ARKIV FÖR KEMI Band 31 nr 27

4.68 12 15 Read 9 October 1968

High-speed computers as a supplement to graphical methods. 8

Some devices to speed up computations on chemical equilibria and simplify programming for LETAGROP. Application to complex formation

By Robert Arnek, Lars Gunnar Sillén and Olle Wahlberg

ABSTRACT

Repeated solution of chemical equilibrium and mass balance equations, which is needed for instance in application of LETAGROP to equilibrium problems, is speeded up considerably by a package of procedures, BDTV, containing i.a. an improved version of procedures Kålle and Kille. By the introduction of these and a few other general procedures, the writing of new special programs can be much simplified.

As an example is given the program for adjusting the formation constants for a number of species A_pB_q from data $Z(\log a, B)$ or $\eta(\log a, B)$, where Z is the average number of A bound per B, and $\eta = \log (B/b)$.

Our general minimizing program LETAGROP searches for the set of parameters (for instance equilibrium constants) that will minimize a certain function (typically an error square sum) U. The majority of the blocks are the same for all types of problems; the only special blocks are PUTS for preliminary data treatment and UBBE for the calculation of U from the parameters ($\vec{\mathbf{k}}$, $\mathbf{k}_s$) and the data (**ag**, **as**, **ap**).

In the course of time the general program has been applied to many different problems, chemical and others, and in all about 40 different special blocks have been used. Since many of these special blocks contained quite similar parts, it was thought desirable to reorganize PUTS and UBBE so that as little rewriting as possible would be necessary for each new problem. Hence, a number of convenient common procedures were made up.

Moreover, whereas initially less attention had been paid to the economy of time for solving equations than to the reliability of the algorithms, an effort was made to speed up the computations.

The present paper reports on some such improvements in the special parts.

Mass balance and equilibrium equations; Kuska and early Kålle

Suppose we have two or three reagents, A and B, or A, B and C, which can form a number of complex species, A_pB_q or $A_pB_qC_r$. Their formation constants will be called β_{pq} or β_{pqr}. If we denote the total concentrations in a solution by capital letters A, B, C, and the concentrations of the free reagents by small letters a, b, c, then the conditions for mass balance and equilibrium give, for two components

$$A = a + \sum p\beta_{pq} a^p b^q; \quad B = b + \sum q\beta_{pq} a^p b^q \qquad (1\,\mathrm{a},\ 1\,\mathrm{b})$$

With three reagents we have analogously

$$A = a + \sum p\beta_{pqr} a^p b^q c^r; \quad B = b + \sum q\beta_{pqr} a^p b^q c^r; \quad C = c + \sum r\beta_{pqr} a^p b^q c^r \qquad (2\,\mathrm{a},\ 2\,\mathrm{b},\ 2\,\mathrm{c})$$

In equilibrium calculations, one usually knows either the total or the free concentration for each component, assumes a certain set of equilibrium constants and wishes to calculate the concentrations of all species, simple or complex, in the solution. In our work hitherto, the most common combinations of known quantities have been, for two reagents (a, B) or (A, B), and for three reagents (a, B, C).

Now, the functions (1) and (2) all increase steadily. For instance, if a is kept constant, $B(b)$ in equation (1 b) is an increasing function, either the q values are positive or negative. For the solution of such equations we first used a program KUSKA (part 2). As a start, a value for b was guessed, and B was calculated from (1 b). If, for instance, the calculated B (and hence the guessed b) was too small, b was increased by a given step; if b was still too small, first twice that step, then four times the step etc was added, until one had two b values, one on each side of the right value. Then the range between them was halved repeatedly, and the right value confined to a smaller and smaller range until one had a b value that satisfied the equation (1 b) within the tolerance prescribed.

In part 4 another set of instructions is described, which finds the value x_0 that makes a certain increasing function $y(x)$ equal to y_0. In the present case, $y_0 = B$ and $x = \ln b$. These instructions (after label SLINGA) were subsequently written as a procedure Kålle.

The plan in Kålle is that $x = \ln b$ is changed by steps of 2 units until one has one value on each side the right one, after which the steps are halved as in Kuska. However, when the steps have reached a certain limit (*"stegbyt"*), a more rapid approach is reached by drawing chords.

The method of Kålle is also applied (at ADA) in our program HALTAFALL (part 5), for solving a general equilibrium problem with an arbitrary number of reagents.

Since there are a number of unknowns to adjust, HALTAFALL uses essentially Kålle (i) with an integer parameter, and indexed variables for the quantities used in Kålle. In problems for LETAGROP with several unknowns we have used procedure Kålle without a parameter for the innermost loop (which is most frequently used) and a completely analogous procedure Kille(i) for the other loops.

In most of our early special programs for LETAGROP, Kålle and Kille(i) were used to solve equations for increasing functions. Our main interest was to have a completely reliable method since equations of types (1) or (2) were to be solved thousands of times in each calculation, and even a probability of misbehavior of 10^{-4} would be enough to ruin most calculations.

In the course of a LETAGROP calculation, the concentrations of the components in each solution are calculated a number of times, with slightly shifted values for the equilibrium constants. So, we sometimes used the earlier value for the concentrations as the first guess in the next calculation. However, this was not done consistently.

Newton–Raphson method and improved Kålle method

Dr. Alberto Vacca from the University of Firenze, Italy, while visiting our laboratory for some computer work, pointed out that he had been able to speed up the calculations in LETAGROP for a case with $(a\ B\ C)$ known, by consistently recording and using the previously solved values for b and c, and by applying the Newton-Raphson method. Hence, we decided to make an attempt to speed up the calculations. On the one hand we tried a Newton-Raphson approach based on the equations (from 2b and 2c)

$$\delta B = \delta \ln b(b + \sum q^2 c_{pqr}) + \delta \ln c(\sum qr c_{pqr}) \qquad (3\,b)$$

$$\delta C = \delta \ln b(\sum qr c_{pqr}) + \delta \ln c(c + \sum r^2 c_{pqr}) \qquad (3\,c)$$

For any initial guess of $\ln b$ and $\ln c$, one may calculate B and C, and then calculate the corrections to be made to reach the values desired. To avoid misbehavior on not too good guesses, we maximized the changes in $\ln b$ or $\ln c$ to $+2$ and -2.

On the other hand, we tried to improve the work of Kålle and related parts of the program, with the following differences from the version in part 4:

(*a*) The variables to be changed are still logarithmic, $\ln b$ and $\ln c$ (or $\ln a$), but as in the earlier Kuska, the steps are doubled until one has one value on each side of the right one, after which the steps are halved.

(*b*) Earlier calculations for the same data point are recorded and used as the first guess in the next calculation; in the very first calculation, the first guess for each point (except the very first one) is the value found for the preceding point. (We also did the same in the Newton–Raphson calculation).

(*c*) The first step is chosen quite small, as $0.4 \times$ *stegbyt*, once (*b*) has given a guess of the right order of magnitude; with good luck, the second point will be on the other side, so that the chord method can be used at once.

(*d*) A device is built in for counting the number of turns, "varv" (Swedish = turn), through the central loop in Kålle. If the computer has not solved the equations within the tolerance desired, after a given number ($v_{\max}$) of turns the values for x and y are printed for a number of successive points, which gives an idea of why the equations are solved so slowly, and how to correct the input after *Rurik* = 8.

Calculations were carried out on a certain set of emf titrations, with two equations of type (2b) and (2c) to solve, and a number of "shots" in LETAGROP were passed to compare the time taken by the two methods. It was then found that the Newton–Raphson method was considerably faster than the earlier Kålle. However, the improved Kålle was as fast as Newton–Raphson, and even faster when suitable steps were chosen. Since Kålle also has the advantage of being generally applicable to all increasing functions $y(x)$, without the necessity of writing the derivatives explicitly, we decided to use the improved Kålle rather than Newton–Raphson in the following.

For convenience, the various procedures for solving equilibrium and mass balance equations were combined to form a fixed package called BDTV (see below).

Special example on complex formation

Table 1 gives the special blocks PUTS and UBBE for two problems to be used together with the general LETAGROP program given in parts 6 and 7. The problems are characterized by the number $Typ=1, 2$.

Z (and η) problem, $Typ=1(2)$

Background. The components A and B form a series of complexes, A_pB_q, each with its own formation constant, β_{pq}. The equilibrium law gives

$$c_{pq}=[A_pB_q]=\beta_{pq}a^pb^q \tag{4}$$

where a and b are as before the concentrations of free A and B. The total concentration of B is B(“*Btot*”) mol/l, and BZ mol A per liter are bound to B, so that Z is the average number of A bound per B.

$$B=b+\sum qc_{pq}=b+\sum q\beta_{pq}a^pb^q \tag{1 b}$$

$$BZ=\sum pc_{pq}=\sum p\beta_{pq}a^pb^q \tag{5}$$

With $Typ=1$, we know a, A and B for each point and may calculate $Z=(A-a)/B$. B is kept constant in each group, which is usually the data from a single potentiometric titration. A small systematic error in the analysis for A may make the experimental Z too high by a constant value δZ, throughout the group.

With $Typ=2$, we also measure b (e.g. with a metal or amalgam electrode) so that we know

$$\eta=\log(B/b)=\log(1+\Sigma q\beta_{pq}a^p\,b^{q-1}) \tag{6}$$

In each group (usually an emf titration) B is again constant, and we assume that there are constant systematic errors δZ and $\delta\eta$, which may be different in different groups. We shall let the arrays for the data and parameters have the following meaning.

$Typ=1$: $ag=$(none); $as=B_{\text{tot}}$; $ap=\log a, Z, +\ln a, +\ln b$; $k=\beta_{pq}$;
$ak=pot,p,q$; $ks=\delta Z$
$Typ=2$: $ag=$(none); $as=B_{\text{tot}}$; $ap=\log a, Z, \eta, +\ln a, +\ln b$;
$k=\beta_{pq}$; $ak=pot, p, q$; $ks=\delta Z, \delta\eta$

The “+” before an *ap* means that the value is not given in the input but calculated in PUTS or UBBE. E.g. the last *ap*, “$+\ln b$” is the value for $\ln b$ stored for the next application of Kålle.

Calculation

Assuming a set of β_{pq} values, (1 b) is in both cases used to calculate $\ln b$ for each point from the known $\ln a$ and B. For each Typ we may choose, by selecting the control number *val*, between several definitions of the error square sum

$$U=\Sigma fel[val]^2 \tag{7}$$

Table 1. PUTS and UBBE for Z + ETA problem. The general parts include "BDTV"
Procedures subordinate to Valhal are marked out by an asterisk * for clarity.

```
PUTS:         begin if Rs = 0 then begin Napa: = if Typ = 1 then 4 else 5 ; goto DATA end ;
              cell: = apcell[Rs] ;
              for Rp: = 1 step 1 until Np[Rs] do begin ap[cell + Napa - 1]: = ln10 × ap[cell + 1] ;
                        cell: = cell + Napa end ;
              goto DATA end PUTS ;
UBBE:         if Koks and not Tage and not Rakt then goto SÄRK ; comment not for Spefo ;
              begin real , Atot, B2HX, Btot, CHX, CHXall, Ctot, dil, E, E0, Eber, Efak, Ej,
                        Hfri, jac, KHX, Kwjalk, lna, lnb, lnc, lnh, lnl, lnv, Ltot, temp ;
              integer dirt, ix; Boolean Yksi ; real array beta, c, lnbeta[1:20], fel[1:6] ;
              integer array fas, p, q, r, t[1: 20] ;
              comment special variables and procedures;
              real etaber, Zber ;
              switch Apfel: = Apfel1, Apfel2; switch Asoks: = Asoks1,Asoks2; switch Kag: = Kag1,
                        Kag2; switch Satsa: = Satsa1,Satsa2; switch Uttåg: = Uttåg1,Uttåg2;


              comment BDTV = procedures Betain, Dirty, Totber, Valhal ;
procedure     Betain(i1,i2) ; integer i1, i2 ; begin ix: = 0;
              Nkom: = if Fas2 then Nak - 2 else Nak - 1 ; Yksi: = true ;
              for ik: = i1 step 1 until i2 do begin ix: = ix + 1 ; pot[ix]: = ak[ik,1] ;
                 if k[ik] > 0 then lnbeta[ix]: = ln(k[ik]) + ln10 × pot[ix] else lnbeta[ix]: = -1000;
                 p[ix]: = ak[ik,2] ;
                 if Nkom > 1 then begin q[ix]: = ak[ik,3] ;
                    if q[ix]≠0 and q[ix]≠1 then Yksi: = false end ;
                 if Nkom > 2 then r[ix]: = ak[ik,4] ; if Nkom > 3 then t[ix]: = ak[ik,5] ;
                 if Fas2 then fas[ix]: = ak[ik,Nak] ; beta[ix]: = exp(lnbeta[ix]) end ;
              Nx: = ix end Betain ;
procedure     Dirty ; if dirt > 0 then begin Hfri: = exp(lna); w: = CHX × dil + CHXall ;
              if dirt = 1 then y: = y - w × KHX /(KHX + Hfri) ;
              if dirt = 2 then y: = y - w × (2 × B2HX + KHX × Hfri)/(Hfri × (Hfri + KHX) +
                        B2HX)
              end Dirty ;
procedure     Totber(tal) ; integer array tal ; begin
              y: = if Fas2 then 0 else exp(x) ;
              for ix: = 1 step 1 until Nx do y: = y + c[ix] × tal[ix] end Totber ;


procedure     Valhal (BA,arum) ; integer BA,arum ; begin
              real fak, step, step0, tola, tolb, tolc, toll, toly, x1, x2, y0, y1, y2 ;
              integer Karl, Nvar, varv ; Boolean Avar, Bvar, Nytt ;
              real array lnba, lnba1[1:20], faki, stegi, tolyi, x1i, x2i, y1i,y2i[1:5] ;
              integer array Kal[1:5] ;
*procedure    Aprov(i) ; integer i ; begin
              x: = lna ; y0: = Atot ; Totber(p) ; Dirty ;
              if i = 0 then Kålle else Kille(i);
              if not Bra then lna: = x else goto Nog end Aprov ;
*procedure    Byksis ; begin w: = if Fas2 then 0 else 1 ;
              for ix: = 1 step 1 until Nx do w: = w + q[ix] × exp(lnba[ix]) ;
              x: = lnb: = ln(Btot/w) end Byksis ;
*procedure    Cber(tal) ; integer array tal ;
              for ix: = 1 step 1 until Nx do begin w: = lnba[ix] + tal[ix] × x ;
              if w > 2 then w: = 2 ; c[ix]: = exp(w) end Cber ;
*procedure    Cprov(i) ; integer i ; begin
              x: = lnc ; y0: = Ctot ; Totber(r) ;
              if i = 0 then Kålle else Kille(i) ;
              if not Bra then lnc: = x ;
              if Bra and Nvar = i + 1 then goto Nog end Cprov ;
```

```
*procedure   Kille(i) ; value i ; integer i ; begin switch Ada: = Ada1,Ada2,Ada3,Ada4;
             w: = abs(y − y0) ; if tolyi[i] > w then begin Bra: = true ; goto Slut end ;
             Bra: = false ;
             if y > y0 then begin x2i[i]: = x ; y2i[i]: = y end
                else begin x1i[i]: = x ; y1i[i]: = y end ;
             goto Ada[Kal[i]] ;
Ada1:        faki[i]: = 2 ; Kal[i]: =  if y > y0 then 3 else 2 ;
             stegi[i]: = 0.5 × stegi[i] ; goto Kliv ;
Ada2:        if y > y0 then goto Grepp else goto Kliv ;
Ada3:        if y < y0 then goto Grepp else goto Kliv ;
Grepp:       faki[i]: = 0.5; Kal[i]: = 4 ;
Ada4:        if stegi[i] < stegbyt then goto Korda ;
Kliv:        stegi[i]: = faki[i] × stegi[i] ;
             if y > y0 then x: = x − stegi[i] else x: = x + stegi[i] ; goto Slut ;
Korda:       w: = (x2i[i] − x1i[i])/(y2i[i] − y1i[i]) ; x: = x1i[i] + w × (y0 − y1i[i]) ; goto Slut ;
Slut:        end Kille ;
*procedure   Kålle ; begin switch Ada: = Ada1,Ada2,Ada3,Ada4;
             w: = abs(y − y0); if toly > w then begin Bra: = true ; goto Slut end ;
             Bra: = false ;
             if y > y0 then begin x2: = x ; y2: = y end else begin x1: = x ; y1: = y end ;
             varv: = varv + 1 ; if varv > vmax − 1 then begin
                if  varv = vmax then   output(61,  '/,B'VARV',3ZD,B,'SATS',2ZD,B,'PUNKT',
                      2ZD,B,'Y0 = ', − ZD.7D$_{10}$ − 2ZD',  varv,Rs,Rp,y0) ;
                output(61, '/,B 'X = ', − 3ZD.7D,B'Y = ', − ZD.7D$_{10}$ − 2ZD', x,y) ;
                if varv > vmax + 30 then begin Bra: = true ; goto Nog end end ;
             goto Ada[Karl] ;
Ada1:        fak: = 2 ; Karl: =  if y > y0 then 3 else 2 ; step: = 0.5 × step ; goto Kliv ;
Ada2:        if y > y0 then goto Grepp else goto Kliv ;
Ada3:        if y < y0 then goto Grepp else goto Kliv ;
Grepp:       fak: = 0.5 ; Karl: = 4 ;
Ada4:        if step < stegbyt then goto Korda ;
Kliv:        step: = fak × step ; x: = if y > y0 then x − step else x + step ; goto Slut ;
Korda:       w: = (x2 − x1)/(y2 − y1) ; x: = x1 + w × (y0 − y1) ; goto Slut ;
Slut:        end Kålle ;
*procedure   Lnbad (bas,tal,lnat) ; real array bas ; integer array tal ; real lnat ;
             for ix: = 1 step 1 until Nx do bas[ix]: = bas[ix] + tal[ix] × lnat ;
*procedure   Lprov(i) ; integer i ; begin x: = lnl ; y0: = Ltot ; Totber(t) ;
             Kille(i) ; if not Bra then lnl: = x ;
             if Bra and Nvar = i + 1 then goto Nog end Lprov ;

             Avar: = if BA = 1 or BA = 3 then true else false ;
             Bvar: =  if BA > 1 then true else false ;
             Nvar: =  Nkom ;
             if not Avar then Nvar: = Nvar − 1 ; if Nkom > 1 and not Bvar then Nvar: = Nvar − 1 ;
             if Bvar and Btot ≤ 0 then begin lnb: =  − 1000 ; Bvar: = false ; Nvar: = Nvar − 1 end ;
             if Orvar = 0  and  (not  Tage  or  (Tage  and  Orvarko = 0))  then  Nytt: = true
                else Nytt: = false ;
             tola: = tol[Nkom] ; tolb: = if Btot > 0 then Btot × tol[1] else tol[1] ;
             if Nkom > 2 then begin
                tolc: = if Ctot > 0 then Ctot × tol[2] else tol[2] ;
                if Nkom > 3 then toll: = if Ltot > 0 then Ltot × tol[3] else tol[3] end ;
             if Bvar then begin toly: = tolb ;
                tolyi[1]: = if (Nvar = 2 and Avar) then tola else tolc ;
                if Nvar > 2 then tolyi[2]: =  if Avar then tola else toll end ;
             if not Bvar then begin
                toly: =  if Nvar  =  1 and Avar then tola else tolc ;
                if Nvar > 1 then begin tolyi[1]: =  if (Avar and Nvar  =  2) then tola else toll ;
                  if Nvar  =  3 then tolyi[2]: = tola end end ;
             varv: = 0 ; m: = cell + arum ; step0: = 0.4 × stegbyt ;
             if Nytt then begin if Bvar then lnb: = start[1] ;
                if Nkom > 2 then lnc: = start [2] ; if Nkom > 3 then lnl: = start[3] ;
                if Avar then lna: = start[Nkom] ; if Rp = 1 then step0: = 1.0 end
```

```
               else begin if Avar then lna: = ap[m] ; if Bvar then lnb: = ap[m + 1] ;
                  if Nkom > 2 then lnc: = ap[m + 2] ; if Nkom > 3 then lnl: = ap[m + 3] end ;
               step: = step0 ;
               if Nvar > 1 then begin Kal[1]: = 1 ; stegi[1]: = step0 ;
                  if Nvar > 2 then begin Kal[2]: = 1 ; stegi[2]: = step0 end end ;
               for ix: = 1 step 1 until Nx do lnba1[ix]: = lnbeta[ix] ;
               if Fas2 then Lnbad(lnba1,fas,lnv) ;
               if not Avar then Lnbad(lnba1,p,lna) ; if Nkom > 1 and not Bvar then
                        Lnbad(lnba1,q,lnb) ;
Lnbas:         for ix: = 1 step 1 until Nx do lnba[ix]: = lnba1[ix] ; if Nvar = 0 then goto Nog ;
               if Avar and Nvar = 1 then goto Arunt ;
               if Avar then Lnbad(lnba,p,lna) ;
               if (Nkom > 2 and Bvar) then Lnbad(lnba,r,lnc) ;
               if Nkom > 3 then Lnbad(lnba,t,lnl) ;
               if Bvar then goto Brunt else goto Crunt ;
Arunt:         Karl: = 1 ; x: = lna ;
Tjata:         Cber(p) ; Aprov(0) ; goto Tjata ;
Brunt:         if Yksi then Byksis ; y0: = Btot ; Karl: = 1 ; x: = lnb ;
Tjatb:         lnb: = x ; Cber(q) ; Totber(q) ; Kålle ;
               if not Bra then goto Tjatb ; if Nvar = 1 then goto Nog ; goto Klivut ;
Crunt:         Karl: = 1 ; x: = lnc ;
Tjatc:         Cber(r) ; Cprov(0) ; if not Bra then goto Tjatc ; goto Klivut ;
Klivut:        if Nvar = 2 and Avar then begin Aprov(1) ; goto Lnbas end ;
               if Bvar then Cprov(1) else Lprov(1) ; if not Bra then goto Lnbas ;
               Kal[1]: = 1 ; if Avar then Aprov(2) else Lprov(2) ; goto Lnbas ;
Nog:           if Nytt then begin if Btot > 0 then start[1]: = lnb ;
                  if Nkom > 2 and Ctot > 0 then start[2]: = lnc;
                  if Nkom  > 3 and Ltot  > 0 then start[3]: = lnl ;
                  start[Nkom]: = lna end ;
               ap[m]: = lna ; if Nkom > 1 then ap[m + 1]: = lnb ;
               if Nkom > 2 then ap[m + 2]: = lnc ;
               if Nkom > 3 then ap[m + 3]: = lnl end Valhal ;
               comment here ends BDTV ;

               comment general introduction to UBBE ;
               U: = 0 ; if not Tage then Rs: = Rs1 ; goto Kag[Typ] ;
Nysa:          Rp: = 0 ; cell: = apcell[Rs] – Napa ;
               if Rurik = 2 then goto Satsa[Typ] ; goto Asoks[Typ] ;
Nyp:           Rp: = Rp + 1 ; cell: = cell + Napa ;
               if Rp > Np[Rs] then begin
                  if Rs2 > Rs and not Tage then begin Rs: = Rs + 1 ; goto Nysa end
                  else begin
                     if Skrikut  =  1 or (Skrikut  =  0 and not Tage) or (Prov and not (Tage and
                           Skrikut < 0)) then begin UVAR ; if Prov then output(61, ‘/’) end ;
                     goto UBBEUT end end ;
               goto Apfel[Typ] ;
Uber:          U: = U + fel[val] ↑ 2 ; if Rurik = 2 then goto Uttåg [Typ] ; goto Nyp ;
               comment special orders ;
Kag1:Kag2:     Betain(1,Nk) ; dirt: = 0 ; goto Nysa ;
Satsa1:        SATSUT ; output(61, ‘/,18B‘F = 1000 × ”), output(61, ‘/‘loga Zexp Zber – Zexp
                        F × Btot F/Zber”) ;
               goto Asoks1 ;
Satsa2:        SATSUT ; output(61, ‘/, 27B‘1000 ×  ’4B‘1000 × ”) ; output(61, ‘/4B‘ LOGA’7B  ‘Z’
                        5B ‘ETA (ZBER – Z) (ETABER – ETA)”) ;
               goto Asoks2 ;
Asoks1:Asoks2: Btot: = as[Rs,1] ; if Btot ⩽ 0 then Rp: = Np[Rs] ; goto Nyp ;
Apfel1:        lna: = ap[cell + 3] ; Valhal(2,3) ; x: =  – 1000 ; Totber(p) ; Zber: = y/Btot ;
               fel[1]: = Zber – ap[cell + 2] + ks[Rs,1] ;
               fel[2]: = Btot × fel[1] ; fel[3]: = fel[1]/Zber ;
               goto Uber ;
Apfel2:        lna: = ap[cell + 4]  ; Valhal(2,4) ;  x: =  – 1000  ; Totber(p) ; Zber: = y/Btot ; etaber: =
                        (ln(Btot) – lnb)/ln10 ;
               fel[1]: = Zber – ap[cell + 2] + ks[Rs,1] ;
```

```
              fel[2]: = etaber − ap[cell + 3] + ks[Rs,2] ;
              goto Uber ;
Uttågl:       ouput(61, '/B, −ZD.4D, −ZD.5D,3( +3ZD.2D)',  ap[cell + 1],  ap[cell + 2],  1000 ×
                    fel[1], 1000 × fel[2], 1000 × fel[3]) ;
              goto Nyp ;
Uttåg2:       output(61, '/, −ZD.4D,2( −ZD.5D), 2( −3ZD.2D)',  ap[cell + 1],  ap[cell + 2],  ap
                    [cell + 3], 1000 × fel[1], 1000 × fel[2]);
              goto Nyp end UBBE ;
FINAL:        end LETAGROP ;
```

Typ = *1*. We insert the calculated ln b into (5). This gives a first calculated value Z_{ber} which is corrected by the assumed error δZ. The errors are defined as follows

$$fel[1] = Z_{\text{ber}} + \delta Z - Z_{\text{exp}};\ fel[2] = B_{\text{tot}} \times fel[1];\ fel[3] = fel[1]/Z_{\text{ber}} \tag{8}$$

Typ = *2*. From the calculated ln b and (5) we again obtain Z_{ber}, from ln b and (6) we get η_{ber}. We introduce the group parameters δZ (corresponding to a small analytical error) and $\delta\eta$ (corresponding to an error in the E_0 value assumed):

$$fel[1] = Z_{\text{ber}} + \delta Z - Z_{\text{exp}};\ fel[2] = \eta_{\text{ber}} + \delta\eta - \eta_{\text{exp}} \tag{9}$$

The program can be used even if only one of η and Z is measured; the other is given as 0 in the input, and the corresponding output is ignored or may be suppressed by changing the program.

If there are some groups where both Z and η will be used, and some where only one of them has been measured, or can be used, the groups may be arranged as follows; 1) groups with only Z, 2) groups with Z and η, 3) groups with only η; *Rs1* and *Rs2* are chosen accordingly for *val* = 1 and *val* = 2. When quantities that have not been measured have been set equal to 0 in the output the corresponding "error" should of course not be treated as such.

Typ = *1*. Constant input ("data"): 14(*Rurik*), text, 9(*Rurik*), 1(*Typ*), 6(*Rurik*), *Ns*, 0(*Nag*), 1(*Nas*), 2(*Nap*), $(Np, B, (\log a, Z,)_{Np})_{Ns}$.

Variable input ("Dagens spaning") may begin: 7(*Rurik*), *Nk*, *Nk*, 3(*Nak*), $(k, pot, p, q)_{Nk}$, 1(*Nks*), 0,0 (if all $k_s = 0$ in first attempts), 0 (if no information on twist matrix is to be used), 11, *Rs1*, *Rs2*, 8(*Rurik*), 1(*Nok*), *stegbyt*, *start*(ln b), *tol*($B_{\text{calc}}/B_{\text{tot}}$)

Typ = *2*. Constant input: 14(*Rurik*), text, 9(*Rurik*), 2(*Typ*), 6(*Rurik*), *Ns*, 0(*Nag*), 1(*Nas*), 3(*Nap*), $(Np, B, (\log a, Z, \eta)_{Np})_{Ns}$

Dagens spaning may begin 7(*Rurik*), *Nk*, *Nk*, 3(*Nak*), $(k, pot, p, q)_{Nk}$, 2(*Nks*), 0,0 (if all δZ and $\delta\eta$ are set = 0 at the beginning), 0(*skin*), 8(*Rurik*), 1(*Nok*), *stegbyt*, *start*(ln b), *tol*($B_{\text{calc}}/B_{\text{tot}}$).

Comment on UBBE

The common part (labels Nysa, Nyp, Uber) in this and other UBBE blocks contains references to five switches where the correct label is found according to the value of *Typ*:

Kag reads the common parameters k and the common data *ag* and transforms them to quantities useful for UBBE. If the k are equilibrium constants, with attached values for the power of 10 (*pot*) and numbers p and q (r, t, *fas*), then procedure Betain reads them from the memory and transforms them to logarithms (*lnbeta*) and numerical values (*beta*).

Satsa: the headline for a new group ("sats") is printed if a complete output is asked for ($Rurik=2$).

Asoks: the data *as* and parameters *ks* characteristic of a certain group are transformed to quantities to be used in the calculation of U.

Apfel: the data characteristic of each point, *ap*, are used for calculating the errors ("*fel*"), the squares of which are used in the error square sum U(eqn. 7).

Uttåg: values for various experimental data and calculated quantities are printed if $Rurik=2$.

The mass balance equations (1) or (2) are solved at Apfel where a single reference to Valhal with two parameters is enough.

Comment on BDTV

BDTV (Betain + Dirty + Totber + Valhal) is a set of procedures that solve the equilibrium and mass balance equations for various cases with 2–4 reacting components. When BDTV is applied in various versions of LETAGROP, certain conventions must be followed for naming the reactants and storing the data.

If there are two components (as in the special case considered), they are called A and B, if three A, B and C, if four A, B, C and L. The coefficients in a species are written as follows: $A_pB_qC_rL_t$, and form four arrays, *p*[*ix*] etc. In cases with two-phase distribution equilibria, there is one more array of coefficients, *fas*[*ix*].

In the case with four components we have for each complex

$$c_i=[A_pB_qC_rL_t]=\beta_i a^p b^c c^r l^t \tag{10}$$

The mass balance equations give, for the total concentrations A, B, C, L (*Atot* etc.)

$$A=a+\Sigma p_ic_i;\quad B=b+\Sigma q_ic_i;\quad C=c+\Sigma r_ic_i;\quad L=l+\Sigma t_ic_i. \tag{11}$$

The data for the various experimental points are stored in a matrix **ap**. Each experimental point has *Napa* cells reserved for it, the first *Nap* of which contain values given in the input, whereas the rest contain values calculated in PUTS or UBBE.

The first *ap* value for the first point ($Rp=1$) in the group with serial number *Rs*, is placed as *ap*[*apcell*[*Rs*]+1]. For point number *Rp* in that group the data occupy the cells *ap* [*cell*+1] through *ap* [*cell*+*Napa*], if we set *cell*: =*apcell*[*Rs*]+(*Rp*−1)× *Napa*. In the cells, starting with *ap*[*cell*+*trum*+1], are stored *Atot*, *Btot*, *Ctot*, and *Ltot*, at last *dil* (see part 9). In the cells starting with *ap*[*cell*+*arum*] we store *lna*, *lnb*, *lnc*, *lnl* and *lnv*, where *v* is the volume ratio in the two-phase case. The numbers *arum* and *trum* are given in the program and need not bother the occasional user. The *ak* are given in the order *pot*, *p*, *q*, *r*, *t*, *fas*.

There is some lack of symmetry in the treatment of the components. Typically A is H^+, B a metal ion, C and L organic ligands, although neither is necessarily so. The concentrations of free A and B, a and b, may sometimes be measured (by emf) but for C and L only the total concentrations are known.

The starting values and tolerances *start*[*i*] and *tol*[*i*] for the components are given (after $Rurik=8$) in the order B C L A. The *tolerance* conditions for the other components are *relative* (if the total concentrations are $\neq 0$), but *for A* the tolerance is *absolute*. Since the total excess concentration of protons, H, is often negative or near zero, it is not practical to use a relative tolerance when solving (1a) or (2a) for ln a.

It is tested in Betain whether all species that contain B are mononuclear ($q=1$ or 0). If this is true, the Boolean *Yksi* (Finnish = one) is set: = **true**, and since (1b) and (2b) are then linear in b, these equations can be solved in a simpler way. (This is analogous to the treatment in HALTAFALL, part 5, but is done here only for B.)

Function

Betain transforms the k and ak values to lnβ, p, q etc. Dirty treats cases with a mono- or dibasic "dirt acid" and Totber calculates total concentrations of the components.

Valhal contains a number of subordinate procedures, among them Kille(i) and Kålle and solves the equilibrium-mass balance equations by a series of loops just as in HALTAFALL except that the order between the components is fixed to B, C, L, A. The program is controlled by the Booleans *Avar* (which is **true** if A is known but not a) and *Bvar* (**true** if B known but not b), and by *Nvar*, the number of free concentrations to be determined (unknowns). The machine can calculate *Nvar* for each new problem since the number of components *Nkom* is known (from Betain) and *Avar* and *Bvar* are given in a digital form by the parameter *BA* (3 = *Avar* and *Bvar*, 2 = *Bvar*, 1 = *Avar*, 0 = neither).

The following table gives the values for *Nkom* and *BA*, the known quantities in alphabetical order, and the order of solving the equation for the unknowns.

(*Nkom*, *BA*)	2,1	2,2	2,3	3,0	3,1	3,2	3,3	4,0	4,1	4,2
known	*Ab*	*aB*	*AB*	*abC*	*AbC*	*aBC*	*ABC*	*abCL*	*AbCL*	*aBCL*
solving order	*a*	*b*	*ba*	*c*	*ca*	*bc*	*bca*	*cl*	*cla*	*bcl*

Valhal contains a certain number of branching points where the choice is made depending on the value for *BA*. The little time the choice may take is outweighed by the advantage of having a single procedure for many different types of equilibrium calculations. Valhal can easily be extended to other types of equations, a larger number of variables etc.

Input after Rurik = 8

The general form of the input is

Rurik = 8, *Nok*, *stegbyt*, $(start,\ tol)_{Nok}$ (Input 1)

Stegbyt is the value for the step where Kålle (or Kille) passes from binary approach to "chord shooting". For equations of high degree, say n, a value for *stegbyt* $\approx 1/n$ seems appropriate. With too high a value for *stegbyt*, a chord is often drawn between two points with a strongly concave bend between them, which may lead to slower convergence.

Whereas the choice of starting values and *stegfak* may have some influence on the time needed for the calculation, a bad choice of the tolerances may ruin the whole calculation. One should be careful to notice whether the tolerance in a certain case refers to absolute or relative tolerance in the quantity appearing to the left in the equation, and then one should set a tolerance as small as possible without taking any risk that it could not be handled by the computer. If there are several unknowns, it seems advisable to use somewhat narrower (relative) tolerance limits in the innermost loops, corresponding to the *tol* values given first.

If the tolerance is taken too small, it may happen that two values for x differing only in the last digit of the computer will give values for y that are on both sides of the prescribed value y_0, and that both differ too much from it to be accepted. This calls for some caution, especially with high-degree equations. On the other hand, if the tolerance is too high, there will be a scatter in the U values because the accepted values for the unknown differ a little from the correct value. So, for instance, if one is really on a practically flat part of the $U(k)$ curve, the spread of points may still give an apparent bend which leads the computer temporarily astray in the calculations. If one has a computer that allows only a high tolerance, say 10^{-5}, it would be best to change *tolU*.

We might finally add that the present improved Kålle approach has been applied to a recent edition of HALTAFALL (5). For our applications, HALTAFALL is already so fast that the improvement in economy is not important.

ACKNOWLEDGEMENTS

We wish to thank our friends Dr. Alberto Vacca (University of Firenze, Italy) for giving us the impetus to writing the faster version of Valhal + Kålle (Kille), and for a pleasant cooperation around the CDC3600, and Mr Ashley Stevens, M. A. (Oxon) for revising the English of the manuscript. This work is part of a series of investigations supported by Statens Naturvetenskapliga Forskningsråd (Swedish Natural Sciences Research Council). Computer time was made available from funds of the Royal Institute of Technology (KTH).

Department of inorganic chemistry, Royal Institute of Technology (KTH) and the University of Stockholm, Sweden.

REFERENCES

Part 2 = Ingri, N., and Sillén, L. G., Acta Chem. Scand. *16*, 173 (1962).
Part 4 = Ingri, N., and Sillén, L. G., Arkiv Kemi *23*, 97 (1964).
Part 5 = Ingri, N., Kąkołowicz, W., Sillén, L. G., and Warnqvist, B., Talanta *14*, 1261 (1967), correction ibid. *15*, XI (1968).
Part 6 = Sillén, L. G., and Warnqvist, B., Arkiv Kemi *31*, 315 (1969).
Part 7 = Sillén, L. G., and Warnqvist, B., Arkiv Kemi *31*, 341 (1969).

Tryckt den 26 augusti 1969

Uppsala 1969. Almqvist & Wiksells Boktryckeri AB