TECHNICAL MEMORANDUM

23

The vectorisation of the vertical interpolation

D. Robertson

Research Department

April 1981

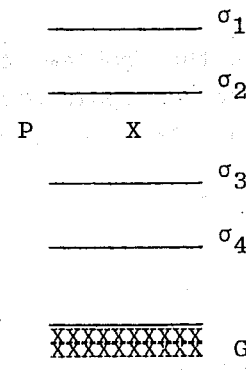
European Centre for Medium-Range Weather Forecasts
Europäisches Zentrum für mittelfristige Wettervorhersage
Centre européen pour les prévisions météorologiques à moyen

1. Introduction

This note is a brief explanation of the method used to vectorize the vertical interpolation in the post processing of the ECMWF model output. It is hoped that it will be of some use to anyone working on that section of the code and also it provides an example of the use of the "gather" operation to vectorize and hence materially speed up a calculation.

2. Original formulation

The model outputs data on σ levels, (NLEV = 15 of them) and output is desired at some pressure level P, which, in general, does not coincide with a σ level. At some particular point in space one has a situation such as



(here NLEV = 4)

A sophisticated formula is used to calculate the value of a given quantity at the pressure level, namely

$$\text{RESULT} = \left\{ \frac{(X-k(I+3))}{(k(I+5)-k(I+3))} \cdot \Big[\{X-k(I+3)\}\cdot A(I+1) - \{X-k(I+5)\}\cdot A(I) \Big] \right.$$

$$\left. \frac{(X-k(I+4))}{(k(I+4)-k(I+2))} \cdot \Big[\{X-k(I+2)\}\cdot A(I) - \{X-k(I+4)\}\cdot A(I-1) \Big] \right\}$$

$$\Big/ (K(I+4)-k(I+3)) \qquad \ldots\ldots\ldots 1)$$

where

$$A(I) \equiv \frac{(X-k(I+2))\cdot S(I+2) - (X-k(I+5))\cdot S(I+1)}{K(I+5) - K(I+2)} \cdot$$

Here RESULT is the answer at a pressure level which has a σ value of X. The quantities K(1), K(2), K(3) ..... are real numbers which are the values of the model σ levels, and S(1), S(2), S(3) ..... are quantities which are calculated previously from the data values at the σ levels. The origin of this formula is discussed in Appendix 1. (It is not necessary to understand the theory to follow the rest of this note.) However, let us note that an evaluation at a point involves 27 additions/subtractions, 14 multiplications, and 7 divisions (in the domain of real numbers), as well as indexing calculations. In addition, when the evaluation routine is called, with X, S(1)→S(N)

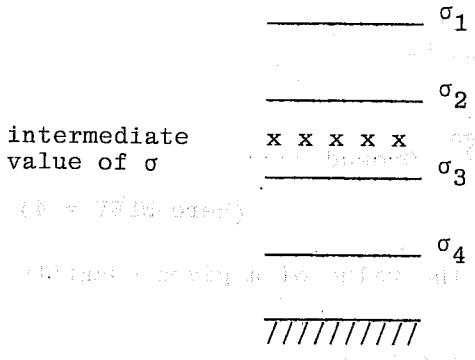
and K(1)→K(N) as inputs, it must locate X relative to the σ levels, (determine the relevant value of I in the above formula). In the original formulation this calculation was done by a routine EØ2BBF from the NAG library. EØ2BBF used a binary search to locate X.

3. Vectorized formulation

The output σ level data is presented on circles of constant latitude, (LINE = 194 points per circle). The above interpolation is embedded inside a loop around the circle e.g.

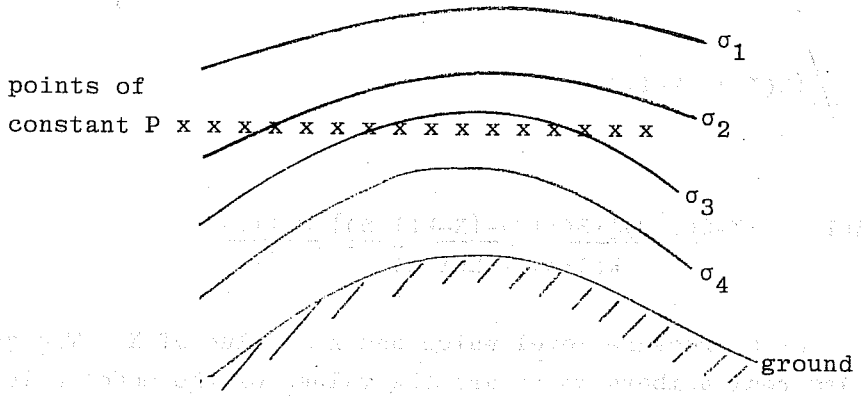
```
      DO  xxx  J = 1,  LINE
          CALL EØ2BBF(.....)
xxx   ....
```

It is natural to try and reformulate the problem to interpolate all points on a circle at the same time. If the task was to interpolate to an intermediate σ value, this would be trivial. All the points on a latitude circle would lie between the same σ levels, e.g.



and the J loop could be taken inside immediately. The co-efficients, S, have two indices, I for level and J for position on the circle and the J loop vectorizes immediately.

However, when one is interpolating to a pressure level, things are more complex. The relation between P and σ depends on horizontal position and one has a situation like



Here the location of the P level relative to the σ levels changes as one changes position, and different values of I are found by the binary search at different places. Hence the data points used in the formula are not uniformly spaced in memory and the vector instructions of the CRAY-1 cannot be used immediately.

Preliminary work must be done to organise the desired data into convenient, regularly spaced locations before doing the interpolation. Such an operation is called a "gather" and corresponds to a loop of the form

```
DO  xxx I = 1 , N
  A(I) = B(INDEX(I))
```

Here arbitrarily located data, selected from B by the vector INDEX, are collected into consecutive locations in A. †

The gather operation is not a hardware primitive on the CRAY-1, but an efficient CRAY VECTOR FUNCTION, GATHR, is available in ECLIB to perform this operation. The list of pointers, INDEX, holds the integers which locate the pressure level relative to the σ levels, as a function of J (the index which moves around a latitude circle). The original binary search algorithm to determine INDEX is a difficult one to adapt to the CRAY-1 hardware. It turns out that a simple algorithm can be used which exploits the architecture of the CRAY-1 and is substantially faster than a binary search for the problem in question.

Examining the formula 1), we see that 6 σ level positions and 4 data values, (K(I+1)...K(I+6), S(I),...S(I+3)) enter into the final result. Thus 1Ø invocations of the gather operation which are needed (with trivially different values of INDEX for each). This looks like a substantial amount of overhead in moving data around but the vector instructions are so much faster that the new formulation is substantially faster than the old one. After performing the gather operations the calculation is done by a loop

```
DO xxx J = 1 , LINE
  ....
```

where the calculation is done by using an analogue of 1), where each quantity appearing there has a subscript J added to it and the I subscripts in K and S have disappeared because of the gathering done. Overall the new method was about four times as fast as the original one.

4. Some general remarks

First of all, it should be noted that this method does require some extra storage space, to hold the gathered circles of data. In the problem in question, this was not a serious difficulty but in other cases it might become so. It could be desirable to explicitly subdivide the original loop into loops 64 long, thus obtaining maximum hardware speed from the CRAY-1, while shortening the temporary vectors.

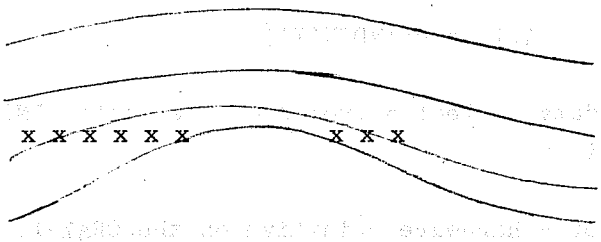
---

† The "inverse" operation, distributing contiguous elements to irregular locations,

```
DO  xxx  I = 1 , N
  A(INDEX(I)) = B(I)
```

is called a "scatter" operation. There are gather and scatter subroutines available in ECLIB.

As presented, work was done in organizing the data but no extra, possibly irrelevant, floating point calculations are done. In fact the situation is more complicated. In a mountainous region, the desired pressure level may not really exist but would be below the earth's surface.



The original code had an IF test.

```
IF (above ground) CALL EØ2BBF
IF (below ground) CALL extrapolation routine
```

Since the test depends on position (J), it would prevent vectorization of the loop. In the vector formulation all points were calculated by the vector analogue of EØ2BBF. Then, in a subsequent scalar loop, extrapolation was performed to overwrite those points below ground.

```
      DO  xxx  J = 1 , LINE
xxx   IF (point J is underground)
         use extrapolation formula to recalculate
         the value at point J
```

This raises two questions of interest. The original formula is being used in regions where it was not applied before. Although the output of these calculations is later over written and is never used, one must be sure that a spurious arithmetic error does not occur, (overflow, divide by Ø etc.) which would abort the job. In this case, one is merely making a different extrapolation and no problem arises. Secondly, it is now clear that the vectorized method is performing more calculations than the scalar method. As a result the overall speed up depends on the pressure level desired. At pressure levels well above the ground, little or no extrapolation is done and the scalar loop does almost no calculation. Nearer the ground, more and more extrapolation is done. The overall speed increase was found to vary from a factor of 5 at the top of the atmosphere to 20% near the surface of the earth. The performance near the ground could probably be improved by a merging technique, where both the interpolation and the extrapolation are done by vector calculations and the conditional merge functions are used to select the desired answer. Unfortunately the method of extrapolation depends on the physical quantity in question, so this would require a lot of new code for probably not all that much improvement. On the other hand, for pressure levels near the top of the atmosphere, the scalar extrapolation loop could be simply by-passed as the test will never be satisfied and no calculations need be done.

A sample code showing both methods of calculation follows:

Sample Code

In this code, PS(*) is a vector holding values of surface pressure around a latitude circle. R(LINE,NLEV) is an array which holds the B spline coefficients. R(J,*) describes the spline which fits the vertical column of data at horizontal point J. ZLNS is the σ value of the desired pressure level, at point J. XXSIG holds the σ levels, XXKNOT the knot positions.

The original code was something like:

```
C     INTERPOLATE   ONE LATITUDE CIRCLE
         DO 1ØØ    J = 1, LINE

C     EXTRACT COEFFICIENTS FOR THE COLUMN
         DO 2Ø     K = 1, NLEV

      2Ø   WK(K) = R(J,K)

C     FIND SIGMA VALUE OF PRESSURE LEVEL
         ZLNS = ALOG(P) - ALOG(PS(J))

C     INTERPOLATE OR EXTRAPOLATE
         IF(ZLNS.LE.XXSIG(NLEV))
      $    CALL EØ2BBF( NLEV+4,XXKNOT,WK,ZLNS,RESULT,IFAIL)
         IF(ZLNS.GT.XXSIG(NLEV))
      $    CALL EXTRAP(ZLNS,.............)
         ANSWER(J) = RESULT

1ØØ      CONTINUE

C  ANSWER(*) NOW HOLDS A CIRCLE OF VALUES
C  AT PRESSURE LEVEL P.
```

In the vectorized version, ZLNS is conceptually best thought of as a vector of coordinates. The positions of these pressure levels are found by INDEX.

```
CDIR$  VFUNCTION  INDEX
       IPXKN = LOC (XXKNOT)

C     THE FOLLOWING LOOP VECTORIZES
         DO 2ØØ    J = 1, LINE
            ZLNS(J) = ALOG(P) - ALOG(PS(J))
            IDX(J)  = INDEX(ZLNS(J),IPXKN,NLEV+4) - 2

2ØØ      CONTINUE
```

In fact, the CRAY FORTRAN compiler is able to automatically convert loop index dependent scalars into temporary vectors, so it is not necessary or desirable to convert ZLNS into a vector for vectorization purposes. However, the value is needed later so it is saved.

```
C     GATHER KNOT POSITIONS INTO WORK SPACE
C     XK(*,*).   (A VECTOR LOOP)
CDIR$     VFUNCTION   GATHR
          DO   3ØØ   KNOT = 1 , 6
          DO   299   J = 1 , LINE
               XK(J,KNOT) = GATHR(IPXKN+IDX(J))
               IDX(J)     = IDX(J)+1
299       CONTINUE
3ØØ       CONTINUE

C     NOW GATHER THE SPLINE COEFFICIENTS INTO WORK SPACE PTC(*,*).  NOTE THAT
C     WE ARE GATHERING ON THE SECOND INDEX OF THE ARRAY AND HENCE MUST
C     TRANSFORM THE POINTER VECTOR APPROPRIATELY.  JVEC(*) HOLDS
C     1,2,3,4,..... → LINE
          DO   35Ø   J = 1 , LINE
35Ø       IDX(J) = (IDX(J)-8)*LINE + JVEC(J)
          IPR = LOC(R)
          DO   38Ø   K = 1 , 4
          DO   379   J = 1 , LINE
379       PTC(J,K) = GATHR(IPR+IDX(J))
          IPR = IPR + LINE
38Ø       CONTINUE
```

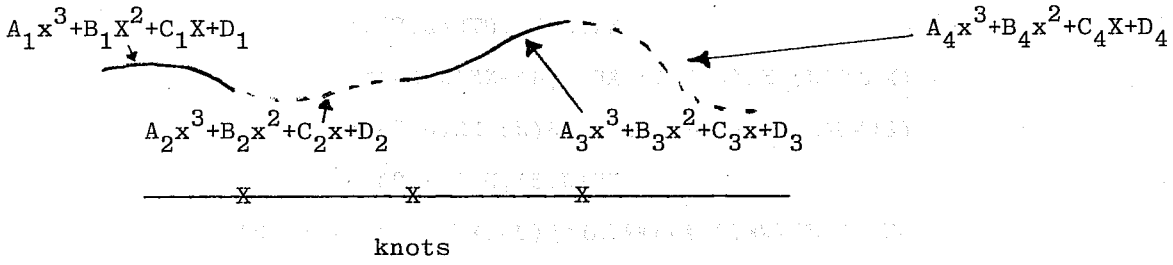
Now one is all set to interpolate. A is a statement function,

```
A(X,X2,C2,X5,C1)=((X-X2)*C2-(X-X5)*C1)/(X5-X2)

C     INTERPOLATION LOOP (VECTORIZES)
          DO   7ØØ  J = 1 , LINE
```

```
      ANSWER(J) = (XLNS(J)-XK(J,3))/(XK(J,5)-XK(J,3))*(
     1              (XLNS(J)-XK(J,3))*A(XLNS(J),XK(J,3),PTC(J,4),
     2                                 XK(J,6),PTC(J,3))-
     3             (XLNS(J)-XK(J,5))*A(XLNS(J),XK(J,2),PTC(J,3),
     4                                 XK(J,5),PTC(J,2)))
     5            - (XLNS(J)-XK(J,4))/(XK(J,4)-XK(J,2))*(
     6              (XLNS(J)-XK(J,2))*A(XLNS(J),XK(J,2),PTC(J,3),
     7                                 XK(J,5),PTC(J,2))-
     8              (XLNS(J)-XK(J,4))*A(XLNS(J),XK(J,1),PTC(J,2),
     9                                 XK(J,4),PTC(J,1)))
     1             )/(XK(J,4)-XK(J,3))

 7ØØ     CONTINUE

      C  OVERWRITE POINTS WHERE EXTRAPOLATION IS NEEDED

         DO   8ØØ  J = 1, LINE

              IF (XLNS(J).GT.XXSIG(NLEV))

      $         CALL EXTRAP(XLNS(J)·····)
```

The interpolation scheme APPENDIX 1

The interpolation method used was cubic spline interpolation. A spline is defined as a curve which is piecewise cubic. The points at which the cubic changes form are called knots.



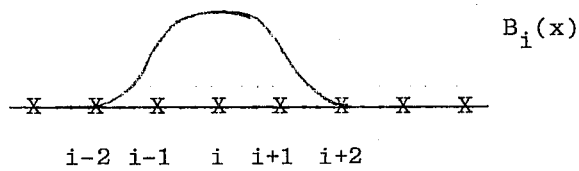
Usually at a knot, the curve, and its first and second derivative are continuous, while the third derivative shows a finite jump discontinuity, e.g. y''' changes from A_1 to A_2 at the left most knot above. By allowing adjacent knots to coalesce a lesser degree of continuity is possible.

If one has a spline defined over M intervals ($xo \leq x1 \leq x2 \ldots \leq xM$), it has M+3 degrees of freedom, (since there are 4M co-efficients of the M cubics, and 3 continuity conditions at each of the M-1 interior points $X_1 \ldots X_{M-1}$).

It turns out that one can define special splines called B-splines once a mesh (≡ set of knots) has been chosen, which form a basis. That is,there are M+3 B-splines and a general spline S(x) has a unique expansion in B-splines,

$$S(x) = \sum_{\ell=1}^{M+3} S_\ell B_\ell(x)$$

The B splines are chosen to be non-zero over as small an interval as possible and an interior B-spline looks like this.



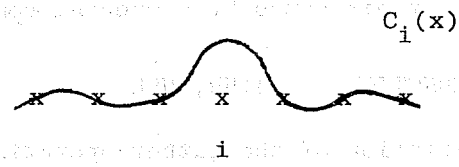
Consequently at any point x, when evaluating the spline S, at most 4 B-splines contribute to the sum.
The NAG routines EØ2BAF, EØ2BBF use this form of spline representation.
EØ2BBF evaluates a spline at a point, X, given vectors S, K (K is REAL), which are the co-efficients and the knot positions respectively. One evaluates via

```
CALL  EØ2BBF(M+7,K,S ,X,RESULT, IFAIL)
```

As applied in the post processing, for a given horizontal point, an interpolating spline is constructed to fit the data at the NLEV σ levels. (Ignoring the slight complication of full and half levels), NLEV degrees of freedom imply NLEV-3 intervals and the choice used, (the recommended one) is to make the spline knots coincide with the σ levels, omitting the 2nd and the 2nd last σ levels, e.g.

```
. . . . . . . . . . . . . . . σ levels
x   x x x x x x x x x x x   x knots
```

To minimise later calculation, in an initial stage cardinal splines are constructed, (in the sense that the co-efficients of their B-spline expansionsare found). A cardinal spline is defined to take the value 1 at a specific knot and $\emptyset$ at all other knots.



Given the representation of the Cs

$$C_i(x) = \Sigma_{\ell} XXC_{i,\ell} \; B_{\ell}(x) \quad ,$$

a spline which assumes values F_R at the level σ_R can be written as

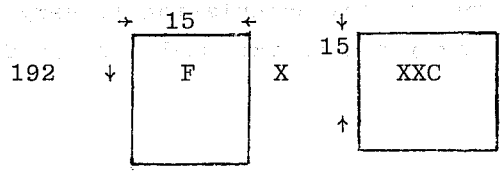
$$S(x) = \Sigma_k F_k C_k(x) = \Sigma_{k,\ell} F_k \cdot XXC_{k,\ell} \cdot B_{\ell}(x)$$

$$= \Sigma_{\ell} S_{\ell} \cdot B_{\ell}(x)$$

In fact one works with a complete latitude circle of data at a time, so that one can add a subscript j to the above to indicate the particular point in question

$$S_j(x) = \Sigma_{k,\ell} F_{j,k} \cdot XXC_{k,\ell} \; B_{\ell}(x)$$

Here $F_{j,*}$ represents the vertical column of data at point j, and the k sum can be thought of as a matrix multiplication.



It is best performed by a suitable invocation of the (CAL) routine, MXMA,

```
CALL  MXMA(F,1,LINE, XXC,1,NLMAX, S ,I,LINE,LINE, NLEV, NLEV)
```

(with LINE = 194 the actual first dimension of F and S and XXC is actually dimensioned by NLMAX). Once the B spline co-efficients are found, the calculation point by point proceeds as outlined previously.

The CAL vector functions GATHR and INDEX

In CFT, (CRAY-1 FORTRAN) there is the possibility of employing special user supplied functions, (which must be written in CAL, the assembler language), which are called vector functions. The compiler can vectorise loops containing references to these functions. It must be made aware of their names by a special compiler directive, e.g.

```
CDIR$     VFUNCTION     GATHR,INDEX
```

GATHR provides an efficient implementation of the gather operation and allows loops involving indirect addressing to be vectorised. INDEX provides a fast way of locating elements in an ordered table. Both GATHR and INDEX are described in the bulletin on ECLIB (ECMWF bulletin B6.1/3)

The algorithm used in INDEX is worth mentioning. To locate a single point, X, in an ordered table Y(1)<=Y(2)...<=Y(N) one can do a single vector subtraction,

$$T(I) = X - Y(I) \quad , \quad I = 1, N \cdot$$

The elements of T are positive as long as Y exceeds elements of Y, then become negative. Instructions are available to extract the sign information from vector X, producing a bit vector consisting of consecutive 1 bits, with the transition from 1 to Ø marking the point which locates X in the table. By counting the number of 1 bits, one finds the position of X in the table. For a table of length N this requires an amount of work proportional to N to look up an item, while a binary search only requires an amount of work proportional to log N. Since the vector instructions are so much faster than scalar instructions, the method which uses more operations is faster for reasonable table lengths (up to a few hundred). When the table is at most 64 long it may be held in a CRAY vector register and the calculation is very efficient indeed. It is at least 1Ø times as fast as a binary search for tables up to 64 long.