paper was partially supported by the A.F.C.S. through the European office, Aerospace Research, U. S. Air Force.

sion are needed. The best way of keeping standard matrices is to have them in absolute accuracy. (2)—a more "emotional" reason—there are those aesthetic people who like "nice" numbers such as $(105/1024)^{1/2}$ and who do not like "dirty" numbers such as 3.141592653589. These people would prefer, of course, the second way.

In this paper we describe some of the methods we used in a program for calculating angular integrals which appear in the energy matrix of electrons and nucleons in various configurations.

The following sections contain

(1) A number form that can be used as an alternative to the Rotenberg form [4]. Arithmetic calculations in this form make use of a new algorithm for calculating the greatest common divisor (GCD) of two integers.

(2) Methods to shorten the computation time of 3-j or 6-j symbols.

I. Multiprecision Arithmetic

Rotenberg [4] numbers[2] are very useful for calculations in the Racah algebra; they have, however, one disadvantage: they take up much computer memory. As the computer we used had only $8K$ memory locations we had to use a more compact representation of the numbers, which is described in this section. This representation has also the advantage of not having the limitations of the Rotenberg representation. The price that we had to pay for these advantages was execution time; mainly in the multiplications. We could not compare times with the Rotenberg numbers because there were no subroutines for addition and multiplication of Rotenberg numbers on the PHILCO 2000.

Every number in the Racah Algebra can be expressed in the form

[2] In the Rotenberg form [4] the number a is decomposed in the form

$$a = b \cdot \prod_{1}^{N} p_k^{n_k} = b \cdot 2^{n_1} \cdot 3^{n_2} \cdot 5^{n_3} \cdot 7^{n_4} \cdot \ldots \cdot p_N^{n_N}$$

where p_k are the prime numbers in ascending order, n_k are integers or half-integers and b is an integer which is not divisible by $p_1 \ldots p_N$. The number appears in the computer memory as one integer b and a vector of the small integers $2n_1 \ldots 2n_N$. In practical calculations, N must be large enough so that, in all the factorials which appear in the 3-j or 6-j symbols, b will be equal to one. A generalization of this representation where b can be a fraction or a root is much less useful, and in the rare cases where it is needed, our procedure is much simpler and, probably, shorter.

$\pm (a/b)^{1/2} \cdot 2^{k/2}$ where a and b are odd integers, and k is an integer. For most purposes single- or double-precision integers a, b are enough (for computers having 48 bits per word). A number in this form takes three or five words in a computer memory (if one does not exceed double precision): one for the sign and powers of 2; one or two for a; one or two for b (the reason for using a and b odd will be explained soon). A good arrangement of the auxiliary word is: the sign in the lowest bit and the powers of 2 in the left half. The numbers will be called a, b, and p where $p = (Q, S)$ Q = powers of 2, S = sign (1 for minus).

In this paper we shall use the abbreviation GCD for greatest common divisor.

Multiplication

$$(a_1, b_1, p_1) * (a_2, b_2, p_2) \rightarrow (a_3, b_3, p_3)$$

where

$$a_3 = \frac{a_1}{\mathrm{GCD}(a_1, b_2)} * \frac{a_2}{\mathrm{GCD}(a_2, b_1)},$$

$$b_3 = \frac{b_1}{\mathrm{GCD}(a_2, b_1)} * \frac{b_2}{\mathrm{GCD}(a_1, b_2)},$$

$$p_3 = p_1 + p_2.$$

So, if a_1/b_1 and a_2/b_2 are reduced to the lowest terms, so also is a_3/b_3.

Addition:

$$(a_1, b_1, p_1) + (a_2, b_2, p_2) \rightarrow (a_3, b_3, p_3).$$

Let us assume that $Q_1 < Q_2$ and that the two numbers have the same sign; then

$$b_3' = \frac{b_1 * b_2}{\mathrm{GCD}(b_1, b_2)}, \qquad T = \mathrm{GCD}(a_1, a_2),$$

$$a_1'' = a_1/T, \qquad a_2'' = a_2/T,$$

$$a_1' = \frac{a_1''}{b_1} * b_3', \qquad a_2' = \frac{a_2''}{b_2} * b_3' * 2^{Q_2 - Q_1},$$

$$a_3' = ((a_1')^{1/2} + (a_2')^{1/2})^2.$$

Now we shift a_3' to the right as far as possible and obtain

$$a_3'' = a_3' \cdot 2^{-R},$$

$$a_3 = (a_3''/\mathrm{GCD}(a_3'', b_3')) * T, \qquad b_3 = b_3'/GCD(a_3'', b_3'),$$
$$Q_3 = Q_1 + R,$$
$$S_3 = S_1[= S_2].$$

Greatest Common Divisor

The bottleneck of the addition and the multiplication is the calculation of the GCD of various numbers. The old algorithm for this is the algorithm of Euclid [5]. Here a new algorithm is developed, adopted specially to binary computers.

The new algorithm is based on the obvious statement that if a and b are odd integers and $a > b$, then $\mathrm{GCD}(a, b) = \mathrm{GCD}(b, (a - b)/2)$.

Let a and b be two odd integers,

$$a_1 = \max(a, b), \qquad b_1 = \min(a, b);$$

then

$$c_1 = a_1 - b_1.$$

Now we shift c_1 to the right as far as possible without losing nonzero digits; this operation will be called SHIFT.

$$d_1 = \mathrm{SHIFT}(c_1)$$
$$a_2 = \max(d_1, b_1), \qquad b_2 = \min(d_1, b_1)$$
$$d_2 = \mathrm{SHIFT}(a_2 - b_2)$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$a_n = \max(d_{n-1}, b_{n-1}), \qquad b_n = \min(d_{n-1}, b_{n-1})$$
$$d_n = \mathrm{SHIFT}(a_n - b_n).$$

The procedure continues until $a_n = b_n[= \mathrm{GCD}(a, b)]$.

The number of steps is at most equal to the length of a_1 (in bits) minus 1. On the average, if $\mathrm{GCD}(a, b) = 1$, the number of steps is equal to half of that length.

It is suggested that the first step will always be a division to equalize the order of magnitude of the two numbers and then to proceed by the new algorithm.

This new way is much faster than the old one, mainly because of the long

division time compared with the short addition or shifting time. The convergence speed is also better by the factor 2 than in the old algorithm. When one goes to a higher precision than double, the divisions become intolerably complicated and long, and then this method becomes even more powerful.

This discussion would not be complete without mentioning an alternative way, which, although not used by us, is very reasonable for certain computers. In the algorithm described above, we eliminated the lowest bit in each step and then shifted to the right; in the alternative algorithm, we eliminate the left-most bit in each step and then shift to the left.

The method is based on the instruction "NORMALIZE" which exist in some computers (e.g., the CDC 6000 series or the IBM 1130). This instruction is normally used for normalizing unnormalized floating-point numbers and to convert fixed —to floating-point numbers, by shifting them to the left in one register and counting the number of shifts in another register.

The advantage of using left shifts instead of right shifts is that the right shifts are done by a loop of shifts and tests, while the left shifts can be done by a single instruction, which also counts the shifts.

This left-shift method is based on a somewhat less obvious theorem than the first one:

If a and b are positive integers and if $a > b$, then

$$\mathrm{GCD}(a, b) = \mathrm{GCD}(b, a - 2^k b)$$

where $k = [\log_2 a] - [\log_2 b]$ (the brackets indicate that the integer part of the log must be taken).

The proof is obvious, except for the powers of 2, because this time we did not demand that a and b be odd. We shall, therefore, prove it for the powers of 2.

Let us assume that $a = c \cdot 2^m$ and $b = d \cdot 2^n$ where c and d are odd integers, and therefore $\mathrm{GCD}(a, b) = e \cdot 2^{\min(m,n)}$ where e is also an odd integer; all we have to prove is that if $a - 2^k \cdot b = f \cdot 2^q$, then $\min(m, n) = \min(n, q)$.

If $m \geq n$, then

$$a - 2^k b = (c \cdot 2^{m-n} - d \cdot 2^k) \cdot 2^n;$$

therefore $q \geq n$, and $\min(m, n) = n = \min(n, q)$.

If $m < n$, then

$$a - 2^k b = (c - d \cdot 2^{n+k-m}) \cdot 2^m;$$

$n > m$, so $c - d \cdot 2^{n+k-m}$ is odd, and therefore $q = m$, and $\min(m, n) = \min(n, q)$. Q.E.D.

Let $N(a)$ be the "normalized" a, and $S(a)$ the number of left shifts needed for the "normalization". Suppose that $a > b$; then GCD(a, b) can be calculated by the following procedure;

$$\begin{aligned} a_1 &= N(a), & S_1 &= S(a) \\ b_1 &= N(b), & T_1 &= S(b) \\ & & U_1 &= S_1, \\ a_2 &= N(|\, a_1 - b_1 \,|), & S_2 &= S(|\, a_1 - b_1 \,|) + U_1, \\ b_2 &= m(b_1, a_2), & T_2 &= \max(S_2, T_1), \\ & & U_2 &= \min(S_2, T_1), \end{aligned}$$

where $m(b_1, a_2)$ is equal to b_1 if $T_1 > S_2$, and to a_2 if $T_1 < S_2$.

$$\begin{aligned} a_3 &= N(|\, a_2 - b_1 \,|), & S_3 &= S(|\, a_2 - b_1 \,|) + U_2, \\ b_3 &= m(a_3, b_2), & T_3 &= \max(S_3, T_2), \\ & & U_3 &= \min(S_3, T_2), \\ & \cdot & & \\ & \cdot & & \\ & \cdot & & \\ & \cdot & & \end{aligned}$$

$$\text{for } n > 2: \quad \begin{aligned} a_n &= N(|\, a_{n-1} - b_{n-2} \,|), & S_n &= S(|\, a_{n-1} - b_{n-2} \,|) + U_{n-1}, \\ b_n &= m(a_n, b_{n-1}), & T_n &= \max(S_n, T_{n-1}), \\ & & U_n &= \min(S_n, T_{n-1}). \end{aligned}$$

The calculation terminates when $a_{n+1} = 0$, and then

$$\text{GCD}(a, b) = a_n \cdot 2^{-S_n}$$

This method, like the previous one, can also be used with multiple-precision numbers, where, of course, the shifting will be a little more complicated; but there is no essential difference.

II. Calculation of 3-j and 6-j Symbols

There is no doubt that the best way to calculate Racah and Wigner coefficients at random is by using Rotenberg numbers, and the simplest formulas for these

coefficients [6]. The only problem is how to minimize the time needed for these calculations. For this purpose the following methods are useful.

(1) 3-j and 6-j symbols are build up from factorials. The calculation of these factorials is long and each 6-j symbol contains at least 24 of them. The calculation time can therefore be reduced by a large factor if one keeps, in the computer-memory, a table of all the factorials one may need in Rotenberg form. The number of factorials needed is not great: it is equal to $4J_{\max} + 1$, when $J_{\max}$ is the largest value of J used in the calculation. In all the applications to atomic spectroscopy in the near future $J_{\max} = 50$ seems to be enough; which means 201 factorials.

(2) Additions and subtractions are the bottleneck of the calculation of the 3-j and 6-j symbols. In order to make the additions as short as possible one should avoid the use of subroutines for adding pairs of Rotenberg numbers, because these subroutines cause a great waste of computer time if the sum to be evaluated contains more than two terms. When one has to add three or more terms in a calculation of a 3-j or a 6-j symbol, then for each pair of numbers the subroutine finds the common denominator, takes out the GCD of the two numerators, transforms what remains of the numerators into integers, adds, and then *transforms the sum back to the Rotenberg form.* This final transformation is the longest part of the addition, because the computer has to try to divide the sum by all the prime numbers up to $4J_{\max} + 1$ where $J_{\max}$ is the greatest J in the calculation. In order to save computer time when there are more than two terms to add, one should postpone the transformation back to Rotenberg numbers, to the end of the addition process. (When there are many terms in the sum, it is useful to divide it into smaller groups of 5–10 terms, and so to reduce the use of multiprecision integers). For this purpose, we have written a generalized addition subroutine, as a part of the 6-j subroutine, based on the following algorithm:

Consider a sum of n terms, $n \geq 2$: $A_1, A_2, A_3, \ldots, A_n$

(1) Calculate the greatest common divisor (GCD) of A_1, A_2.

$$R_2 = \mathrm{GCD}(A_1, A_2), \quad A_1' = A_1/R_2, \quad A_2' = A_2/R_2.$$

Convert A_1' and A_2' to integers and add: $B_2 = A_1' + A_2'$.

(2) Calculate the $\mathrm{GCD}(R_2, A_3)$: $R_3 = \mathrm{GCD}(R_2, A_3)$;

$$A_3' = A_3/R_3, \qquad R_2' = R_2/R_3.$$

Convert R_2' and A_3' to integers and add: $B_3 = B_2 * R_2' + A_3'$

$$(n-1)\quad R_n = \mathrm{GCD}(R_{n-1}, A_n) \qquad \text{(Rotenberg)},$$

$$A_n' = A_n/R_n, \quad R_{n-1}' = R_{n-1}/R_n \qquad \text{(Rotenberg)}.$$

Convert A_n' and R'_{n-1} to integers and add:

$$B_n = B_{n-1} * R'_{n-1} + A_n' \qquad \text{(integers)}.$$

At the end, transform B_n to the Rotenberg form.

Timing

In order to compare computation times of 6-j symbols in our method to other programs, we wrote a 6-j subroutine for the IBM 7040. The times were compared to those quoted in [1]. In this comparison it is necessary to take into account that the times of [1] were measured on the IBM 7090[3] which is about 4 *times* as fast as the 7040.

6-j	Time 7040	Time 7090
$\begin{Bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{Bmatrix}$	27 msec	12 msec
$\begin{Bmatrix} 8 & 8 & 8 \\ 8 & 8 & 8 \end{Bmatrix}$	114 msec	160 msec
$\begin{Bmatrix} 16 & 16 & 16 \\ 16 & 16 & 16 \end{Bmatrix}$	300 msec	1 sec

The greatest value of j for which our subroutine can calculate $\begin{Bmatrix} j & j & j \\ j & j & j \end{Bmatrix}$ using only single-precision integers (35 bits) is 8. Using double-precision integers we can go up to $j = 16$. The subroutine is not built for integers of greater than double-precision, but it can be modified to use higher precision without increasing the calculation-time of those 6-j symbols for which lower precision is enough.

ACKNOWLEDGEMENT

We are indebted to Professor J. M. Blatt and to Dr. Y. Bordarier for fruitful discussions of the methods described in this paper.

[3] From [1] it is not clear whether the times were measured on the IBM 7090 or on the 7094, which is much faster; however, for our comparison, it is enough to assume that the times were measured on the 7090.

References

1. R. M. Baer and M. G. Redlich, *Commun. ACM* **7**, 657 (1964).
2. Y. Bordarier (unpublished).
3. A. R. Edmonds, "Angular Momentum in Quantum Mechanics," Princeton University Press, Princeton, New Jersey (1957).
4. M. Rotenberg, R. Bivins, N. Metropolis, and J. K. Wooten, "The 3-j and 6-j Symbols," Technology Press, Cambridge, Massachusetts (1959).
5. Euclid, "The Elements," Vol. VII, Propositions 1–3 (circa 300 BC).
6. U. Fano and G. Racah "Irreducible Tensorial Sets," Academic Press, New York (1959).