Indices in tridiagonal matrices

Mark van der Loo

June 2015

Introduction. Distance matrices are (usually) symmetric, and have 0 on the diagonal. Hence, it makes sense to store only the upper or lower triangular matrix in a simple array. For example, for a 4×4 matrix one only needs to store six elements, indexed $k = 0, 1, \ldots, 5$. The problem is to identify, in general the index k with matrix row, and column indices i and j so the matrix can be reconstructed, as illustrated below (dashes indicat the location of the digonal).

$$\begin{pmatrix} - & & & \\ 0 & - & & \\ 1 & 3 & - & \\ 2 & 4 & 5 & - \end{pmatrix} \qquad \begin{array}{c|cc} \text{k} & \text{i} & \text{j} \\ \hline 0 & 1 & 0 \\ 1 & 2 & 0 \\ 2 & 3 & 0 \\ 3 & 2 & 1 \\ 4 & 3 & 1 \\ 5 & 3 & 2 \end{array}$$

Note that we define all indices using base 0. We remind the reader of the result

$$m + (m-1) + \cdots + 1 = \frac{m(m+1)}{2}. \tag{1}$$

From row- and column indices to array index. Denote with $N(j)$ the number of elements that can be stored in columns $0, 1, \ldots, j$ and let n be the order of the matrix. Observe that the first column contains $n-1$ elements, so

$$\begin{aligned} N(j) &= (n-1) + (n-2) + \cdots + (n-1-j) \\ &= \frac{n(n-1)}{2} - \frac{(n-2-j)(n-1-j)}{2} \\ &= -\frac{1}{2}j^2 + \frac{2n-3}{2}j + n - 1, \end{aligned} \tag{2}$$

for $0 \leq j \leq n-1$ and n the number of columns in the distance matrix. To compute the value of k as a function of i and j, observe that $N(j) - n + 1$ (n being the number of rows) is the ofset in the jth column from where i is counted. Since k is counted base-0, we add $i - 1$ to get

$$k(i,j) = N(j) - n + 1 + i - 1 = -\frac{1}{2}j^2 + \frac{2n-3}{2}j + i - 1. \tag{3}$$

From array index to row- and column index. To go the other way around observe first that for the lower triangular matrix we have $0 < i \leq j$. So given a k, we first find the largest j for which $N(j) > k$, or equivalently $N(j) \geq k + 1$. We can compute this by solving $N(j) = k + 1$ for j and rounding up. We get

$$j_{\pm} = \frac{2n-3}{2} \mp \sqrt{\left(\frac{2n-3}{2}\right)^2 + 2(n-k-2)}.$$

Since at $k = n-2$ we know that $j = 0$, the solution j_+ is the one we want. This yields, after simplifying the discriminant a little

$$j = \left\lceil \frac{2n-3}{2} - \sqrt{\left(n - \tfrac{1}{2}\right)^2 - 2(k+1)} \right\rceil. \tag{4}$$

Once j is computed, i is retrieved simply by solving for it in Eq. (3). After simplifying it a bit we get

$$i = k + \frac{1}{2}j(j - 2n + 3) + 1. \tag{5}$$

One can trivially show that for each $j \in \mathbb{Z}$, it must be that $i \in \mathbb{Z}$ by working out the cases where j is even ($j = 2m$ for some $m \in \mathbb{Z}$) and odd ($j = 2m + 1$).

Algorithm for serial execution. If we run serially (single-core) through the distance matrix, the following (pseudo) `C` code correctly updates i and j.

```
int colmax = n-1
  , i = 0
  , j = 0;

for ( int k = 0; k < n*(n-1); k++ ){
  i++;
  // compute d[k] = f(i,j)
  if ( i == colmax ){
    j++;
    i = j;
  }
}
```

The idea is to simply update the offset for rows each time the column counter needs to be updated.

Algorithm for parallel execution. We can use Eqs. (4) and (5) to compute i and j but computing square roots is expensive and we're not sure whether roundoff errors may occur causing the ceiling function to round off to the wrong integer (causing possible segfault). Especially since the term $(n - \frac{1}{2})^2$ can get rather large and is therefore a possible source for numerical instability. We therefore limit it's use to a single calculation per thread. The following code works inside an `openmp` parallel region.

```
int core_id = omp_get_thread_num()            // 0..ncores
  , n_cores = omp_get_num_threads()           // number of threads
  , colmax = n-1                              // n = #rows (=columns)
  , nn = n*(n-1)/2                            // reused constant
  , p = nn / n_cores;                         // reused constant

int k1 = core_id * p                          // start k
  , k2 = core_id < n_cores -1 ? k1 + p : nn   // end k
  , j = get_j(k1,n)                           // Eq (4)
  , i = get_i(k1,j,n) - 1                     // Eq (5)

for (int k = k1; k<k2; k++){
    i++;
    // d[k] = f(i,j)
    if ( i == colmax ){
      j++
      i = j
    }
}
```

The functions `get_j` and `get_i` implement Eqs. (4) and (5). Numerical stability can be enhanced by checking whether `i` and `j` correctly reproduce `k` using Eq. (3).