Numer. Math. 16, 145—156 (1970)

Error Analysis of Algorithms for Matrix Multiplication and Triangular Decomposition using Winograd's Identity

R. P. BRENT

Received December 3, 1969

Summary. The number of multiplications required for matrix multiplication, for the triangular decomposition of a matrix with partial pivoting, and for the Cholesky decomposition of a positive definite symmetric matrix, can be roughly halved if Winograd's identity is used to compute the inner products involved. Floating-point error bounds for these algorithms are shown to be comparable to those for the normal methods provided that care is taken with scaling.

1. Introduction

Winograd [8] has given an algorithm for matrix multiplication which involves only about half as many multiplications as the normal method, although about 50% more additions are required. The algorithm is based on the following identity for the inner product of two n-vectors, n even:

$$\sum_{i=1}^{n} x_i y_i = \sum_{i=1}^{n/2} (x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1}) - (\xi + \eta), \tag{1.1}$$

where

$$\xi = \sum_{i=1}^{n/2} x_{2i-1} x_{2i}$$

and

$$\eta = \sum_{i=1}^{n/2} y_{2i-1} y_{2i}.$$

(1.1) holds in any commutative ring, but we shall restrict our attention to the real and complex fields and their limited-precision floating-point approximations.

For matrix multiplication, $C = AB$ say, a number ξ is computed for each row of A, a number η for each column of B, and then (1.1) is used to compute the inner products giving C. If A and B are n by n matrices, n even, the computation involves $\frac{1}{2}n^3 + n^2$ multiplications and $3n^3/2 + 2n(n-1)$ additions, compared to the usual n^3 and $n^3 - n^2$ respectively (we shall never distinguish between additions and subtractions). Hence we can expect a saving of up to 50% in the computation time, for moderate and large matrices, if a multiplication takes considerably longer than an addition. This is borne out in practice: for some experiments with real and complex matrices on an IBM 360/67 computer, and comparisons with other methods, see [1].

In the next section we shall give a floating-point error analysis of Winograd's algorithm for matrix multiplication. The analysis shows that Winograd's identity

(1.1) is practically useful for floating-point computation if, and only if, care is taken with scaling. We shall then describe numerically satisfactory algorithms for the LU decomposition of a square matrix with partial pivoting, and for the Cholesky decomposition of a real positive definite symmetric matrix, using (1.1). In all these algorithms the use of (1.1) roughly halves the number of multiplications required, compared to the number required with the usual algorithms.

2. Rounding-Error Analysis of Winograd's Algorithm

Consider using t-digit rounded floating-point arithmetic to base $\beta \geqq 2$. We assume, following Wilkinson [6], that

$$fl(x \times y) = (x \times y)(1+\varepsilon_1) \tag{2.1}$$

and

$$fl(x \pm y) = x(1+\varepsilon_2) \pm y(1+\varepsilon_3),$$

where

$$|\varepsilon_i| \leqq \tfrac{1}{2}\beta^{1-t}.$$

If β and t are suitably chosen these assumptions should be valid for any reasonable computer, and they should also hold for complex arithmetic if t is reduced slightly.

To find the inner product $\sum_{i=1}^{n} x_i y_i$ by "Winograd's method" for even $n \geqq 2$, we compute

$$w = fl(\theta - \zeta), \tag{2.2}$$

where

$$\theta = fl\left(\sum_{i=1}^{n/2} (x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1})\right),$$

$$\zeta = fl(\xi + \eta),$$

$$\xi = fl\left(\sum_{i=1}^{n/2} x_{2i-1} x_{2i}\right),$$

and

$$\eta = fl\left(\sum_{i=1}^{n/2} y_{2i-1} y_{2i}\right).$$

If n is odd, the term $fl(x_n y_n)$ is added to the partial inner product $\sum_{i=1}^{n-1} x_i y_i$, computed by (2.2) with n replaced by $n-1$. When w is computed in this way we write

$$w = fl_W \sum_{i=1}^{n} x_i y_i.$$

A simple example shows what can happen: take $n=2$, $\beta=10$, $t=4$, $x_1 = x_2 = 1.000_{10} + 2$, $y_1 = y_2 = 1.000_{10} - 2$. Then $\xi = 1.000_{10} + 4$ and $\eta = 1.000_{10} - 4$ are both exactly correct, $\zeta = 1.000_{10} + 4$ and $\theta = 1.000_{10} + 4$ have small relative errors, but $w = 0.000$ instead of 2.000. The difficulty lies in forming $fl(x_{2i-1} + y_{2i})$ and $fl(x_{2i} + y_{2i-1})$ when $\|x\|$ and $\|y\|$ have widely different magnitudes. In the next section we shall describe how this difficulty can be overcome, but first we derive some rigorous "worst case" error bounds.

For brevity let $a = \max |x_i|$, $b = \max |y_i|$, and with an obvious notation write

$$\varepsilon_\xi = \xi - \sum_{i=1}^{n/2} x_{2i-1} x_{2i} \quad \text{etc.}$$

We shall assume that

$$n \cdot \tfrac{1}{2}\beta^{1-t} \leqq 1 \tag{2.3}$$

and let

$$t_1 = t - \frac{n\,\beta^{1-t}}{\log \beta}. \tag{2.4}$$

t_1 is chosen in this way to guarantee the useful inequality

$$\beta^{t-t_1} \geqq 1 + n\,\beta^{1-t}. \tag{2.5}$$

In practice (2.3) will usually be satisfied very easily, and the distinction between t and t_1 may be ignored. The main results of this section may now be summarized in the following Lemma:

Lemma. The error in the computed inner product $s = \sum_1^n x_i\, y_i$, for $n \geqq 1$, is bounded by

$$|\varepsilon_N| \leqq \tfrac{1}{4}\beta^{1-t_1} a\, b\,(n^2 + 3n) \tag{2.6}$$

if s is computed in the usual way, by

$$|\varepsilon_W| \leqq \tfrac{1}{8}\beta^{1-t_1}(a+b)^2\,(n^2 + 16n) \tag{2.7}$$

if s is computed by Winograd's method as described above, and by

$$|\varepsilon_W| \leqq \tfrac{3}{8}\beta^{1-t_1} a\, b\,(n^2 + 17n) \tag{2.8}$$

if s is computed by Winograd's method and $a = b$.

To prove (2.6), we have (see, for example, [6])

$$\varepsilon_N = \sum_1^n x_i y_i \varepsilon_i, \tag{2.9}$$

where, writing γ for $\frac{1}{2}\beta^{1-t}$,

$$|\varepsilon_1| \leqq (1+\gamma)^n - 1 \tag{2.10}$$

and

$$|\varepsilon_i| \leqq (1+\gamma)^{n+2-i} - 1 \quad \text{for} \quad 2 \leqq i \leqq n.$$

Thus

$$|\varepsilon_N| \leqq a\, b \left[\sum_2^n \left((1+\gamma)^k - 1\right) + (1+\gamma)^n - 1\right]. \tag{2.11}$$

For $2 \leqq k \leqq n+2$ we have

$$\left(1 - \frac{k-2}{3}\gamma\right)^{-1} \leqq \frac{3}{2} \tag{2.12}$$

by assumption (2.3), and

$$(1+\gamma)^k - 1 \leqq k\gamma + \frac{k(k-1)}{2}\gamma^2\left(1 - \frac{k-2}{3}\gamma\right)^{-1}, \tag{2.13}$$

so from (2.11), (2.12) and (2.13) we have

$$|\varepsilon_N| \leqq \gamma a b \left(\frac{n^2+3n-2}{2}\right)\left(1+\frac{n\gamma}{2}\right). \tag{2.14}$$

Now (2.6) follows from (2.5) and (2.14).

To prove (2.7), we first suppose n is even, say $n=2m$, and proceed as in the proof of (2.6). By (2.3),

$$m\gamma \leqq \tfrac{1}{2}, \tag{2.15}$$

so the right side of (2.12) may be replaced by $\frac{6}{5}$ for $2 \leqq k \leqq m+2$, and in place of (2.14) we get

$$|\varepsilon_\xi| \leqq \tfrac{1}{2}\gamma a^2(m^2+3m-2)\left(1+\frac{2m\gamma}{5}\right), \tag{2.16}$$

and similarly

$$|\varepsilon_\eta| \leqq \tfrac{1}{2}\gamma b^2(m^2+3m-2)\left(1+\frac{2m\gamma}{5}\right). \tag{2.17}$$

Now

$$|\varepsilon_\xi| \leqq \tfrac{6}{5}\gamma a^2 m^2 \tag{2.18}$$

is clear if $m=1$, and follows from (2.15) and (2.16) if $m>1$. Also

$$|\xi| \leqq a^2 m + |\varepsilon_\xi|, \tag{2.19}$$

so

$$|\xi| \leqq a^2 m\left(1+\frac{6m\gamma}{5}\right), \tag{2.20}$$

and using this and the corresponding bound for η gives

$$|\xi| + |\eta| \leqq (a^2+b^2)\, m\left(1+\frac{6m\gamma}{5}\right). \tag{2.21}$$

Since $\zeta = fl(\xi+\eta)$, we have

$$|\varepsilon_\zeta| \leqq |\varepsilon_\xi| + |\varepsilon_\eta| + \gamma(|\xi| + |\eta|). \tag{2.22}$$

Collecting our results (2.16), (2.17), (2.21) and (2.22) gives

$$|\varepsilon_\zeta| \leqq \gamma(a^2+b^2)\left[m\left(1+\frac{6m\gamma}{5}\right) + \tfrac{1}{2}(m^2+3m-2)\left(1+\frac{2m\gamma}{5}\right)\right]; \tag{2.23}$$

but $m \leqq \frac{1}{2}(m^2+3m-2)$ for $m \geqq 1$, so

$$|\varepsilon_\zeta| \leqq \tfrac{1}{2}\gamma(a^2+b^2)(m^2+5m-2)\left(1+\frac{4m\gamma}{5}\right). \tag{2.24}$$

In a similar fashion we may show that

$$|\varepsilon_\theta| \leqq \tfrac{1}{2}\gamma(a+b)^2(m^2+7m-2)\left(1+\frac{7m\gamma}{6}\right). \tag{2.25}$$

Now

$$|\varepsilon_W| \leqq |\varepsilon_\theta| + |\varepsilon_\zeta| + \gamma(|\theta| + |\zeta|),$$

so

$$|\varepsilon_W| \leqq (|\varepsilon_\theta| + |\varepsilon_\zeta| + m\gamma[(a+b)^2 + a^2 + b^2])(1+\gamma). \tag{2.26}$$

The inequality $a^2+b^2 \leqq (a+b)^2$ and (2.24) to (2.26) give

$$|\varepsilon_W| \leqq \gamma(a+b)^2(m^2+8m-2)(1+3m\gamma), \tag{2.27}$$

and (2.7) now follows from (2.5) and (2.27). If $a=b$ we can derive the slightly stronger result (2.8) in the same way. Finally, if n is odd, say $n=2m+1$, the right side of (2.27) bounds the error in forming the partial inner product $\sum_{i=1}^{n-1} x_i y_i$, and it is easy to show that (2.7) holds after the addition of the term $fl(x_n y_n)$. (If $n=1$ then (2.7) is trivial.) (2.8) also holds if n is odd, by a similar argument.

(2.7) shows that Winograd's method is very bad numerically if a/b is either large or small compared to unity, just as the example above led us to expect, for then $(a+b)^2 \gg ab$. By comparison, the bound (2.6) shows that the ratio a/b is of no consequence if the normal method of forming inner products is used.

3. The Necessity for Scaling

Ignoring the simple cases $a=0$ and $b=0$, there is an integer λ such that $\beta^{-\frac{1}{2}} \leqq \beta^\lambda a/b < \beta^{\frac{1}{2}}$. If we replace x by $\beta^\lambda x$, computed without rounding errors, and then use Winograd's identity (1.1) as above, multiplying the final result by $\beta^{-\lambda}$, (2.7) shows that the error ε_W in the computed inner-product satisfies

$$|\varepsilon_W| \leqq (2+\beta^{\frac{1}{2}}+\beta^{-\frac{1}{2}})\beta^{1-t_1} a b (n^2+16n)/8. \tag{3.1}$$

This is of the same form as (2.6), and lacks the term $(a+b)^2$ of (2.7). If we want the matrix product $C=AB$, even the crude scaling $A \leftarrow \beta^\lambda A$, $B \leftarrow \beta^{-\lambda} B$, where $\beta^{-1} \leqq \frac{\beta^\lambda a}{\beta^{-\lambda} b} < \beta$, and then the application of Winograd's algorithm, will give a result $AB+E$ with

$$\max|e_{ij}| \leqq (\beta+1)^2 \beta^{-t_1} a b (n^2+16n)/8, \tag{3.2}$$

where $a=\max|a_{ij}|$ and $b=\max|b_{ij}|$. Hence Winograd's method with scaling is nearly as accurate as the usual method (without accumulation of inner products in double-precision), and the scaling takes time $O(n^2)$ compared with the total time $O(n^3)$. On the other hand, we have already shown that, without scaling, Winograd's method can lead to disastrous rounding errors. The general necessity for scaling, although fairly obvious, does not seem to have been mentioned by other authors, e.g. Fox [2] or Winograd [8], when they advocate the use of (1.1).

4. *LU* Decomposition using Winograd's Identity

Winograd [8] suggests that his matrix multiplication algorithm could be used to solve systems of linear equations by partitioning. This idea seems impractical for floating-point computation, for it is not clear how any form of pivoting could be used. As shown below, though, it is possible to use the identity (1.1) to roughly halve the number of multiplications required for the triangular decomposition of a matrix by Gaussian elimination, and we may easily use partial pivoting. Because of the pivoting and a scaling device described below, we can obtain an error bound which is satisfactory unless the growth factor g is too large. This is a rare occurrence in practice [5, 7], and in any case g affects the

error bound for ordinary Gaussian elimination with partial pivoting in the same way. A compact form of Gaussian elimination similar to Crout's or Doolittle's method is used, but inner products are evaluated using (1.1). We obtain a decomposition $PA+E=LU$, where P is a permutation matrix, L is lower triangular with unit diagonal, U is upper triangular, and the error matrix E would vanish for exact computation. L and U overwrite A in the natural way, and the only extra storage required is for four n-vectors p, d, ξ and η to be described later.

We shall first describe the algorithm by giving an ALGOL 60 procedure for the triangular decomposition of a real square matrix, and then give some comments on the algorithm:

```
procedure lu(a, p, n, pmin, g);
    value n; integer n; real pmin, g;
    integer array p; array a;
comment
Input to procedure lu:
    a: an array [1:n, 1:n] representing a real matrix A,
    n: the order of A.
Output from procedure lu:
    p:    an integer array [1:n] representing a permutation matrix P thus:
          (P)ij = 1 iff p[i] = j,
    pmin: the absolute value of the smallest pivot,
    g:    the largest absolute value of elements of U,
    a:    the input array is overwritten by the lower triangular matrix L (with
          unit diagonal implicit) and the upper triangular matrix U in the natural
          way. We have LU = PA + E, with a bound for the error matrix E
          given in Sec. 5;
begin
    integer i, i1, i2, j, k; real e, t, u, v;
    Boolean even; array x, y[1:n];
    real procedure ip(i, k, m); value i, k, m; integer i, k, m;
    comment ip returns fl_W (sum_{j=1}^{m} a_ij a_jk) as defined in the text;
    begin
        real s; integer j;
        comment assume even = true iff m is even;
        s := 0.0;
        comment the following "inner loop" could be machine-coded for greater
                efficiency. Note that only [m/2] real multiplications and 3[m/2]
                real additions are involved;
        for j := 2 step 2 until m do
        s := s + (a[i, j - 1] + a[j, k]) × (a[i, j] + a[j - 1, k]);
        ip := s - (x[i] + y[k]) + (if even then 0.0 else a[i, m] × a[m, k])
    end ip;
    even := true; g := pmin := 0.0;
    for i := 1 step 1 until n do
```

```
begin x[i] := y[i] := 0.0; p[i] := i end;
for i := 1 step 1 until n do
begin
  i1 := i + 1; i2 := i - 1; e := 0.0; k := i;
  comment find pivot in column i: the usual partial pivoting;
  for j := i step 1 until n do if e < abs(a[j, i]) then
  begin k := j; e := abs(a[j, i]) end;
  if k > i then
  begin j := p[k]; p[k] := p[i]; p[i] := j;
    t := x[k]; x[k] := x[i]; x[i] := t;
    comment we actually interchange rows i and k: whether this is the
            fastest method will depend on the machine and compiler
            used;
    for j := 1 step 1 until n do
    begin t := a[i, j]; a[i, j] := a[k, j]; a[k, j] := t end
  end;
  t := a[i, i]; comment e = abs(t);
  if i = 1 or pmin > e then pmin := e;
  comment find row i of U except for u_ii;
  for k := i1 step 1 until n do
  a[i, k] := a[i, k] - ip(i, k, i2);
  even := ¬ even; e := 0.0;
  comment the following scaling is explained in Sec. 5;
  for j := i step 1 until n do if e < abs(a[i, j]) then
  e := abs(a[i, j]);
  if g < e then g := e;
C1: comment for base β, we need only choose u = 0 if e = 0, u = β^λ otherwise,
            where λ is an integer such that β^-1 e ≦ u^2 < β e, and this is
            what we assume in Sec. 5. Here we avoid machine-dependence:
            see the note following Eq. (5.16). We assume that sqrt(0.0) = 0.0
            exactly;
  u := sqrt(e); x[i] := u;
C2: comment u is referred to as d_i in Sec. 5, while x corresponds to ξ and
            y to η. To save space, we save u in x[i] and do not need an
            array d[1:n];
  comment although u is a floating-point number there is no need for a
            tolerance here: we only need to avoid division by zero;
  if u ≠ 0.0 then
  begin for j := i step 1 until n do a[i, j] := a[i, j]/u;
    if t ≠ 0.0 then
    begin v := u/t;
      for j := i1 step 1 until n do a[j, i] := v × a[j, i]
    end
  end;
  if even then
```

```
        begin comment every second step we update ξ and η, which are needed
                      to compute inner products by Winograd's method;
          for j := i1 step 1 until n do
          begin x[j] := x[j] + a[j, i] × a[j, i2];
                y[j] := y[j] + a[i, j] × a[i2, j]
          end
        end;
        comment find u_{i+1,i+1} and column i + 1 of L;
        for k := i1 step 1 until n do
        a[k, i1] := a[k, i1] − ip(k, i1, i)
    end;
    comment unscale L and U. If we had chosen d_i = β^λ above, then scaling
            and unscaling could be done by exponent modification;
    for i := 1 step 1 until n do if x[i] ≠ 0.0 then
    begin u := x[i];
        for j := i step 1 until n do a[i, j] := u × a[i, j];
        for j := i + 1 step 1 until n do a[j, i] := a[j, i]/u
    end
end lu;
```

Procedure *lu* gives a triangular decomposition of a real matrix. For complex matrices a straightforward transliteration is possible, though it is more economical to replace $|x+iy|$ by $|x|+|y|$ (at the expense of increasing the error bound slightly).

If the triangular decomposition of A is to be used to solve a system $Ax=b$, it is desirable to try to minimize the condition number of A by row equilibration before calling procedure *lu* (see [5]).

With exact arithmetic the procedure *lu* would give a decomposition $PA=LU$. If A is nonsingular then PA uniquely determines L and U, provided L is assumed to have unit diagonal. Because of the use of Winograd's identity (1.1), the LU decomposition takes $n^3/6+O(n^2)$ real multiplications instead of the usual $n^3/3+O(n^2)$, and $n^3/2+O(n^2)$ real additions instead of $n^3/3+O(n^2)$.

Finally, if *pmin*, the absolute value of the smallest pivot, is zero or below rounding-error level, then A is singular or nearly singular, but this does not alter the relation $LU=PA+E$, with the bound for the error matrix E given in the next section.

5. Error Analysis of the LU Decomposition

We shall now derive bounds for the elements e_{ij} of the error matrix E defined by $LU=PA+E$. As well as the assumptions of Section 2, we assume that

$$fl(x \# y) = (x \# y)(1+\varepsilon_1) = (x \# y)/(1+\varepsilon_2),$$

where $\#$ is any of the arithmetic operations $+$, $-$, $\times$, $/$, and $|\varepsilon_i| \leqq \frac{1}{2}\beta^{1-t}$. The additional assumptions simplify the analysis, but are not really necessary: without them the term $15n$ in (5.15) would have to be increased slightly.

Let

$$u'_{ij} = \begin{cases} 0 & \text{if } d_i = 0, \\ u_{ij}/d_i & \text{otherwise,} \end{cases} \tag{5.1}$$

$$m'_{ij} = m_{ij} d_j,$$

and

$$a'_{ij} = (PA)_{ij},$$

so u'_{ij}, m'_{ij} and a'_{ij} are just the elements of the matrices U', L' and A' satisfying $L'U' = A' + E$. Note that

$$m'_{ik} u'_{kj} = m_{ik} u_{kj} \tag{5.2}$$

holds even if d_k vanishes, for then both sides of (5.2) vanish.

The "growth factor" $g = \max |u_{ij}|$ will be important in the analysis below (this is the g produced by procedure *lu*). Strictly, the "growth factor" is $g/\max |a_{ij}|$, for we do not need to assume that $\max |a_{ij}| = 1$. The analysis following is similar to that in [5].

By definition,

$$a'_{ij} + e_{ij} = \sum_{k=1}^{\min(i,j)} m'_{ik} u'_{kj}, \tag{5.3}$$

but for $i \leqq j$ we have

$$u_{ij} = fl\left(a'_{ij} - fl_W \sum_{k=1}^{i-1} m'_{ik} u'_{kj}\right). \tag{5.4}$$

Now $|m_{ik}| \leqq 1$ by the pivoting, and $|d_k| < \beta^{\frac{1}{2}} g^{\frac{1}{2}}$ if scaling is done as described in comment *C1* (see comments *C1* and *C2* in the ALGOL procedure above), so

$$|m'_{ik}| < \beta^{\frac{1}{2}} g^{\frac{1}{2}}. \tag{5.5}$$

Also, if $d_k \neq 0$ then

$$\begin{aligned} |u'_{kj}| &= |u_{kj}/d_k| \\ &= |u_{kj}/d_k^2|^{\frac{1}{2}} |u_{kj}|^{\frac{1}{2}}, \end{aligned}$$

so

$$|u'_{kj}| \leqq \beta^{\frac{1}{2}} g^{\frac{1}{2}} \tag{5.6}$$

by the scaling, and (5.6) also holds if $d_k = 0$, for then $u'_{kj} = u_{kj} = 0$.

Hence, by (2.8) with $a = b = \beta^{\frac{1}{2}} g^{\frac{1}{2}}$ and n replaced by $i-1$, we have

$$\left| fl_W \sum_{k=1}^{i-1} m'_{ik} u'_{kj} - \sum_{k=1}^{i-1} m'_{ik} u'_{kj} \right| \leqq 3\beta^{2-t_1} g (i^2 + 15i - 16)/8 \quad \text{for} \quad i \geqq 2, \tag{5.7}$$

but from (5.4)

$$u_{ij}(1+\varepsilon) = a'_{ij} - fl_W \sum_{k=1}^{i-1} m'_{ik} u'_{kj}, \tag{5.8}$$

where $|\varepsilon| \leqq \frac{1}{2}\beta^{1-t}$. (With the weaker assumption about addition the argument becomes more complicated here, as (5.8) may not hold.) Since $|u_{ij}| \leqq g \leqq 3\beta g/4$, we have

$$\left| a'_{ij} - u_{ij} - \sum_{k=1}^{i-1} m'_{ik} u'_{kj} \right| \leqq 3\beta^{2-t_1} g (i^2 + 15i - 15)/8 \tag{5.9}$$

for $2 \leqq i \leqq j \leqq n$. Now $u_{ij} = m_{ii} u_{ij}$, so from (5.2), (5.3) and (5.9),

$$|e_{ij}| \leqq 3\beta^{2-t_1} g(i^2 + 15i - 15)/8 \tag{5.10}$$

for $2 \leqq i \leqq j \leqq n$. If $i = 1$ then $u_{ij} = a'_{ij}$, so $e_{ij} = 0$ and (5.10) holds a fortiori.

In a similar fashion we can bound e_{ij} for $i > j$. We have

$$m_{ij} = \begin{cases} 0 & \text{if} \quad u_{jj} = 0 \\ fl\left(\dfrac{a_{ij} - fl_W \sum_{k=1}^{j-1} m'_{ik} u'_{kj}}{u_{jj}} \right) & \text{otherwise} \end{cases} \tag{5.11}$$

for $1 \leqq j < i \leqq n$, so

$$m_{ij} u_{jj} (1 + \varepsilon) = a_{ij} - fl_W \sum_{k=1}^{j-1} m'_{ik} u'_{kj}, \tag{5.12}$$

where $|\varepsilon| \leqq \beta^{1-t_1}$. (If $u_{jj} = 0$ then both sides of (5.12) vanish.) Using (2.8) as before, this gives

$$\left| a_{ij} - m_{ij} u_{jj} - \sum_{k=1}^{j-1} m'_{ik} u'_{kj} \right| \leqq 3\beta^{2-t_1} g(j^2 + 15j - 14)/8 \tag{5.13}$$

for $1 \leqq j < i \leqq n$. (5.2), (5.3) and (5.13) give

$$|e_{ij}| \leqq 3\beta^{2-t_1} g(j^2 + 15j - 14)/8 \tag{5.14}$$

for $1 \leqq j < i \leqq n$.

From (5.10) and (5.14) we have the uniform bound

$$|e_{ij}| \leqq 3\beta^{2-t_1} g(n^2 + 15n)/8, \tag{5.15}$$

$1 \leqq i, j \leqq n$, for the elements of the error matrix E. Comparing (5.15) with the bound

$$|e_{ij}| \leqq \beta^{1-t_1} g(n^2 + n)/4 \tag{5.16}$$

for the usual Crout decomposition, we see that our bound is worse by a factor $3\beta/2 + O(1/n)$ as $n \to \infty$. The β in this factor may be removed if, after comment *C1*, we choose $d_i = fl(e^{\frac{1}{2}})$ (as is done in the ALGOL procedure above: see comment *C1*). Multiplication and division by d_i will introduce further rounding errors now that d_i is not necessarily an integral power of β, but these can be accounted for by increasing the term $15n$ in (5.15) to $21n$, i.e. with this modification we have a bound

$$|e_{ij}| \leqq \tfrac{3}{8} \beta^{1-t_1} g(n^2 + 21n). \tag{5.17}$$

The factor $3\beta/2$ is not of great significance in practice, for both (5.15) and (5.16) are very pessimistic bounds because of the statistical effect of cancellation of errors.

The reason for the scaling steps should now be apparent. Without them we could guarantee only that $|m_{ij}| \leqq 1$ and $|u_{ij}| \leqq g$, so a factor of $(1+g)^2$ would appear in the error bound. In fact, using (2.7) with $a = 1$ and $b = g$, and proceeding

as above, gives the bound

$$|e_{ij}| \leqq \tfrac{1}{8}\beta^{1-t_1}(1+g)^2(n^2+14n), \tag{5.18}$$

so for any hope of success we would have to scale A so that $\max|a_{ij}| \simeq 1$, for otherwise g could be arbitrarily large or small compared to unity.

6. Cholesky Decomposition using Winograd's Identity

It is clear from the above discussion of the LU decomposition that, if A is real positive definite and symmetric, then we can obtain a decomposition $A = LL^T$ with $n^3/12 + O(n^2)$ multiplications and $n^3/4 + O(n^2)$ additions, instead of the usual $n^3/6 + O(n^2)$ of each, by using (1.1) to evaluate the inner-products involved. We need only keep numbers $\xi_1, \ldots, \xi_n$ corresponding to the rows of L, and update them when necessary. There is no need for pivoting, and as $a_{ii} = m_{i1}^2 + \cdots + m_{ii}^2$, all the elements m_{ij} of L are bounded. The error analysis is similar to that in Section 5 above and in [5], so we merely state that if $LL^T = A + E$, $|a_{ij}| \leqq 1$, and the square roots are computed with a relative error bounded by β^{1-t}, then

$$|e_{ij}| \leqq \tfrac{3}{8}\beta^{1-t_1}(n^2+15n). \tag{6.1}$$

7. Concluding Remarks

We have shown above that Winograd's identity (1.1) is useful for the floating point computation of inner products in certain circumstances, provided care is taken with scaling. In particular, matrix multiplication, the LU decomposition of a square matrix, and the Cholesky decomposition of a positive definite symmetric matrix can be performed with about half the usual number of multiplications, and with nearly the usual accuracy. There is little point in accumulating the inner products on the right side of (1.1) in double-precision unless the additions $x_{2i-1} + y_{2i}$ and $x_{2i} + y_{2i-1}$ are also done in double-precision.

Klyuyev and Kokovkin-Shcherbak [3] have shown that, in general, $n^3/3$ multiplications are required to solve the n by n system $Ax = b$, if we are restricted to operations on rows and columns as a whole. Since this system can easily be solved in $O(n^2)$ operations once the LU decomposition of A is known, their theorem now appears too restrictive to have much practical significance.

Strassen [4] has shown that matrix multiplication can be performed in $O(n^{2.8\cdots})$ arithmetic operations ($2.8\ldots = \log_2 7$). For sufficiently large n, his algorithm must be faster than any other known method. On the other hand, practical tests indicate that Winograd's algorithm, even with the necessary scaling, is faster than Strassen's for $n \lesssim 250$, though the precise changeover point will depend on the machine and compiler used (see [1]), and there is no doubt that Winograd's method is easier to program than Strassen's. Finally, while a satisfactory error bound can be given for floating-point matrix multiplication by Strassen's method, this is not so for his method of matrix inversion or solution of systems of equations, for with his method no pivoting is possible.

Acknowledgement. I wish to thank Drs. G. Forsythe, J. Herriot and J. Wilkinson for their helpful advice, and CSIRO (Australia) for its generous financial support.

References

1. Brent, R. P.: Algorithms for matrix multiplication. Tech. Report CS 157 (March 1970), Computer Sci. Dept., Stanford Uni.
2. Fox, B. L.: Accelerating *LP* algorithms. CACM 12, 7 (July 1969), 384—385.
3. Klyuyev, V. V., Kokovkin-Shcherbak, N. I.: On the minimization of the number of arithmetic operations for the solution of linear algebraic systems of equations. Translation by G. I. Tee: Tech. Report CS 24 (June 1965), Computer Sci. Dept., Stanford Uni.
4. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**, 354—356 (1969).
5. Wilkinson, J. H.: Error analysis of direct methods of matrix inversion. JACM **8**, 281—330 (1961).
6. — Rounding errors in algebraic processes. London: H.M.S.O.; New Jersey: Prentice-Hall 1963.
7. — The algebraic eigenvalue problem. Oxford: Clarendon Press 1965.
8. Winograd, S.: A new algorithm for inner product. IEEE Trans. C-17 (1968), 693—694.

R. P. Brent
Computer Science Department
Stanford University
Stanford, CA. 94305, U.S.A.