REGNECENTRALEN
DANSK INSTITUT FOR MATEMATIKMASKINER
UNOFFICIAL SUBROUTINE

SUBROUTINE

Arthur Evans No. 1
6 May 1960

Interpolation, DASK-tal, in a table with unevenly spaced arguments

Entrance	Entry Conditions	Exit Conditions
Normal entry 0 A8 16 (B) A 35 (n) A 00 (p) A 00 (alarm return) (normal return)	C(AR) = w Table of y[i] = f(x[i]), as follows: B x[0] B+2 y[0] B+4 x[1] B+6 y[1] ... ... B+4(n-1) x[n-1] B+4(n-1)+2 y[n-1] The x[i] must be increasing. B must be even. n must be $\geq$ p.	Normal exit f(w) in AR and MR
Alarm reentry 58 A8 10	IRD must still have the same value it had on the original entry.	Alarm exit Will occur if w is out of range of the table.

Length: 112+4p
Beginning address: Even
Internal parameters: None
Program parameters: See above.

Sub-subroutines: None
Working storage: 104A8 to (111+4p)A8
Permanent constants: (2039), (2041)
Alarm stop: 103A830, in location 103A8.

1. General

A function y = f(x) is given by a set of n pairs (x[i], y[i]), where y[i] = f(x[i]). An argument w is given, and f(w) is to be calculated by p-point interpolation in the table.

2. Storage of the table

The table is to be stored as full-length DASK-tal words with arguments and function values alternating, as follows:

B	x[0]
B+2	y[0]
B+4	x[1]
B+6	y[1]
...	...
B+4(n-1)	x[n-1]
B+4(n-1)+2	y[n-1]

That is, x[i] is to be stored in location B+4i and y[i] is to be in location B+4i+2 . B is the first location of the table (and, of course, it must be even), and there are n pairs of points in the table.

The x[i] must be increasing; i.e., it must be the case that x[i+1] > x[i] for each i. Further, each argument must be less than 1/2 in magnitude, and the first and last arguments must differ by no more than 1/2.

3. Calling the routine

The subroutine is to be called by putting w as a DASK-tal into AR, and giving the following sequence of commands:

```
   u:   0  A8 16          Index jump to the subroutine.
 u+1:  (B) A  35          Table base to IRB.
 u+2:  (n) A  00          Number of pairs of points in the table.
 u+3:  (p) A  00          Order of interpolation desired.
 u+4:  (alarm return)     Used if w is outside the range.
 u+5:  (normal return)
```

The effect of the subroutine will be to put the desired value into both AR and MR. The three index registers will be left unaltered.

The instruction in location u+1 must be exactly as shown and may not be a 37 opcode, since its address is used in one place and it is executed by a 37 opcode in another.

The action of the subroutine is not defined if p > n, or if the table is not increasing in x.

4. Argument outside of range

If w is outside the range of the table (i.e., if w < x[0] or if w > x[n-1]), then the routine will go to the alarm return in location u+4. If this location contains a transfer to location 58 of the subroutine, the desired p-point extrapolation will be performed. If the alarm return occurs, IRB will have been altered. Its former value may be restored by executing the instruction in location 100 of the subroutine (100A837). If the alarm return leads to a routine which, eventually, leads back to 58A8, this routine must leave IRD at the same value it had on the original entrance to the subroutine, although IRB and IRC may be changed. Note, however, that on the final (normal) exit from the subroutine, IRB and IRC will have the values they had on the original jump to the subroutine - not the value they may have been given after the alarm exit.

When the alarm exit occurs, AR will be zero if $w < x[0]$, and will contain $4(n-p)$ if $w > x[n-1]$.

5. Alarm stops

There is one alarm stop in the subroutine, the instruction 103A830 located in 103A8. This stop will occur in case of an overflow or improper division in the double precision calculation corresponding to the last box on the flow chart. For reasonably small values of p - say, less than about 8 - this stop should not usually occur.

6. Accuracy

The most critical part of the routine, the calculation of a new extrapolation from two previous ones, is done in double precision (80 bits). See the instructions in 73A8 to 95A8.

7. Method used

The method is Neville's variation of Aitken's interpolating method, and is described in detail on page 73 of Numerical Calculus, by W. E. Milne. The details of the algorithm may be determined from the flow chart and Algol program, which are part of this write up. See particularly the comments in the Algol program. Essentially, for p-point interpolation a (p-1)-st degree polynomial is passed through the p points surrounding the given argument, and the value of the polynomial at w is obtained. The algorithm gives f(w) without explicitly determining the polynomial.

8. Flow chart

On page 4 of this writeup is given a flow chart of the subroutine.

9. Algol program

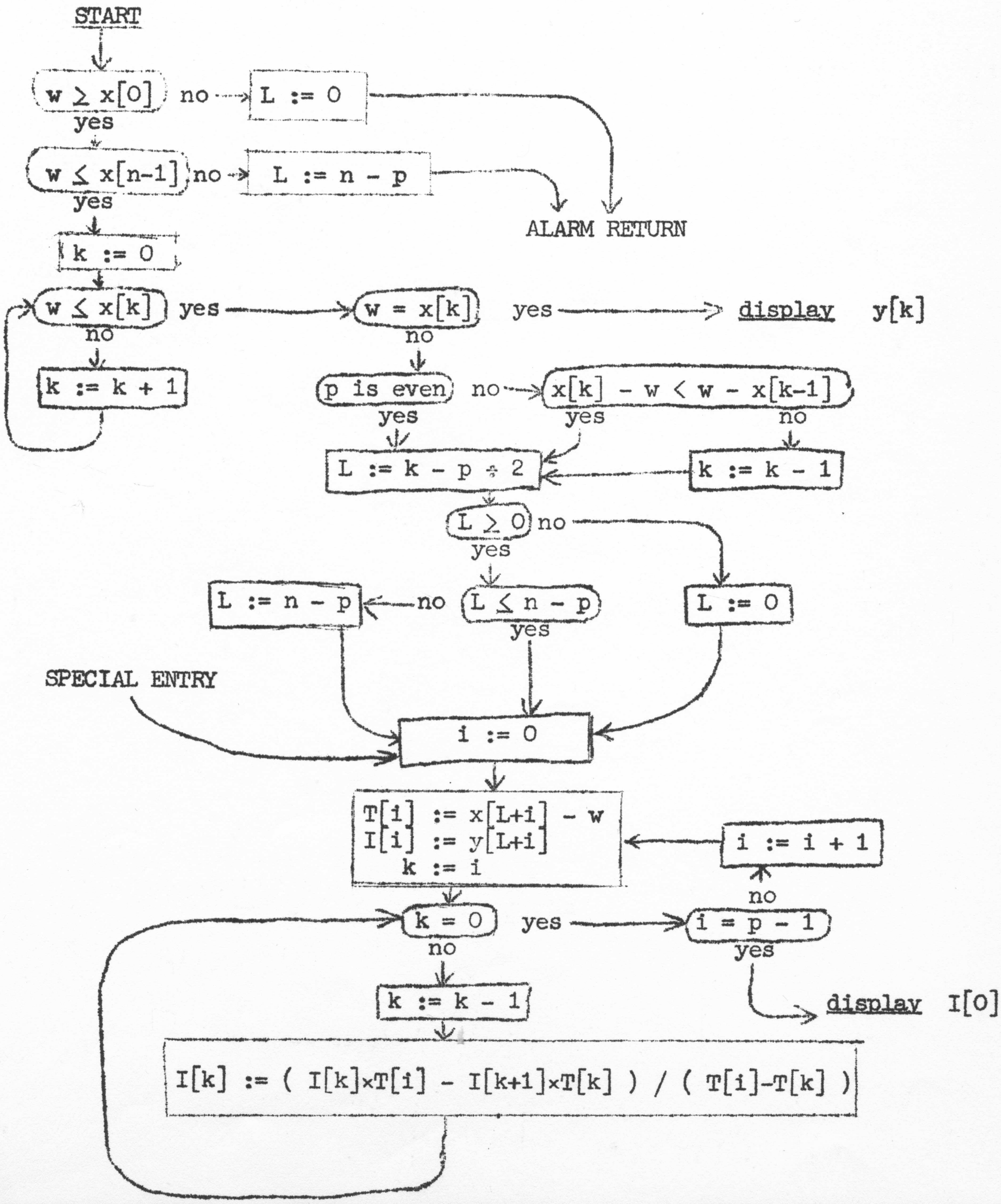
On page 5 of this writeup is given an Algol 60 procedure whose effect will be exactly that of the subroutine (although, of course, the calling sequence is different). Since an Algol procedure may have only one entrance, the effect of the Alarm entrance to 58A8 is provided by switch, a Boolean variable, used to indicate whether to extrapolate or return to the alarm return if the argument w is outside the range of the table.

The identifiers used in the Algol program for variables are the same ones used above in the subroutine description and below in the flow chart.

10. Listing

On pages 6 to 8 of this writeup is given a listing of the **subroutine**. The following information may be of some help to the interested user. (All variable names correspond to those in the flow chart.) The quantity i is kept in IRB, and k is in IRC. T[i] is in location (110+4i)A8, and I[k] is in (112+4k)A8. The calculation corresponding to the box on the bottom of the flow chart is done double precision, in that 80-bit products are kept for the two multiplications, and the division is with the 80-bit difference as a numerator.

Flow chart of the subroutine



Algol 60 version of the Neville interpolation subroutine:

```
real procedure Neville (w, x, y, n, p, Alarm, switch) ;   value w, n, p ;
array x, y ;  real w ;   integer n, p ;  label Alarm ;  Boolean switch ;

comment:  y is a tabulated function of x, with y[i] = f(x[i]), and there
 are n pairs of points.  For an x-value w, Neville will perform p-point
 interpolation by the Neville variation of Aitken's method to get an
 approximation to f(w).  The x[i] need not be evenly spaced, but they
 must be monotone increasing.
          If w is outside the range of the x[i],  (that is, if  w < x[0]
 or if w > x[n-1] ), switch will be examined.  If it is true the
 desired extrapolation will be performed, while if switch is false
 an exit will be made to Alarm.
          Exact equality of  w  and  x[L]  for some  L  is treated as
 a special case, and the answer  y[L]  is given immediately.
          L will be chosen so that the interpolation, performed using the
 points x[L], x[L+1], ... , x[L+p-1], will be around the best possible
 point.  If p is even, the interpolation will be centered about w.  If
 p is odd, the center point will be the one nearest to w.  If w is too
 close to (or beyond) the end of the table, a suitable adjustment will
 be made to L.
          Restrictions:  1)  x[i+1] > x[i], for all i.
                         2)  n ≥ p.
 The effect is undefined if the above two restrictions are not met.
          The I[k] of this program will, for a given i, contain the value
 of the Aitken polynomial  I[k, k+1, ... , i].     ;

begin    integer L ;

         if w < x[0] then  L := 0
         else if w > x[n-1] then  L := n - p
         else go to  start  ;
         if switch then go to loop else go to Alarm  ;

start:   L := 0  ;
   AA:   if  w > x[L]  then begin L := L + 1 ;  go to AA  end  ;
         if  w = x[L]  then  begin  Neville := y[L] ;  go to  done  end  ;
         if  p ≠ 2×(p÷2)  then
         begin  if  x[L]-w ≥ w-x[L-1]  then  L := L - 1  end  ;
         L := L - p÷2  ;
         if  L < 0  then  L :=  0  else if  L > n - p  then  L := n - p  ;

 loop:   begin  integer  i, k ;  array  I, T [0:p-1] ;
                for i := 0 step 1 until p-1 do
                begin  T[i]  := x[L+i]  - w  ;
                       I[i]  := y[L+i]       ;
                       for  k := i-1 step  -1  until  0  do
                       I[k]  := ( I[k]×T[i] - I[k+1]×T[k] ) / ( T[i]-T[k] )
                end i  ;
                Neville  := I[0]   ;
         end  main computation block  ;
 done:
end Neville
```

Neville Interpolation Subroutine - Listing:

LOCAT	FROM	ADRES	LOC	INSTRUCTION	COMMENTS
START		w	0	104 A8 08	save argument in w
			1	3 D 61	fetch p, negative
			2	2039 A 20	
			3	2 A 0C	- 4×(p-1)
		p4	4	98 A8 29	
		DIF	5	107 A8 28	
			6	2 D 60	fetch n
			7	2039 A 21	
			8	2 A 0C	4×(n-1)
		PA	9	18 A8 29	
		DIF	10	107 A8 26	DIF := 4×(n-p)
		XB	11	100 A8 34	save IRB
		XC	12	101 A8 54	save IRC
		w	13	104 A8 40	
			14	1 D 37	table base to IRB
			15	0 B 01	w - x[0]
		PA	16	18 A8 11	jump if w ≥ first argument
		PB	17	22 A8 50	
PA	16,9	(n4)	18	(0) B 40	fetch x[n-1]
		w	19	104 A8 01	
		PC	20	24 A8 11	jump if w ≤ last argument
		DIF	21	107 A8 60	
PB	17	L	22	94 A8 29	L := zero or 4×(n-p)
			23	4 D 10	ALARM RETURN
PC	20	PE	24	27 A8 34	set address of PE to table base
			25	2044 A 55	k := -1
PD	29		26	4 C 55	k := k + 1
PE	33,39,24	(B)	27	(0) C 44	fetch x[k] to AR and MR
		w	28	104 A8 01	
		PD	29	26 A8 51	cycle back if w > x[k]
			30	2041 A 01	now check for exact equality of
		PF	31	35 A8 11	x[k] and w
			32	2 C 55	yes: so fetch function value
		PE	33	27 A8 37	and return
		XB	34	100 A8 10	
PF	31		35	3 D 60	fetch p
			36	11 A 0C	low order bit of p to sign
		PI	37	47 A8 11	jump if p is even
		PG	38	41 A8 34	set address to table base
		PE	39	27 A8 37	fetch x[k]

LOCAT	FROM	ADRES	LOC	INSTRUCTION	COMMENTS
			40	2044 C 55	
PG	38	(B)	41	(0) C 00	x[k-1]
		w	42	104 A8 01	
		w	43	104 A8 01	set k correctly
		PH	44	46 A8 11	
			45	4 C 55	
PH	44		46	2039 A 60	
PI	37		47	3 D 21	for even p: -p; for odd p: -(p-1)
			48	1 A 0C	-(p÷2) ×4
		L	49	94 A8 54	
		L	50	94 A8 26	L := k - (p÷2)
		PJ	51	53 A8 11	
		PK	52	57 A8 50	for L < 0, set AR to zero and jump
PJ	51	DIF	53	107 A8 60	
		L	54	94 A8 21	
		ENTER	55	58 A8 11	jump if L ≤ 4×(n-p)
		DIF	56	107 A8 60	
PK	52	L	57	94 A8 29	set L to 0 or 4×(n-p)
ENTER	55		58	1 D 60	table base to AR
		L	59	94 A8 20	
		QA	60	65 A8 29	set address to B+L (first point to be used)
		QB	61	68 A8 29	
		QB	62	68 A8 66	set address to B+L+2
			63	2044 A 35	i := -1
LOOPA	99		64	4 B 35	i := i + 1
QA	60	(B+L)	65	(0) B 40	
		w	66	104 A8 01	
		T	67	110 B8 08	T[i] := x[L+i] - w
QB	61, 62	(B+L+2)	68	(0) B 40	
		I	69	112 B8 08	I[i] := y[L+i]
			70	0 B 55	k := i
		TEST	71	96 A8 10	skip to end of inner loop
LOOPB	96		72	2044 C 55	k := k - 1
		T	73	110 B8 40	
		T	74	110 C8 01	
		D	75	108 A8 08	D := T[i] - T[k]
		I	76	112 C8 44	
		T	77	110 B8 4A	I[k] × T[i], 80 bit product
		N	78	106 A8 08	
			79	0 A 07	

LOCAT	FROM	ADRES	LOC	INSTRUCTION	COMMENTS
		I	80	112 C8 08	
		I+4	81	116 C8 45	
		T	82	110 C8 4A	I[k+1] × T[k], 80 bit product
		N	83	106 A8 06	left half of difference
		ALARM	84	103 A8 12	
			85	0 A 07	
		N	86	106 A8 00	right half of difference to MR
			87	39 A 4F	
		N	88	106 A8 06	add left half
		ALARM	89	103 A8 12	
		D	90	108 A8 4B	divide, 80 bit numerator
		D	91	108 A8 42	
		N	92	106 A8 03	abs(D) - abs(N)
		ALARM	93	103 A8 51	alarm for improper division
L			94	(0) A 07	temp: 22, 49, 50, 54, 57, 59
		I	95	112 C8 08	store quotient in I[k]
TEST	71	LOOPB	96	72 A8 53	
			97	0 B 55	for k=0, test end of outer loop
p4	4	(p4)	98	(0) C 55	IRC := IRB - 4×(p-1)
		LOOPA	99	64 A8 53	cycle on outer loop if i ≠ p-1
XB	34, 11		100	(0) A 35	restore IRB
XC	12		101	(0) A 55	restore IRC
			102	5 D 10	RETURN
ALARM		ALARM	103	103 A8 30	alarm halt: 84, 89, 93, 103
w			104	temp	0, 13, 19, 28, 42, 43, 66
			105		
N			106	temp	78, 83, 86, 88, 92
DIF			107	temp	21, 53, 56
D			108	temp	75, 90, 91
			109		
T			110	temp	67, 73, 74, 77, 82
			111		
I			112	temp	69, 76, 80, 81, 95