Sq2.50 Procedure Pt 2

The next stage of the VBA code deals with 'looping' each spot through a set of procedures aimed at validating the analysis in itself, validating it against the governing Task, and evaluating Custom expressions and Built-in expression in a logical sequence, before proceeding to the Task-independent 'central processing of the U-Pb data.

Unfortunately, in the early sections of the code, Ludwig has made extensive use of the numeric values assigned to Booleans (**in VBA, TRUE = -1 and FALSE = 0**), as array-references (for a series of two-column, zero-addressed arrays). He sought to exploit the fact that, numerically, FALSE = -FALSE, and that -TRUE = 1. No doubt this is extremely 'clever', but it makes the code unnecessarily difficult to read and interpret. In this wiki, the original 'Booleans-as-indices' usage has been retained (to guard against the introduction of errors during translation), but wherever possible, plain-English comments have been added in order to clarify what is going on.

Some Definitions

Most of these appear in the VBA code, although they have been extended and reworded in places, for clarity.

- Integer piSpotNum (range from 1 to total number of spots analysed) is the index of the total vector of Spots, in time sequence.
- Integer vector piaNumSpots is a zero-addressed, two-element vector containing counts of the number of analyses by type. piaNumSpots[0] = number of **Sample** spots in the run (91 in demo XML); piaNumSpots[1] = number of **Reference 206/238 material** spots in the run (23 in demo XML).
- Integer array piaSpots is a two-row array in which the rows are zero-addressed (range 0 to 1) but the columns are not (range 1 to max(piaNumSpots) i.e. 91 in demo XML). The first row (i.e. piaSpots[0, x]) contains the index numbers (i.e piSpotNum values) for each **Sample** spot in the run; the second row (i.e. piaSpots[1, x]) contains the index numbers (i.e piSpotNum values) for each **Reference 206/238 material** spot in the run. Unused elements in the shorter of the two rows default to zero.
- Integer vector piaSpotIndx is a zero-addressed, two-element vector containing the index of piaSpots[w, x] of the current spot being processed. piaSpotIndx[0] = piaSpots[0, x] when a **Sample** spot is being processed, and piaSpotIndx[1] = piaSpots[1, x] when a **Reference 206/238 material** spot is being processed.
- Integer vector piaSpotCt is a zero-addressed, two-element vector containing counts of the number of analyses **processed so far**. piaSpotCt[0] = number of **Sample** spots processed so far; piaSpotCt[1] = number of **Reference 206/238 material** spots processed so far.

Paraphrased Code

```
pbStd = FALSE
```

"Start of Reference Material-Sample Loop":

```
Do --Start of Loop A
  pbStd = (pbStd = FALSE) Or (pbStdsOnly = TRUE)
  --The first time through Loop A, this sets pbStd = TRUE,
  --in which case integer/index "[-pbStd]" = 1

  pbDone = FALSE

  Do --Start of Loop B ("first loop of spots")

    Do --Start of Loop C (counting, and "basic" parsing of spot)

      piaSpotCt[-pbStd] = 1 + piaSpotCt[-pbStd]
      piaSpotIndx[-pbStd] = 1 + piaSpotIndx[-pbStd]
      piSpotNum = piaSpots[-pbStd, piaSpotIndx[-pbStd] ]

      ParseRawData piSpotNum, TRUE, IgnoredChangedRunTable, DateStr, TRUE,
          FALSE, TRUE
```

See separately documented subroutine ParseRawData for the argument list. Note that (1) FirstPass = TRUE here (as opposed to the prior call of ParseRawData in GetConcStdData, documented in Sq2.50 Procedure Pt 1, and (2) the prior value of IgnoredChangedRunTable is not specified, but this is ultimately irrelevant as it is set to FALSE near the start of ParseRawData anyway. The incomplete Loop C proceeds as follows:

```
      If (IgnoredChangedRunTable = TRUE) And (SpotNscans > 1)
        MsgBox("Run Table changes at spot number " & StR[piSpotNum]
            & " -- terminating.")
        CrashEnd
      End If

    Loop Until IgnoredChangedRunTable = FALSE --End of Loop C
```

Loop C usually only runs once because IgnoredChangedRunTable is usually FALSE throughout. When it does get set to TRUE (via failure of a ParseRawData test), it still only runs once, because CrashEnd is immediately triggered in the vast majority of cases. But note that there seems to be a gap in the VBA code logic here, in the case where an analysis with

SpotNscans = 1 failed a ParseRawData test, resulting in IgnoredChangedRunTable = TRUE. It looks like such an analysis could never escape Loop C… but I don't have any 'real data' to test this.

The incomplete Loop B proceeds as follows:

```
    DateStr = psaSpotDateTime[piSpotNum]
    ParseTimedate DateStr, Seconds
```

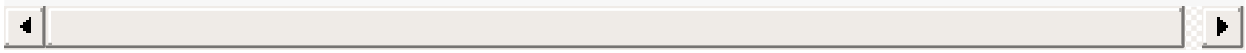
ParseTimeDate is a Ludwig function designed to convert a date-string (DateStr) into its representation as a number of seconds (Seconds) elapsed since the commencement of calendar 1990 (seems arbitrary; presumably chosen because SHRIMP output was not computerised prior to that). The incomplete Loop B proceeds as follows:

```
    If FirstSecond = 0
      FirstSecond = Seconds
    End If

    plSpotOutputRw = 1 + plSpotOutputRw
```

In the following, 'CFs' reflects a Ludwig function best considered as 'Cell Fill with String', and 'CF' reflects 'Cell Fill'. In each case, the function appears to have three arguments: the row the cursor is to be placed in, the column the cursor is to be placed in, and the value to be placed. The incomplete Loop B proceeds as follows:

```
    CFs plSpotOutputRw, 1, psSpotName
    CFs plSpotOutputRw, piDateTimeCol, DateStr
    CF plSpotOutputRw, piHoursCol, Excel.Fixed((Seconds - FirstSecond) / 3600, 3)
```



Excel.Fixed invokes the Excel spreadsheet function FIXED, which returns a text representation of a number rounded to a specified number (3 here) of decimal places (e.g. https://www.techonthenet.com/excel/formulas/fixed.php). The third of these three statements calculates the 'Hours' value, as shown in the third column of processed SQUID-books, as a double-precision number (to 3 decimal places) defining the time elapsed in the analytical session since the commencement of the **first Reference 206/238 material** analysis.

At the time the code was written, this was presumably supposed to rigorously reflect the first analysis collected, chronologically. However, at Geoscience Australia (and probably

elsewhere), sessions rarely commence with an analysis of the Reference 206/238 material, with the result that a small handful of spots have negative Hours values. This is not an operational problem: the primary function of the Hours column is to provide easy chronological sorting of any set of analyses **and** to provide an easy 'axis' for time-dependent representations and calculations (e.g. X-Y plots where X is analytical session time, and calculation of any associated X-Y regressions).

The incomplete Loop B proceeds as follows:

```
GetRatios Ratios(), RatioFractErrs(), False, BadSbm()
PlaceRawRatios plSpotOutputRw, Ratios, RatioFractErrs
```

This pair of statements 'gets and places' the 'isotope ratios' for this row (i.e. in the initial instance, the first (chronological) analysis of the reference 206Pb/238U material). In addition to the Spot Name, Spot Date/Time and Hours data placed previously by the CFs and CF statements above, this pair of statements results in the placement of:

- All spot-specific data harvested directly from the XML file (in SQUID 2.50, these are the QT1Y and QT1Z bit-values, the Stage X, Stage Y and Stage Z positions, and the primary beam intensity in nA).
- BackgroundCps and all TotCps values calculated at the end of SHRIMP: Step 2
- All RatioVal and RatioFractErr (the latter as percentage) values, as calculated for each 'ratio of interest' by SHRIMP: Step 4. Note that this does **not** include the results of any Task expressions, as these calculations have not yet been performed!

The procedure then turns firstly to the *custom* expressions of the governing Task. This first pass through the expressions is (I think) primarily aimed at finding which of the 'custom' expressions requires evaluation as input to any of the 'built-in' expressions. The incomplete Loop B proceeds as follows:

```
For EqNum = 1 to Task.Neqns --Remember 'built-ins' have EqNum < 0

  If Switch.LA = FALSE And Switch.SC = FALSE

    If Switch.FO = TRUE Or Switch.AR = TRUE Or piaEqnRats[EqNum] = 0
      --Second condition is redundant, I think

      Formulae Task.EqnAsString[EqNum], EqNum, pbStd, piaEqCol[pbStd, EqNum]
      --to be documented/paraphrased separately

      If Switch.FO = FALSE
        EqnResu = Cells[SpotOutputRw, piaEqCol[pbStd, EqNum] ]
      End If
```

```
      Else --essentially equivalent to "If Switch.NU = TRUE"

        EqnInterp Task.EqnAsString[EqNum], EqNum, EqnRes, EqnFerr, 1, 0, TRUE
        --See separate documentation of Sub EqnInterp for argument list

        EqnResu = StR( EqnRes ) --double-to-string conversion
        EqnFerro = StR( 100 * EqnFerr ) --formal construction of %err from Ferr
        CFs plSpotOutputRw, piaEqCol[pbStd, EqNum], EqnResu
        CFs plSpotOutputRw, piaEqECol[pbStd, EqNum], EqnFerro
        --Note "piaEqCol" vs "piaEqECol" in the last two lines

      End If

    End If

  Next EqNum

Loop Until piaSpotIndx[-pbStd] = piaNumSpots[-pbStd] --End of Loop B
```

This marks the end of the first loop through a set of spots (i.e either reference 206Pb/238U materials **or** samples, depending on the stage of the processing. Note that Loop A is not yet closed, and the next stage of the processing revisits the Task's **Custom** expressions, this time looking at 'single-cell' (or 'single-array') results generated from **columns** of pre-existing data. The incomplete Loop A proceeds as follows:

```
plHdrRw = flHeaderRow[pbStd]
--Fixed index-number of header-row: 6 for ref 206/238 material, 2 for samples
Frw = FirstDatRw[-pbStd]
--First data-row: essentially = 1 + plHdrRw
--(i.e. 7 for ref 206/238 material, 3 for samples)
Lrw = LastDatRw[-pbStd]
--Last data-row: essentially = piaNumSpots[-pbStd] + plHdrRw (i.e. for demo XML,
--ref 206/238 material = 23 analyses, plHdrRw = 6, so Lrw = 29, and for
--samples = 91 analyses, plHdrRw = 2, so Lrw = 93.
```

With Frw and Lrw defined, the next step is to place any 'single-cell' (or 'single-array') expressions for which Switch.LA (i.e. 'LAst') = FALSE. The incomplete Loop A proceeds as follows:

```
For EqNum = 1 to Task.Neqns --Custom expressions only!

  If Switch.LA = FALSE And ( (Switch.AR = TRUE And Switch.ARrows = 1) Or
    (Switch.SC = TRUE) ) --logic repaired from Ludwig; see below
```

```
        Formulae Task.EqnAsString[EqNum], EqNum, pbStd, Frw, piaEqCol[pbStd, EqNum]
        --to be documented/paraphrased separately
      End If

    Next EqNum
```

The above If statement incorporates repairs to the conditions originally imposed by Ludwig: his VBA code omitted "Switch.AR = TRUE And" under the assumption that this Boolean was implied by a non-zero value for Switch.ARrows. Unfortunately, a bug in the Task Editor permits the proliferation of non-zero values for Switch.ARrows and Switch.ARcols, even when Switch.AR = FALSE, and the real presence of this prohibited combination permitted expressions with Switch.AR = FALSE to satisfy the If condition (it is safe to say this was unintended!). Consequences included 're-evaluation' of NU-switched expressions as FO-switched (in the first data-row only), as observed by Jim Bowring in the 29 April 2017 iteration of the 100142_ShowcaseTask… output.

Loop A remains unfinished, but now that the first loop through the spot-groups has been completed (via Loop B and subsequent), it is time to reset the counters and prepare for the second loop through the spots.

```
plSpotOutputRw = plHdrRw
piaSpotIndx[-pbStd] = piaStartSpotIndx[-pbStd] - 1
piaSpotCt[-pbStd] = 0
```

The second loop through the spot-groups encompasses the calculation and placement, row-by-row, of the Daughter-Parent ratios/constants, using the **built-in** expressions. This will be documented in Procedural Framework: Part 3.

Sq2.50 Procedure Pt 3

Task parameters: Every Task has two fundamental binary switches that control its function and roles of its built-in expressions:
1) "Ratio of primary interest": The choices are 206Pb/238U (i.e. 'primary parent nuclide' Booleans pbU = TRUE, pbTh = FALSE) or 208Pb/232Th (i.e. pbU = FALSE, pbTh = TRUE), and **exactly one** of these combinations must be chosen.
2) "Method for calculating secondary ratio": The choices are "Directly, via the reference zircon" or "Indirectly, via 232Th/238U", and **exactly one** must be chosen. SQUID 2.50 stores this as a Boolean named Switch.DirectAltPD.

The first of these is self-explanatory (and 99+% of the time, it is 206Pb/238U, so pbU = TRUE, and pbTh = FALSE). **Note** that 'built-in' EqNum = -1 **always** refers to the ratio of primary interest, no matter which is chosen.

The secondary ratio is simply whichever of the two candidates was not chosen as the primary ratio, and there are two ways in which it can be calculated. The first (but more rare!) is to simply determine the secondary ratio directly, via calculation of calibration constants analogous to those used for the primary ratio. In the case of zircon, this would entail calculating a '208Pb/232Th calib. constant' alongside the 206Pb/238U calib. constant we are already familiar with. The problem with this approach is usually the reference zircon: it is unusual for any material to be perfectly fit for purpose in both the U-Pb **and** Th-Pb systems, as the geochemical behaviour of U and Th are quite different in most geological environments, and properly closed U-Th-Pb systems seem to be quite difficult to maintain in nature. If (**and only if**) the 'direct' method is employed, ' built-in' EqNum = -2 **always** refers to the ratio of secondary interest, no matter which is chosen. If the 'direct' method is not selected, EqNum = -2 is null, and EqNum = -3 will be non-null (see below).

The alternative (but more common!) method of calculating the secondary ratio is indirect, and relies on the appropriate rearrangement of the general equation:

```
(208Pb/232Th) = (206Pb/238U) * (238U/232Th) * (208Pb/206Pb)
```

Note that one of the first two ratios is independently accounted for (as the primary ratio of interest, via EqNum = -1), and the other is the secondary ratio we are interested in evaluating. In general, the fourth ratio is relatively easily measured, which means we can determine our target ratio **if** we have a means of evaluating the third ratio. In the case of zircon, Williams et al. (1996) formulated an empirical relationship that works well enough; it takes the (familiar) form:

```
232Th/238U = (0.03446 * ["254/238"] + 0.868) * ["248/254"]
```

where the ratios in square brackets represent isotopic "ratios of interest" measured directly via SHRIMP. If (**and only if**) the 'indirect' method is employed, ' built-in' EqNum = -3 **always** refers to the expression specifically governing 232Th/238U. If the 'indirect' method is not selected, EqNum = -3 is null, and EqNum = -2 will be non-null (see above).

So in general, one of EqNum = -2 and EqNum = -3 will always be null, and it is possible for **both** to be null, in the special case where only one system is being investigated (e.g. U-Pb geochronology where no proxy for Th is being measured (e.g. a 'normal' zircon list of mass-stations lacking the 248 (ThO) peak).

So, there are four permutations of the two binary switches, and they are all valid in an isotopic sense:

- Case 1: Primary ratio = 206Pb/238U, Switch.DirectAltPD = FALSE (this is the most common in SHRIMP geochronology)
- Case 2: Primary ratio = 206Pb/238U, Switch.DirectAltPD = TRUE
- Case 3: Primary ratio = 208Pb/232Th, Switch.DirectAltPD = FALSE
- Case 4: Primary ratio = 208Pb/232Th, Switch.DirectAltPD = TRUE

Iteration-specific parameters for a Task: The above relates to fundamental properties of an individual Task. There are a series of auxiliary settings, applied to a Task solely for a specific iteration of its use. In theory, these auxiliary settings should not interact with the fundamental Task properties… in practice, the way SQUID 2.50 is constructed, there is some overlap.

We have already dealt with two of these 'auxiliary' settings:
1) SBM-normalisation (on or off), and
2) Isotope-ratio calculation (weighted average or linear regression)

A third, which unfortunately does interact with the four Cases above, is the choice of 'index isotope'. The 'index isotope' refers to the identity of the isotope of Pb used to calculate the proportion of non-radiogenic Pb present in any given analysis, and perform the appropriate correction. The two most-frequently used options are 204Pb and 207Pb, as neither are used in the fundamental daughter-parent ratios that form the basis of U-Th-Pb geochronology. SQUID 2.50 does offer a third option: 208Pb. Note, however, that this option is not available to any Task involving **direct** calculation of a 208Pb/232Th calibration constant (irrespective of whether 208Pb/232Th is the primary or secondary ratio), and this restriction rules out Cases 2-4. As a result, when index isotope is added to the permutations, there are nine valid combinations requiring consideration:

- Case 1a: Primary ratio = 206Pb/238U, Switch.DirectAltPD = FALSE, index isotope = 204Pb
- Case 1b: Primary ratio = 206Pb/238U, Switch.DirectAltPD = FALSE, index isotope = 207Pb

- **Case 1c:** Primary ratio = 206Pb/238U, Switch.DirectAltPD = FALSE, index isotope = **208Pb**
- Case 2a: Primary ratio = 206Pb/238U, Switch.DirectAltPD = TRUE, index isotope = 204Pb
- Case 2b: Primary ratio = 206Pb/238U, Switch.DirectAltPD = TRUE, index isotope = 207Pb
- Case 3a: Primary ratio = 208Pb/232Th, Switch.DirectAltPD = FALSE, index isotope = 204Pb
- Case 3b: Primary ratio = 208Pb/232Th, Switch.DirectAltPD = FALSE, index isotope = 207Pb
- Case 4a: Primary ratio = 208Pb/232Th, Switch.DirectAltPD = TRUE, index isotope = 204Pb
- Case 4b: Primary ratio = 208Pb/232Th, Switch.DirectAltPD = TRUE, index isotope = 207Pb

Traditionally, SQUID 2.50 requires the user to select a single index isotope, and the calculations for **one** of the nine permutations above are carried out. In general, I think it would be better if, once the Case number is established from the binary switch settings, the calculations were carried out for **all** valid index isotopes for that Case. This would save the user needing to test each index isotope via sequential iterations of data reduction (as SQUID 2.50 currently requires).

The superset of calculatable parameters, identified by column-header, is given below. In all cases, these are row-by-row calculations aimed at filling cells in an array where the rows represent individual analyses, as per the StandardData sheet produced by SQUID 2.50. **Note** that no single data-reduction will ever calculate all of these: every Case will exclude some of them. The parameters are grouped in the list below, to try and highlight some of the data structure.

Calibration constants uncorrected for common Pb

- UncorrPb/Uconst
- UncorrPb/Uconst %err
- UncorrPb/Thconst
- UncorrPb/Thconst %err

Pb/U calibration constants corrected from common Pb

- 4-corr206Pb/238Ucalibr.const
- 4-corr206Pb/238Ucalibr.const %err
- 4-corr206Pb/238U Age(Ma)
- 4-corr206Pb/238U Age(Ma) ±1s
- 7-corr206Pb/238Ucalibr.const
- 7-corr206Pb/238Ucalibr.const %err

- 7-corr206Pb/238U Age(Ma)
- 7-corr206Pb/238U Age(Ma) ±1s
- 8-corr206Pb/238Ucalibr.const
- 8-corr206Pb/238Ucalibr.const %err
- 8-corr206Pb/238U Age(Ma)
- 8-corr206Pb/238U Age(Ma) ±1s

Pb/Th calibration constants corrected for common Pb

- 4-corr208Pb/232Thcalibr.const
- 4-corr208Pb/232Thcalibr.const %err
- 4-corr208Pb/232Th Age(Ma)
- 4-corr208Pb/232Th Age(Ma) ±1s
- 7-corr208Pb/232Thcalibr.const
- 7-corr208Pb/232Thcalibr.const %err
- 7-corr208Pb/232Th Age(Ma)
- 7-corr208Pb/232Th Age(Ma) ±1s

Geochemical parameters

- ppmU
- ppmTh
- 232Th/238U
- 232Th/238U %err

Common and radiogenic 206Pb and 208Pb

- 4-corr%com206
- 7-corr%com206
- 8-corr%com206
- 4-corr%com208
- 7-corr%com208
- 4-corr208Pb/*206Pb*
- 4-corr208Pb/*206Pb* %err
- 7-corr208Pb/*206Pb*
- 7-corr208Pb/*206Pb* %err

204Pb overcounts assessed via 207Pb and 208Pb

- 204overcts/sec(fr. 207)
- 204overcts/sec(fr. 208)
- 204/206(fr. 207)

- 204/206(fr. 207) %err
- 204/206(fr. 208)
- 7-corr206Pb/238Uconst.delta%
- 8-corr206Pb/238Uconst.delta%

Dates and Concordia data

- 4-corr207Pb/206Pbage
- 4-corr207Pb/206Pbage ±1s
- 4-corr207Pb/206Pb
- 4-corr207Pb/206Pb %err
- 4-corr207*/235
- 4-corr207*/235 %err
- 4-corr206*/238
- 4-corr206*/238 %err
- 4-corr errcorr
- 7-corr206*/238
- 7-corr206*/238 %err
- 8-corr207Pb/206Pbage
- 8-corr207Pb/206Pbage ±1s
- 8-corr207Pb/206Pb
- 8-corr207Pb/206Pb %err
- 8-corr207*/235
- 8-corr207*/235 %err
- 8-corr206*/238
- 8-corr206*/238 %err
- 8-corr errcorr

With this 'Task-anatomy' defined, we can rejoin the code. The next phase of Loop A is aimed at the evaluation of many of the columns listed above, with emphasis on the role of the Case-number in defining the exact expression used.

Loop A remains unfinished, but now that the first loop through the spot-groups has been completed (via Loop B and subsequent), it is time to reset the counters and prepare for the second loop through the spots. This second loop encompasses the calculation and placement, row-by-row, of the Daughter-Parent ratios/constants, using the **built-in** expressions.

```
Do --Start of Loop D ("second loop of spots": calculation and placement, row-by-row
   --of the Daughter-Parent ratios/constants)

  Do --Start of Loop E (counting, and "basic" parsing of spot, as per Loop C)
    piaSpotCt[-pbStd] = 1 + piaSpotCt[-pbStd]
```

```
    piaSpotIndx[-pbStd] = 1 + piaSpotIndx[-pbStd]
    piSpotNum = piaSpots[-pbStd, piaSpotIndx[-pbStd] ]

    ParseRawData piSpotNum, FALSE, IgnoredChangedRunTable
  Loop Until IgnoredChangedRunTable = FALSE --End of Loop E
```

See separately documented subroutine ParseRawData for the argument list. Note that (1) FirstPass = FALSE here (as opposed to the prior call of ParseRawData in Sq2.50 Procedure Pt 2, and (2) the prior value of IgnoredChangedRunTable is not specified, but this is ultimately irrelevant as it is set to FALSE near the start of ParseRawData anyway. (I am not certain of the purpose of Loop E, given that every analysis that reaches this point has already been through Loop C.)

Ludwig Q2

SQUID 2.50 proceeds by determining how many Daughter-Parent ratios require direct evaluation. If Switch.DirectAltPD = TRUE, then piNumDauPar = 2 (i.e. calibration constants must be calculated for both 206Pb/238U and 208Pb/232Th); otherwise, piNumDauPar = 1. SQUID 2.50 starts by eliminating the trivial case where the user has specified Switch.DirectAltPD = TRUE but provided no expression for evaluation of the calibration constant of the secondary ratio. The incomplete Loop D proceeds as follows:

```
plSpotOutputRw = 1 + plSpotOutputRw
plOutputRw = plSpotOutputRw
MaxDPnum = piNumDauPar

If MaxDPnum = 2 AND Switch.DirectAltPD = TRUE AND Task.Eqns[-2] = ""
  MaxDPnum = 1
End If
```

SQUID 2.50 then proceeds to evaluate as many calibration constants as are needed (at least one, via EqNum = -1, is mandatory), using the EqnInterp subroutine documented previously, and placing the result in the appropriate column. SQUID 2.50 does this slightly awkwardly, because it always stores the primary ratio calibration constant in the same place; it just applies different column-header names depending on whether the primary ratio is 206Pb/238U or 208Pb/232Th. I think it would be a better idea to simply have all the relevant columns permanently available and named, calculate whichever values are necessary/permitted, and then designate which of the calibration constants is the primary one.

The following code-block populates at least some of UncorrPb/Uconst, UncorrPb/Uconst %err, UncorrPb/Thconst, UncorrPb/Thconst %err, depending on binary switch settings. Note

that the EqnInterp-related arithmetic has already been performed (without context) in Calamari, and the result has been output under the name '206/238 Calib Const'.

```
For DauParNum = 1 To MaxDPnum
  EqnInterp Task.Eqns[-DauParNum], -DauParNum, EqnRes, EqnFerr, 1, TmpRej

  --Next 'If' simply identifies sheets/columns for placing calculated values
  If pbStd = TRUE
    ValCol = piaStdUnCorrAcol[DauParNum]
    ErCol = piaStdUnCorrAerCol[DauParNum]
  Else
    ErCol = piaAeCol[DauParNum]

    If pbCanDriftCorr = TRUE --beyond current scope of SQUID 3.0!
      ValCol = piUnDriftCorrConstCol
    Else
      ValCol = piaAcol[DauParNum]
    End If
  End If

  --Now place EqnRes in ValCol, and EqnFerr (as %) in ErCol
  CFs plSpotOutputRw, ValCol, fsS(EqnRes) --corrects Ludwig's
      CDbl-CSng-CDbl error
  --fsS is a Ludwig function that converts a double-precision number
      to a string.
  --It is not obvious why fsS is employed here.

  If EqnRes <> SQUID_Error_Value AND EqnFerr <> SQUID_Error_Value
    CFs plSpotOutputRw, ErCol, fsS(100 * EqnFerr) --corrects Ludwig's
        CDbl-CSng-CDbl error
    --fsS is a Ludwig function that converts a double-precision number
        a string.
    --It is not obvious why fsS is employed here.
  End If

  If DauParNum = 1
    piWLrej = TmpRej
  End If
Next DauParNum
```

The next step is to populate row-by-row the concentrations of the 'primary element' ppm values. This will populate **at most one** of the columns ppmU, ppmTh:

```
StdElePpm pbStd, plSpotOutputRw --subroutine documented separately
```

Ludwig Q3

Next is a test to see if the run-table contains **both** U-bearing peaks **and** Th-bearing peaks. If so, then it is possible (but not mandatory), based on the 'primary element' ppm values, to calculate 232Th/238U, as well as 'secondary element' ppm values.

SQUID 2.50 uses the Booleans pbHasU and pbHasTh to represent the existence of U- and Th-bearing peaks in the run-table of interest. The values are assigned in a fairly unsophisticated way: pbHasU = TRUE if there is a mass-station in the run-table with a nominal mass of 238 (= U) and/or 254 (= UO) and/or 270 (= UO2), and likewise PbHasTh = TRUE if there is a mass-station in the run-table with a nominal mass of 232 (= Th) and/or 248 (= ThO) and/or 264 (= ThO2). These tests are a good first approximation, but should not be the sole determinant of the Boolean values (as they currently **are** in SQUID 2.50).

A more rigorous approach for SQUID 3.0 would be to display check-boxes next to each mass-station on the run-table screen (one for 'U-bearing' and one for 'Th-bearing'), each ticked TRUE or FALSE according to the above criteria, but user-editable in case of unforeseen nominal masses for U- and/or Th-bearing mass-stations. Then pbHasU could be an aggregated Boolean i.e. set to TRUE if one or more of the mass-station-specific 'U-bearing' check-boxes was marked; likewise pbHasTh would be set to TRUE if one or more of the mass-station-specific 'Th-bearing' check-boxes was marked. Feedback to this effect could be displayed on the run-table screen.

The incomplete Loop D proceeds:

```
If pbHasTh = TRUE AND pbHasU = TRUE

  If piNumDauPar = 1  --"number of daughter-parents"
  --i.e. Switch.DirectAltPD = FALSE and therefore
  --Secondary U-Th/Pb expression (EqNum = -2) is NULL

    ThUfromFormula pbStd, plSpotOutputRw
    --calculates 232Th/238U using EqNum = -3
    --Calamari already does this, without context
    --subroutine documented separately

    SecondaryParentPpmFromThU pbStd, plSpotOutputRw
    --calculates 'secondary element' ppm value, using combination
    --of 'primary element' value and calculated 232Th/238U
    --subroutine documented separately
```

The 'Else' of this unfinished 'If' relates to piNumDauPar = 2, which corresponds to Switch.DirectAltPD = TRUE and therefore Secondary U-Th/Pb expression (EqNum = -2) = NOT NULL. It predominantly caters for Sample (rather than Standard) analyses, for which 'total 206/238' (as well as 'total 238/206' when pbU = TRUE) and 'total 208/232' values are calculated using the respective calibration constants.

From there, and irrespective of whether the analysis is Standard or Sample, 232Th/238U is also calculated based on each pair of calibration constants (although Ludwig has noted these expressions need to be re-evaluated later because 'column mean' values for the calibration constants do not yet exist).

Finally, for Sample analyses only, the 'secondary element' ppm value is calculated, using the combination of 'primary element' ppm value and calculated 232Th/238U. The incomplete 'If' statement proceeds as follows:

```
Else --i.e. if piNumDauPar = 2
--i.e. Switch.DirectAltPD = TRUE and therefore Secondary
--U-Th/Pb expression (EqNum = -2) is NOT NULL

  If pbStd = FALSE --i.e. sample analysis
    Tot68_82_fromA plSpotOutputRw
    --Produces total 206/238 and total 208/232 from
    --calibration constants A(206/238) and A(208/232)
    --subroutine documented separately
  End If

  ThUfromA1A2 pbStd, plSpotOutputRw, TRUE --following from Ludwig:
  --NOTE: must recalc later because WtdMeanA1/2 range doesn't yet exist
  --subroutine documented separately

  If pbStd = FALSE --i.e. sample analysis
    SecondaryParentPpmFromThU pbStd, plSpotOutputRw
    --subroutine documented separately
  End If

End If
```

Ludwig Q4

The next stage of the incomplete Loop D deals with Standard analyses ONLY, firstly by evaluating 204 'overcounts' on a spot-by-spot basis, and secondly by applying a 'common-Pb correction' to the calculated calibration constant value(s) for each spot (and augmenting the calibration constant %err value for each spot, to reflect the additional uncertainty associated with the 'common Pb correction').

The first part of Ludwig Q4 populates the following StandardData columns:

- 204overcts/sec(fr. 207)
- 204overcts/sec(fr. 208)
- 204/206(fr. 207)
- 204/206(fr. 207) %err

- 204/206(fr. 208)
- 7-corr206Pb/238Uconst.delta%
- 8-corr206Pb/238Uconst.delta%

The second part of Ludwig Q4 populates the following StandardData columns:

- 4-corr206Pb/238Ucalibr.const
- 4-corr206Pb/238Ucalibr.const %err
- 7-corr206Pb/238Ucalibr.const
- 7-corr206Pb/238Ucalibr.const %err
- 8-corr206Pb/238Ucalibr.const
- 8-corr206Pb/238Ucalibr.const %err
- 4-corr208Pb/232Thcalibr.const
- 4-corr208Pb/232Thcalibr.const %err
- 7-corr208Pb/232Thcalibr.const
- 7-corr208Pb/232Thcalibr.const %err

The incomplete Loop D proceeds as follows:

```
If pbStd = TRUE

  If Switch.ShowOverCtCols = TRUE --SQUID 2.50 does not offer this
  --Switch as a User-option; presumably set to TRUE by default

    OvercountColumns (plSpotOutputRw)
    --calculates & places apparent 204 overcount data
    --subroutine documented separately
    --first part of Ludwig Q4 completed

  End If

  --Second part of Ludwig Q4 commences:
  --correct 206/238 and/or 208/232 calibration constants for common Pb, and
  --augment calibration constant %errs to reflect uncertainty related to correcti

  For DauParNum = 1 to piNumDauPar
  --i.e. for one or both of the daughter-parent combinations that can be calibrat
  --check to see if the system (for this iteration) is Pb/U or Pb/Th, by setting

    If (pbU = TRUE AND DauParNum = 2) OR (pbTh = TRUE AND DauParNum = 1)
      IsPbTh = TRUE
    Else
      IsPbTh = FALSE
    End If

    --Now calculate common Pb-corrected calibration constant VALUES:
    --piStdCorrType = 0 means '204-corrected'
    --piStdCorrType = 1 means '207-corrected'
    --piStdCorrType = 2 means '208-corrected'
```

```
--Traditionally SQUID 2.50 requires ONE piStdCorrType value, but
--I would like SQUID 3.0 to calculate ALL piStdCorrType values.

If piStdCorrType = 0 -- '204-corrected'

  If IsPbTh = TRUE

    FinalTerm1 = (1 - ["204/206"] / ["208/206"] * sComm84)
        * ["UncorrPb/Thconst"]
    --fills column ["4-corr208Pb/232Thcalibr.const"]

  Else

    FinalTerm1 = (1 - ["204/206"] * sComm64) * ["UncorrPb/Uconst"]
    --fills column ["4-corr206Pb/238Ucalibr.const"]

  End If

Elseif piStdCorrType = 1 -- '207-corrected'

  If IsPbTh = TRUE

    FinalTerm1 = (1 - ["204/206(fr. 207)"] / ["208/206"] * sComm84)
        * ["UncorrPb/Thconst"]
    --fills column ["7-corr208Pb/232Thcalibr.const"]

  Else

    FinalTerm1 = (1 - ["204/206(fr. 207)"] * sComm64) * ["UncorrPb/Uconst"]
    --fills column ["7-corr206Pb/238Ucalibr.const"]

  End If

Elseif piStdCorrType = 2 -- '208-corrected', can't use Pb/Th calibration here

  FinalTerm1 = (1 - ["204/206(fr. 208)"] * sComm64) * ["UncorrPb/Uconst"]
  --fills column ["8-corr206Pb/238Ucalibr.const"]
  --typo in original expression now corrected (2018-03-17)

End If

PlaceFormulae FinalTerm1, plSpotOutputRw, piaSacol[DauParNum]

--Now calculate common Pb-corrected calibration constant ERRORS:

If piStdCorrType = 0 -- '204-corrected'

  If IsPbTh = TRUE

    Term1 = ["UncorrPb/Thconst %err"]^2 +
      ( sComm84 / ( ["208/206"] / ["204/206"] - sComm84 ) )^2
          * ["204/206 %err"]^2
    FinalTerm2 = sqrt( Term1 )
    --fills column ["4-corr208Pb/232Thcalibr.const %err"]

  Else
```

```
        Term1 = ["UncorrPb/Uconst %err"]^2 +
          ( sComm64 / ( 1 / ["204/206"] - sComm64 ) )^2 * ["204/206 %err"]^2
        FinalTerm2 = sqrt( Term1 )
        --fills column ["4-corr206Pb/238Ucalibr.const %err"]

      End If

    Elseif piStdCorrType = 1 -- '207-corrected'

      If IsPbTh = TRUE

        Term1 = ["UncorrPb/Thconst %err"]^2 +
          ( sComm84 / ( ["208/206"] / ["204/206(fr. 207)"] - sComm84 ) )^2 *
          ( ["208/206 %err"]^2 + ["204/206(fr. 207) %err"]^2 )
        FinalTerm2 = sqrt( Term1 )
        --fills column ["7-corr208Pb/232Thcalibr.const %err"]

      Else

        Term1 = ["UncorrPb/Uconst %err"]^2 +
          ( sComm64 / (1 / ["204/206(fr. 207)"] - sComm64 ) )^2 *
          ["204/206(fr. 207) %err"]^2
        FinalTerm2 = sqrt( Term1 )
        --fills column ["7-corr206Pb/238Ucalibr.const %err"]

      End If

    Elseif piStdCorrType = 2 -- '208-corrected', can't use Pb/Th calibration here
    --NOTE that it is also necessary for calculated 232Th/238U and its %err
        to exist!
    --Eqns for Term1 and FinalTerm2 corrected 2017-12-31
    --Eqn for Term1 replaced with Term2, and FinalTerm2 RE-corrected 2018-03-19

      Term2 = ( sComm64 * ["UncorrPb/Uconst"] * ["204/206(fr. 208)"]
                / ["8-corr206Pb/238Ucalibr.const"] )^2
      Term3 = ( ["208/206"] * ["208/206 %err"] /
        ( ["208/206"] - StdRad86fact * ["232Th/238U"] ) )^2
      Term4 = 1 / ( ["208/206"] - StdRad86fact * ["232Th/238U"] ) +
        sComm64 / ( sComm84 - sComm64 * StdRad86fact * ["232Th/238U"] )
      Term6 = ( Term4 * StdRad86fact * ["232Th/238U"] * ["232Th/238U %err"] )^2

      FinalTerm2 = sqrt( ["UncorrPb/Uconst %err"]^2 + (Term2 * (Term3 + Term6)) )
      --fills column ["8-corr206Pb/238Ucalibr.const %err"]

    End If

    If FinalTerm2 <> "" --FinalTerm2 is placed in the "calib.const. %err" column

      PlaceFormulae FinalTerm2, plSpotOutputRw, piaSaEcol[DauParNum]

    End If

  Next DauParNum

End If --end of Ludwig Q4
```

```
    plHdrRw = flHeaderRow[pbStd]
    --just returns the cursor to the top row of StandardData data, I think

  Loop Until piaSpotIndx[-pbStd] = piaEndSpotIndx[-pbStd] --End of Loop D
```

Loop A still remains unfinished, but now that the second loop through the spot-groups has been completed (via Loop D and subsequent), it is time to do the rest of the simple row-by-row calculations, followed by preparations for calculation of the key "column mean" values, which are in turn used in other row-by-row expressions. The remainder of Loop A, encompassing these processes, is documented in Sq2.50 Procedural Framework: Part 4.

SQUID 2.50 Sub: StdElePpm

The subroutine evaluates, on an analysis-by-analysis basis, the concentration of the 'primary' parent element (usually U, but can be Th, and it is also possible for the user to specify 'No U or Th concentration', whereupon this subroutine is exited).

Usage

StdElePpm(Std, SpotRow)

Mandatory variables

Std: Boolean input which dictates, for each specific invocation of StdElePpm, whether the analysis is that of a 'primary-ratio' reference material, or not.

SpotRow: Index number of the output-row, to which the value calculated by StdElePpm is to be written.

Definition of variables

Values of type Boolean
Std

Values of type Integer
c, p, SpotRow

Values of type Double
v, pdMeanParentEleA, pdConcStdPpm

Vectors comprising values of type Double
BkrdCPS, count_time_sec, NetCps, TotCps

Arrays comprising values of type Double
AbsNetPkCps, NetPkCps, PkCounts, PkCps, PkFerr, SBMCounts, SBMCps, time_stamp_sec, TrimMass, TrimTime

Arrays comprising values of type String
psaSpotNames, psaSpotDateTime

The function of the subroutine is paraphrased as follows:

```
--Recalling that Std = -1 if TRUE, 0 if FALSE, use
--p and c to define sheet/column for output
p = -Std

If pbUconcstd = TRUE --if user-defined primary element is U
  c = piaPpmUcol[p]
Else
  c = piaPpmThcol[p]
End If

--Note that if user selected "No U or Th Std", then c = 0:
If c = 0
  Exit Sub
End If
```

If the code reaches this point, then the arithmetic is performed by reference to pdMeanParentEleA, which was calculated in Sq2.50 Procedure Pt 1. Note that piLwrIndx simply refers to the EqNum with the lowest numeric value i.e. piLwrIndx = - 4 (corresponding to the expression for the 'primary parent' concentration constant) for SQUID 2.50 Tasks for U-Th-Pb geochronology. *(Ludwig uses the variable piLwrIndx because SQUID 2.50 allows 'non-geochronology' Tasks, which are characterised by the absence of all 'built-in' expressions; in those Tasks (which are outside our scope), all user-defined expressions are 'custom' and piLwrIndx = 1)*. The code proceeds:

```
If pdMeanParentEleA > 0 AND pdMeanParentEleA <> SQUID_Error_Value
  --define the address for the output:
  piSpotOutputCol = piaEqCol[Std, piLwrIndx]

  EqnInterp Task.Eqns[piLwrIndx], piLwrIndx, v, 0, 1, 0
  --Note that Calamari already does this arithmetic, without context

  If v = SQUID_Error_Value
    Exit Sub
  End If

  --pdConcStdPpm is the reference/model ppm value for the primary element,
  --in the concentration reference material. In our demo XML, that material
  --has the prefix 'M257', and its model U value is 840 ppm.

  v = v / pdMeanParentEleA * pdConcStdPpm
End If

If v > 0
  --CFs fills the target cell
  --fsS converts double-precision number to string (not sure why)
  CFs SpotRow, c, fsS(v)
```

```
End If

End Sub
```

SQUID 2.50 Sub: ThUfromFormula

The subroutine evaluates, on an analysis-by-analysis basis, the 232Th/238U, using the "built-in" expression for 232Th/238U specified by Task EqNum = -3.

Usage

ThUfromFormula(Std, SpotRow)

Mandatory variables

Std: Boolean input which dictates, for each specific invocation of ThUfromFormula, whether the analysis is that of a 'primary-ratio' reference material, or not.

SpotRow: Index number of the output-row, to which the value calculated by ThUfromFormula is to be written.

Definition of variables

Values of type Boolean
Std

Values of type Integer
c, p, SpotRow

Values of type Double
MeanV, MeanVferr

The function of the subroutine is paraphrased as follows:

```
--Recalling that Std = -1 if TRUE, 0 if FALSE, use
--p and c to define sheet/column for output, and then
p = -Std
c = piaTh2U8col[p]
piSpotOutputCol = piaEqCol[Std, -3]
```

The code proceeds, apparently without any error-check on the calculated value of MeanV:

```
EqnInterp Task.Eqns[-3], -3, MeanV, MeanVferr, 1, 1
--Note that Calamari already does this arithmetic, without context
--CFs fills the target cell
--fsS converts double-precision number to string (not sure why)
CFs SpotRow, c, fsS(MeanV)
```

There is, however, an error-check on MeanVferr, as follows:

```
If (Std = FALSE OR piaTh2U8ecol[1] > 0) AND
  (MeanV <> SQUID_Error_Value) AND
  (MeanVferr <> SQUID_Error_Value)

  --CFs fills the target cell
  --fsS converts double-precision number to string (not sure why)
  CFs SpotRow, c + 1, fsS(100 * MeanVferr)

End If

End Sub
```

SQUID 2.50 Sub: SecondaryParentPpmFromThU

The subroutine evaluates, on an analysis-by-analysis basis, the 'secondary parent' (Th or U) ppm concentration, based on a previously-calculated 'primary parent' (U or Th) ppm value and a previously-calculated 232Th/238U value. (U-Pb geochron Tasks only.)

Note that unlike other SQUID 2.50 expressions implemented thus far, SQUID 2.50 constructs this result as an 'Excel equation' that is placed in the target cell via PlaceFormulae. In Excel, this obviously means the output updates whenever any of the inputs update. Given that SQUID 3.0 won't have these sorts of 'moving parts', perhaps it's best if I simply specify the expression constructed by PlaceFormulae.

Usage

SecondaryParentPpmFromThU(Std, SpotRow)

Mandatory variables

Std: Boolean input which dictates, for each specific invocation of SecondaryParentPpmFromThU, whether the analysis is that of a 'primary-ratio' reference material, or not.

SpotRow: Index number of the output-row, to which the value calculated by SecondaryParentPpmFromThU is to be written.

Definition of variables

Values of type Boolean
Std

Values of type Integer
c, p, SpotRow

Values of type String
t1

The function of the subroutine is paraphrased as follows:

```
--Recalling that Std = -1 if TRUE, 0 if FALSE, use
--p and c to define sheet/column for output, and then
p = -Std

If pbUconcStd = TRUE --i.e. concentration material constrains U(ppm)
  c = piaPpmThcol[p] --i.e. 'secondary element' column
Elseif pbThconcStd = TRUE --i.e concentration material constrains Th(ppm)
  c = piaPpmUcol[p] --i.e. 'secondary element' column
End If
```

Note that the above expression from c is constructed so that c is only assigned if pbUconcStd = TRUE OR pbThconcStd = TRUE. In SQUID 2.50, it is possible to specify (via checkbox) "No U or Th concentration standard", which presumably sets both pbUconcStd = FALSE AND pbThconcStd = FALSE, and which in this situation would presumably ensure that c is not calculated. And in fact, the subroutine only continues if the three relevant columns have been satisfactorily identified:

```
If piaTh2U8col[p] > 0 AND piaPpmUcol[p] > 0 AND piaPpmThcol[p] > 0
--i.e. column-indices for 232Th/238U AND Uppm AND Thppm (respectively)
--are non-zero (i.e. they exist), then define string t1:

  If pbUconcStd = TRUE
    t1 = ["ppmU"] * ["232Th/238U"] / 1.033
  ElseIf pbThconcStd = TRUE
    t1 = ["ppmTh"] / ["232Th/238U"] * 1.033
  End If
  --see notes about 'magic number' 1.033 below

  PlaceFormulae "= t1", SpotRow, c
  --subroutine placing formula-string in cell defined by
  --row SpotRow and column c
End If

End Sub
```

The placed formula contains a mysterious 'magic number' 1.033, which will need to be retained in the initial instance, for the purpose of SQUID 2.50 vs SQUID 3.0 comparisons. But I have done some work unpacking it, and it appears to be the product of (1) the ratio of the masses of the U and Th 'isotopes of interest' (i.e. 238 and 232 respectively), and (2) the abundance of 238U as a proportion of 'total' U(ppm), which in turn reflects a 'constant' value for present-day 238U/235U of 137.88 (e.g. Steiger & Jaeger, 1977). Basically:

```
1.033 = (238/232) * [Present 238U/235U] / ([Present 238U/235U] - 1)
--where [Present 238U/235U] = 137.88.
```

The snag is that no record survives regarding **how** Steiger & Jaeger (1977) arrived at 137.88, and more recent triple-spiked ID-TIMS measurements (Hiess et al., 2012) instead indicate a value near 137.82. The main point is that the user should be able to specify their preferred value of [Present 238U/235U], and therefore that 'magic numbers' like 1.033 should not be hard-wired into the production version of SQUID: the above expansion should be used instead.

SQUID 2.50 Sub: Tot68_82_fromA

The subroutine evaluates, on an analysis-by-analysis basis (and for SAMPLE spots only), the total (calibrated) values for **both** 206Pb/238U and 208Pb/232Th, using the values for **both** 206Pb/238U calibration constant and 208Pb/232Th calibration constant, evaluated using the "built-in" expressions specified by Task EqNum = -1 and EqNum = -2.

Note that this subroutine is not traversed unless Switch.DirectAltPD = TRUE and piNumDauPar = 2 (i.e. it applies to Sample analyses in Perm2 and Perm4 only)

Usage

Tot68_82_fromA(SpotRow)

Mandatory variables

SpotRow: Index number of the output-row, to which the value calculated by Tot68_82_fromA is to be written.

Definition of variables

Values of type String
t1, t2, t3

Values of type Integer
DpNum, m, SpotRow, w

The function of the subroutine is paraphrased as follows (remembering that piNumDauPar is always 2 by the time we reach this point: it has been specified by the enclosing If statement (within 'Ludwig Q3' of Squid2.50 Procedural Framework: Part 3):

```
For DpNum = 1 To piNumDauPar

  If (pbU = TRUE AND DpNum = 1) OR (pbTh = TRUE AND DpNum = 2)
    w = 1
    m = piPb6U8_totCol --i.e. write to column ["Total 206Pb/238U"]
  Else
    w = 2
    m = piPb8Th2_totCol --i.e. write to column ["Total 208Pb/232Th"]
```

```
    End If
```

There is a slight complication here, arising from the more restricted function of SQUID 2.50 relative to that envisaged/ implemented for SQUID 3.0. You will recall that for SQUID 3.0, I requested **all** the permissible index-isotope ratio-combinations to be calculated for **each** permutation (e.g. ["206Pb/238U calibration constant"] in Perm1 can have 204corr-, 207corr-, and 208corr- variants; whereas the range of calibration constants calculated in Perm2, Perm3 and Perm4 can only have 204corr- and 207corr- variants: see the set of Cases in the Intro of the Procedural Framework Part 3).

The issue is that SQUID 2.50 calculates only *one* (pre-specified by 'index isotope') variant of each calibration constant, so there is never any ambiguity when performing calculations on those values, and using the outputs of those calculations in other expressions downstream. Obviously, such ambiguity **does** exist in our SQUID 3.0 implementation, so we need to resolve it (at least in a preliminary way) via a similar user-specified control (which would live on the same screen as the controls for SBM-normalisation and SpotAvg vs LinReg) called 'Preferred index isotope', with options 204Pb, 207Pb, 208Pb (strictly, the availability of 208Pb on this list should hinge on whether the selected Task is Perm1-type or not; something to look at later). Down the track, it would be good if there was scope for the user to switch 'preferred index isotope' after the StandardData and SampleData sheets are produced, but we can talk about that later too.

This is only a minor change - I still want you to "calculate everything" as you have been doing. The aim of the control is solely to specify which specific permutation should be propagated to downstream calculations, in a bid to minimise the number of pointless permutations we generate downstream.

The (paraphrased) code continues:

```
If w = 1 --then we are dealing with the 206Pb/238U system, and
  --t1 and t2 refer to calculations that use 'preferred' index
  --isotope X to generate ["X-corr206Pb/238Ucalibr.const"]:

  t1 = StandardData!WtdMeanA1 --calculated by Sub WtdMeanAcalc
  t2 = StandardData!ExtPerr1 --calculated by Sub WtdMeanAcalc

  --t3 has NO dependence on index isotope; it is an intrinsic property
  --of the 206Pb/238U reference material (see Intro to Part 3):

  t3 = StandardData!StdUPbRatio

  --Now fill column ["Total 206Pb/238U"] i.e. the VALUE column:

  PlaceFormulae "=["UncorrPb/Uconst"] / t1 * t3", SpotRow, m
```

```
      --Now fill column ["Total 206Pb/238U %err"] i.e. the UNCERTAINTY
      --in the adjacent column to the right (i.e. m + 1):

      PlaceFormulae "=SQRT( ["UncorrPb/Uconst %err"]^2 + (t2)^2 )", SpotRow, m + 1

    Else --we are dealing with the 208Pb/232Th system, and
      --t1 and t2 refer to calculations that use 'preferred' index
      --isotope X to generate ["X-corr208Pb/232Thcalibr.const"]:

      t1 = StandardData!WtdMeanA2 --calculated by Sub WtdMeanAcalc
      t2 = StandardData!ExtPerr2 --calculated by Sub WtdMeanAcalc

      --t3 has NO dependence on index isotope; it is an intrinsic property
      --of the 208Pb/232Th reference material (see Intro to Part 3):

      t3 = StandardData!StdThPbRatio

      --Now fill column ["Total 208Pb/232Th"] i.e. the VALUE column:

      PlaceFormulae "=["UncorrPb/Thconst"] / t1 * t3", SpotRow, m

      --Now fill column ["Total 208Pb/232Th %err"] i.e. the UNCERTAINTY
      --in the adjacent column to the right (i.e. m + 1):

      PlaceFormulae "=SQRT( ["UncorrPb/Thconst %err"]^2 + (t2)^2 )", SpotRow, m + 1

    End If

  Next DpNum

  End Sub
```

SQUID 2.50 Sub: ThUfromA1A2

The subroutine evaluates, on an analysis-by-analysis basis, the 232Th/238U, by ratioing the 206Pb/238U calibration constant and the 208Pb/232Th calibration constant. Note that this subroutine only applies when Switch.DirectAltPD = TRUE, and piNumDauPar = 2 (i.e. it is essential that calibration constants for both U-Pb and Th-Pb systems have been calculated: Perm2 and Perm4).

Usage

ThUfromA1A2(Std, SpotRow, Only1)

Mandatory variables

Std: Boolean input which dictates, for each specific invocation of ThUfromA1A2, whether the analysis is that of a 'primary-ratio' reference material, or not.

SpotRow: Index number of the output-row, to which the value calculated by ThUfromFormula is to be written.

Optional variables

Only1: Boolean input which dictates, for each specific invocation of ThUfromA1A2, whether the 232Th/238U value column is to be populated with the dummy value '1', or not. **DEFAULT** value = **FALSE**.

According to Ludwig, Only1 = TRUE happens when "the two WtdMeanA formulae and values have not yet been calculated, [it is necessary to] pass first time & put dummy values of 1. Put in real formulae later." With context from both invocations, it might be possible to rework this later.

Definition of variables

Values of type Boolean
Std, Only1

Values of type Integer
p, SpotRow, Th2U8col, Th2U8ecol

Values of type String
Exp232, Exp238, t1, t2

The function of the subroutine is paraphrased as follows:

```
--Recalling that Std = -1 if TRUE, 0 if FALSE, use p to define
--sheet/column for output. Note also t2 is explicitly cleared:

p = -Std
Th2U8col = piaTh2U8col[p]
Th2U8ecol = piaTh2U8ecol[p]

t2 = ""
```

As per subroutine Tot68_82_fromA, there is a slight complication here, arising from the more restricted function of SQUID 2.50 relative to that envisaged/implemented in SQUID 3.0. You will recall that for SQUID 3.0, I requested **all** the permissible index-isotope ratio-combinations to be calculated for **each** permutation (e.g. the pairs of calibration constants calculated in Perm2 and Perm4 can have 204corr- and 207corr- variants: see the set of Cases in the Intro of the Procedural Framework Part 3).

The issue is that SQUID 2.50 calculates only **one** (pre-specified by 'index isotope') variant of each calibration constant, so there is never any ambiguity when performing calculations on those values, and using the outputs of those calculations in other expressions downstream. Obviously, such ambiguity **does** exist in our SQUID 3.0 implementation, so we need to resolve it (at least in a preliminary way) via a similar user-specified control (which would live on the same screen as the controls for SBM-normalisation and SpotAvg vs LinReg) called 'Preferred index isotope', with options 204Pb, 207Pb, 208Pb (strictly, the availability of 208Pb on this list should hinge on whether the selected Task is Perm1-type or not; something to look at later). Down the track, it would be good if there was scope for the user to switch 'preferred index isotope' after the StandardData and SampleData sheets are produced, but we can talk about that later too.

This is only a minor change - I still want you to "calculate everything" as you have been doing. The aim of the control is solely to specify which specific permutation should be propagated to downstream calculations, in a bid to minimise the number of pointless permutations we generate downstream.

The (paraphrased) code continues:

```
If Only1 = TRUE
```

```
    t1 = "1" --note that this is a string (in VBA), not a number

  ElseIf Th2U8col > 0 --i.e. a destination exists for 232Th/238U values

    If (Std = TRUE) AND (piStdRad86col > 0) --the second part means that if
    --'preferred index isotope' = 204Pb, then there exists a column named
    --["4-corr208Pb*/206Pb*"], or alternatively, if 'preferred index isotope'
    --= 207Pb, then there exists a column named ["7-corr208Pb*/206Pb*"]. In
    --SQUID 3.0, I think piStdRad86col > 0 can simply be assumed TRUE.

    --Recall that Lambda238Ma and Lambda232Ma are the decay constants of 238U
    --and 232Th respectively, expressed in units of "Ma^-1" i.e. 1.55125e-04
    --and 4.9475e-05 respectively.

      If 'preferred index isotope' = 204Pb
        Exp238 = "( EXP ( Lambda238Ma * ["4-corr206Pb/238UAge(Ma)"] ) - 1 )"
        Exp232 = "( EXP ( Lambda232Ma * ["4-corr208Pb/232ThAge(Ma)"] ) - 1 )"
        t1 = "=["4-corr208Pb*/206Pb*"] * Exp238 / Exp232"
      Else --'preferred index isotope' = 207Pb
        Exp238 = "( EXP ( Lambda238Ma * ["7-corr206Pb/238UAge(Ma)"] ) - 1 )"
        Exp232 = "( EXP ( Lambda232Ma * ["7-corr208Pb/232ThAge(Ma)"] ) - 1 )"
        t1 = "=["7-corr208Pb*/206Pb*"] * Exp238 / Exp232"
      End If

    ElseIf Std = FALSE --sample spots only; use uncorrected values

      t1 = ["208/206"] * ["Total206Pb/238U"] / ["Total208Pb/232Th"]

    End If --(Std = TRUE) AND (piStdRad86col > 0)
```

Now evaluate the associated uncertainty by quadratic addition. The unfinished ElseIf (Th2U8col > 0) continues:

```
    If Th2U8ecol > 0 --i.e. a destination exists for 232Th/238U uncertainties

      If Std = TRUE

        If 'preferred index isotope' = 204Pb
          t2 = '=SQRT( ["4-corr208Pb*/206Pb* %err"]^2 +
                       ["4-corr206Pb/238Ucalibr.const %err"]^2 +
                       ["4-corr208Pb/232Thcalibr.const %err"]^2 )
        Else --'preferred index isotope' = 207Pb
          t2 = '=SQRT( ["7-corr208Pb*/206Pb* %err"]^2 +
                       ["7-corr206Pb/238Ucalibr.const %err"]^2 +
                       ["7-corr208Pb/232Thcalibr.const %err"]^2 )
        End If

      Else --i.e. sample spots; Ludwig wrote
      --"Elseif 'preferred index isotope' is not 208Pb", but this seems redundant:
      --entry into ThUfromA1A2 precludes 208Pb as preferred index isotope
```

```
        t2 = '=SQRT( ["208/206 %err"]^2 + ["Total206Pb/238U" %err]^2 +
                     ["Total208Pb/232Th" %err]^2 )

      End If  --(Std = TRUE)

    End If  --(Th2U8ecol > 0)

  End If  --(ElseIf Th2U8col > 0)
```

Finally, place the analysis-by-analysis formulae according to the expression-strings developed above. First, the value:

```
PlaceFormulae t1, SpotRow, Th2U8col
```

And secondly, the uncertainty (if the expression has been developed):

```
If t2 <> ""
  PlaceFormulae t2, SpotRow, Th2U8ecol
End If

End Sub
```

SQUID 2.50 Sub: OverCtMeans

This subroutine (which is solely **for the Standard**) does a bit more than the name implies. Firstly it places, row-by-row, formulae to calculate the 204-corrected 207Pb/206Pb **ratio** and its 1sigma percentage uncertainty, as well as invoking (row-by-row) the relevant LudwigLibrary functions to calculate the associated 204-corrected 207Pb/206Pb **date** and its 1sigma absolute uncertainty.

Secondly, it identifies which columns can usefully have robust means calculated, performs those calculations (using LudwigLibrary function TukeysBiweight, with tuning 9) and places the output of the expression as a 3 x 1 array beneath the relevant column. The SQUID 2.50 subroutine requires the index number of the last row of analytical data as an input, so it can determine in which rows the "summary" results should be placed so that the calculated biweights appear directly beneath the input data.

Usage

OverCtMeans plaLastDatRw

Mandatory variable

plaLastDatRw: Integer index number of the last row containing spot-by-spot data (for the Standard).

Definition of variables

Values of type Boolean
ShowOverCtCols

Values of type Integer
c, k, kk, plaFirstDatRw

Values of type String
t0, t1, t2, t3, t4, t5

The subroutine starts by calculating the 204Pb-corrected 207Pb/206Pb, its uncertainty, and the associated date and uncertainty.

```
If piaAgePb76_4Col[1] > 0 --i.e. if column ["4-corr207Pb/206Pb"] exists on Standard

  t0 = "=( ["207/206"] / ["204/206"] - sComm_74 ) / ( 1 / ["204/206"] - sComm_64 )"

  --Then place this formula in column ["4-corr207Pb/206Pb"] on StandardData:
  PlaceFormulae t0, plaFirstDatRw, piStdPb76_4Col, plaLastDatRw

  --Now calculate the associated %err:
  t1 = "( ( ["207/206"] * ["207/206%err"] )^2 +
       ( ["204/206"] * ( ["4-corr207Pb/206Pb"] * sComm_64 - sComm_74)
          * ["204/206%err"] )^2 )"
  t2 = "( ["207/206"] - ["204/206"] * sComm_74 )^2"
  t3 = "=sqrt( t1 / t2 )"

  --Then place this formula in column ["4-corr207Pb/206Pb%err"] on StandardData:
  PlaceFormulae t3, plaFirstDatRw, piStdPb76_4eCol, plaLastDatRw

  --Now invoke LudwigLibrary functions to calculate associated age and error:
  t4 = "=AgePb76( ["4-corr207Pb/206Pb"] )"

  --Then place this formula in column ["4-corr207Pb/206Pbage"] on StandardData:
  PlaceFormulae t4, plaFirstDatRw, piaAgePb76_4Col, plaLastDatRw

  --Convert ["4-corr207Pb/206Pb%err"] to absolute, to calculate age error:
  t5 = "=AgeErPb76( ["4-corr207Pb/206Pb"] , ["4-corr207Pb/206Pb"]
      * ["4-corr207Pb/206Pb%err"] / 100 )"

  --Then place this formula in column ["4-corr207Pb/206Pbage±1sigma"] on StandardDa
  PlaceFormulae t5, plaFirstDatRw, piaAgePb76_4eCol, plaLastDatRw

End If
```

The second part of the subroutine calculates the various biweight means, for the range of columns for which they are relevant. In practice, the "entry" Boolean is always TRUE, because Ludwig set the nominally user-defined Boolean ShowOverCtCols to TRUE, and then removed the relevant check-box from the user form! Note that a possibly unintended consequence of "locking in" this Boolean is that it formally requires *all* SQUID 2.50 U-Pb Geochronology Tasks to include 204Pb in the list of mass-stations.

__(There will come a time when it is necessary for us to formally assess the absolute minimum mass-stations in order to perform U-Pb (or Th-Pb) geochronology in SQUID 3.0. My feeling is that the absolute minimum lists are as follows:

- 206Pb/238U: 206Pb, 238U (or a proxy thereof), and ONE OF (204Pb or 207Pb)
- 208Pb/232Th: 208Pb, 232Th (or a proxy thereof), and ONE OF (204Pb or 207Pb)

At present, SQUID 2.50 enforces "BOTH OF". Certainly the "alternate" daughter-isotope (i.e.

208Pb for U-Pb, 206Pb for Th-Pb) should be optional, and probably Background should be optional too (even though including 204Pb in a run-table without including Background would be terrible practice. Something to revisit later, and the stakes are relatively low: it doesn't really matter exactly what the 'bare bones' list of mass-stations is, as long as we warn SQUID 3.0 users up-front… it would be an improvement on SQUID 2.50!)__

The subroutine proceeds:

```
If ShowOverCtCols = TRUE OR piaAgePb76_4Col[1] > 0 --i.e. if StandardData
  contains ["4-corr207Pb/206Pbage"]

  If piaOverCts4Col[7] > 0 Or piaOverCts4Col[8] > 0 Or piaAgePb76_4Col[1] > 0
  --i.e. if ANY of the columns ["204overcts/sec(fr. 207)"],
      ["204overcts/sec(fr. 208)"],
  --["4-corr207Pb/206Pbage"] exist on the StandardData sheet, then define
      the extent of calculations
  --to be performed:

    If ShowOverCtCols = TRUE
      kk = 1
    Else
      kk = 5
    End If

    For k = kk To 5

      Select Case k --all column-indices refer to StandardData sheet:
        Case 1: c = piaOverCts4Col[7] --index for ["204overcts/sec(fr. 207)"]
        Case 2: c = piaOverCts4Col[8] --index for ["204overcts/sec(fr. 208)"]
        Case 3: c = piacorrAdeltCol[7] --index for ["7-corr206Pb/238Uconst.delta%"]
        Case 4: c = piacorrAdeltCol[8] --index for ["8-corr206Pb/238Uconst.delta%"]
        Case 5: c = piaAgePb76_4Col[1] --index for ["4-corr207Pb/206Pbage"]
      End Select

      If c > 0

        --Define Range bw as 3 x 1, with the first row immediately following
            analytical data:
        Set bw = frSr(1 + plaLastDatRw, c, 3 + plaLastDatRw) --sets Range
            bw = 3 rows by 1 col

        --Invoke LudwigLibrary function Biweight (tuning 9) on the data in
            the column above bw.
        --Note that for k = 3 or 4, the values in this range have not yet
            been calculated!
        bw.FormulaArray = "=Biweight(" & frSr(plaFirstDatRw, c, plaLastDatRw).Addre

        --Finally, add Names (and explanatory Notes) to result-cells. These are
            relevant because
        --the Names can be used in Task-expressions, and the Notes matter
            because the columns are
        --quite abstract concepts for non-SHRIMP geochronologists (as well
```

```
      as SHRIMP beginners):

--Note that I have recast the following If to make it longer but more trans

If k = 1
  t1 = "StandardData!Pb204OverCts7corr"
  AddName t1, TRUE, 1 + plaLastDatRw, c --Name for the
      Biweight Value element
  t2 = "StandardData!Pb204OverCts7corrEr"
  AddName t2, TRUE, 3 + plaLastDatRw, c --Name for the
      Biweight 95% conf. element
  t3 = "Robust avg 204 overcts assuming 206Pb/238U-207Pb/235U age
      concordance"
ElseIf k = 2
  t1 = "StandardData!Pb204OverCts8corr"
  AddName t1, TRUE, 1 + plaLastDatRw, c --Name for the Biweight Value
      element
  t2 = "StandardData!Pb204OverCts8corrEr"
  AddName t2, TRUE, 3 + plaLastDatRw, c --Name for the Biweight
      95% conf. element
  t3 = "Robust avg 204 overcts assuming 206Pb/238U-208Pb/232Th age
      concordance"
ElseIf k = 3
  t1 = "OverCtsDeltaP7corr"
  AddName t1, TRUE, 1 + plaLastDatRw, c --Name for the Biweight Value
      element
  t2 = "OverCtsDeltaP7corrEr"
  AddName t2, TRUE, 3 + plaLastDatRw, c --Name for the Biweight
      95% conf. element
  t3 = "Robust avg of diff. between 207-corr. and 204-corr. calibr. const."
ElseIf k = 4
  t1 = "OverCtsDeltaP8corr"
  AddName t1, TRUE, 1 + plaLastDatRw, c --Name for the Biweight Value
      element
  t2 = "OverCtsDeltaP8corrEr"
  AddName t2, TRUE, 3 + plaLastDatRw, c --Name for the Biweight
      95% conf. element
  t3 = "Robust avg of diff. between 208-corr. and 204-corr. calibr. const."
Else --interestingly, SQUID 2.50 does not assign Names (t1, t2) to the
    Biweight
  --elements calculated for the ["4-corr207Pb/206Pbage"] column.
      Probably an oversight!
  t3 = "Robust average of 204-corrected 207/206 age"
End If --k = 1 to 5

--Now VBA function "Note" to add text t3 as Comment on the Biweight
    Value element:
Note 1 + plaLastDatRw, c, t3

--Corresponding Comment on the Biweight 95% conf. element is
    context-sensitive:
If k = 3 OR k = 4
  t3 = "95%-conf. error in above difference"
Else
  t3 = "95%-conf. error in above"
End If
```

```
        Note 3 + plaLastDatRw, c, t3

      End If --c > 0

    Next k

  End If --piaOverCts4Col[7] > 0 Or piaOverCts4Col[8] > 0 Or
      piaAgePb76_4Col[1] > 0

  ClearObj bw

End If --ShowOverCtCols = TRUE OR piaAgePb76_4Col[1] > 0

End Sub
```

Sq2.50 Procedure Pt 4

The next phase of Loop A does many of the remaining simple row-by-row calculations, followed by preparations for calculation of the key "column mean" values, which are in turn used in other row-by-row expressions. The incomplete Loop A proceeds as follows:

```
If pbStd = FALSE --Calculate a couple of "SampleData-only" columns:

  --Firstly ["7-corr204Pb/206Pb"]:
  Term1 = Pb46cor7( ["207/206"], sComm64, sComm74, ["207corr206Pb/238UAge"])
  --recalling that subroutine "Pb46cor7" is defined in LudwigLibrary
  PlaceFormulae Term1, Frw, piPb46_7col, Lrw

  --Secondly ["8-corr204Pb/206Pb"]:
  Term1 = Pb46cor8( ["208/206"], ["232Th/238U"], sComm64, sComm84,
    ["208corr206Pb/238UAge"] )
  --recalling that subroutine "Pb46cor8" is defined in LudwigLibrary
  PlaceFormulae Term1, Frw, piPb46_8col, Lrw

End If
```

The next step is to calculate ALL of the applicable **proportions** of common Pb (which in turn reflect all the permutations of index isotope (204-, 207-, and 208-corrected) and daughter-Pb isotope (206Pb and 208Pb, for Pb/U and Pb/Th respectively), firstly for the StandardData sheet and secondly for the SampleData sheet:

```
If pbStd = TRUE --StandardData first

  --First, fill column ["4-corr%com206"]:
  Term2 = 100 * sComm64 * ["204/206"]
  PlaceFormulae Term2, Frw, piStdCom6_4col, Lrw

  --Second, fill column ["7-corr%com206"]:
  Term2 = 100 * sComm64 * ["204/206 fr.207"]
  PlaceFormulae Term2, Frw, piStdCom6_7col, Lrw

  --Third, fill column ["8-corr%com206"]:
  Term2 = 100 * sComm64 * ["204/206 fr.208"]
  PlaceFormulae Term2, Frw, piStdCom6_8col, Lrw

  --Fourth, fill column ["4-corr%com208"]:
  Term2 = 100 * sComm84 / ["208/206"] * ["204/206"]
  PlaceFormulae Term2, Frw, piStdCom8_4col, Lrw

  --Fifth, fill column ["7-corr%com208"]:
  Term2 = 100 * sComm84 / ["208/206"] * ["204/206 fr.207"]
  PlaceFormulae Term2, Frw, piStdCom8_7col, Lrw

Else --i.e. SampleData second
```

```
  --First, fill column ["4-corr%com206"], same as StandardData:
  Term2 = 100 * sComm64 * ["204/206"]
  PlaceFormulae Term2, Frw, piStdCom6_4col, Lrw

  --Second, fill column ["7-corr%com206"], different to StandardData:
  Term2 = 100 * sComm64 * ["7-corr204Pb/206Pb"]
  PlaceFormulae Term2, Frw, piStdCom6_7col, Lrw

  --Third, fill column ["8-corr%com206"], different to StandardData:
  Term2 = 100 * sComm64 * ["8-corr204Pb/206Pb"]
  PlaceFormulae Term2, Frw, piStdCom6_8col, Lrw

  --Fourth, fill column ["4-corr%com208"], same as StandardData:
  Term2 = 100 * sComm84 / ["208/206"] * ["204/206"]
  PlaceFormulae Term2, Frw, piStdCom8_4col, Lrw

End If
```

The next step is to calculate all the applicable **radiogenic** 208Pb/206Pb values. Note that SQUID 2.50 currently calculates **both** possible 208Pb/206Pb values (204-corrected and 207-corrected) for all analyses on the SampleData sheet, but only calculates (at most) **one** radiogenic 208Pb/206Pb value for the StandardData sheet. I don't see why this should be so: I think SQUID 3.0 should calculate both (and therefore that the If criteria below should be modified accordingly):

```
If pbStd = FALSE OR piStdCorrType = 0 --i.e. all SampleData,
--plus StandardData when index isotope selected is 204Pb:

  --First, prepare to fill column ["4-corr208Pb*/206Pb*"]:
  FinalTerm1 = ( ["208/206"] / ["204/206"] - sComm84 ) / ( 1 / ["204/206"]
    - sComm64)

  --Second, assemble the expression for column ["4-corr208Pb*/206Pb* %err"]:
  --Note that the following If now *seems* obsolete, because we are using
      the same
  --column-name ( ["4-corr208Pb*/206Pb*"] ) for 204Pb-corrected 208Pb/206Pb
      in both
  --the StandardData and SampleData sheets (Ludwig did not do this).
  --All expressions for Term5 and FinalTerm2 corrected 2018-03-21:

  If pbStd = TRUE
    Term5 = ( (["208/206 %err"] / 100 * ["208/206"])^2 +
      (["4-corr208Pb*/206Pb*"] * sComm64 - sComm84)^2 *
      (["204/206 %err"] / 100 * ["204/206"])^2 )
      / (1 - sComm64 * ["204/206"])^2
    FinalTerm2 = 100 * sqrt( Term5 ) / abs( ["4-corr208Pb*/206Pb*"] )

    --Now place the FinalTerm1 formula for ["4-corr208Pb*/206Pb*"] and
    --and the FinalTerm2 formula for ["4-corr208Pb*/206Pb* %err"]:
    OutpCol = ["4-corr208Pb*/206Pb*"]
    PlaceFormulae FinalTerm1, Frw, OutpCol, Lrw
```

```
    PlaceFormulae FinalTerm2, Frw, 1 + OutpCol, Lrw

  Else

    Term5 = ( (["208/206 %err"] / 100 * ["208/206"])^2 +
      (["4-corr208Pb*/206Pb*"] * sComm64 - sComm84)^2 *
      (["204/206 %err"] / 100 * ["204/206"])^2 )
      / (1 - sComm64 * ["204/206"])^2
    FinalTerm2 = 100 * sqrt( Term5 ) / abs( ["4-corr208Pb*/206Pb*"] )

    --Now place the FinalTerm1 formula for ["4-corr208Pb*/206Pb*"] and
    --and the FinalTerm2 formula for ["4-corr208Pb*/206Pb* %err"]:
    OutpCol = ["4-corr208Pb*/206Pb*"]
    PlaceFormulae FinalTerm1, Frw, OutpCol, Lrw
    PlaceFormulae FinalTerm2, Frw, 1 + OutpCol, Lrw

  End If

End If

If pbStd = FALSE OR piStdCorrType = 1 --i.e. all SampleData,
--plus StandardData when index isotope selected is 207Pb:

  --First, prepare to fill column ["7-corr208Pb*/206Pb*"]:
  If pbStd = TRUE

    FinalTerm1 = ( ["208/206"] / ["204/206 fr.207"] - sComm84 ) /
      ( 1 / ["204/206 fr.207"] - sComm64)

  Else

    FinalTerm1 = ( ["208/206"] / ["7-corr204Pb/206Pb"] - sComm84 ) /
      ( 1 / ["7-corr204Pb/206Pb"] - sComm64)

  End If

  --Second, assemble the expression for column ["7-corr208Pb*/206Pb* %err"]:
  If pbStd = TRUE
    --assemble 10 arguments for separate subroutine StdPb86radCor7per:
    --the following expression was corrected 2018-03-26:
    Term5 = ["208/206"], ["208/206 %err"], ["207/206"], ["207/206 %err"],
      ["7-corr208Pb*/206Pb*"], ["204/206 fr.207"], Std_76, sComm64, sComm74,
          sComm84
    FinalTerm2 = StdPb86radCor7per( Term5 )
    --subroutine StdPb86radCor7per to be documented separately

    --Now place the FinalTerm1 and 2 formulae for ["7-corr208Pb*/206Pb*"]
        and
    ["7-corr208Pb*/206Pb* %err"] respectively:
    OutpCol = ["7-corr208Pb*/206Pb*"]
    PlaceFormulae FinalTerm1, Frw, OutpCol, Lrw
    PlaceFormulae FinalTerm2, Frw, 1 + OutpCol, Lrw

  Else --and assuming ["Total206Pb/238U"] and ["207corr206Pb/238UAge"]
      both exist
```

```
    --assemble 10 arguments for separate AND DIFFERENT subroutine
    Pb86radCor7per:
    Term5 = ["208/206"],["208/206 %err"], ["207/206"],["207/206 %err"],
    ["Total206Pb/238U"],
      ["Total206Pb/238U %err"],["207corr206Pb/238UAge"], sComm64, sComm74,
      sComm84
    FinalTerm2 = Pb86radCor7per( Term5 )
    --subroutine Pb86radCor7per to be documented separately

    --Now place the FinalTerm1 and 2 formulae for ["7-corr208Pb*/206Pb*"] and
    ["7-corr208Pb*/206Pb* %err"] respectively:
    OutpCol = ["7-corr208Pb*/206Pb*"]
    PlaceFormulae FinalTerm1, Frw, OutpCol, Lrw
    PlaceFormulae FinalTerm2, Frw, 1 + OutpCol, Lrw

  End If

End If
```

The next step is to calculate some column-means, firstly a weighted mean of the common-Pb calibration constants, and then biweights of some of the other columns related to 204-overcounts. The incomplete Loop A proceeds:

```
If pbStd = TRUE

  If piaSpotCt[-pbStd] > 1
  --i.e. if more than one spot-analysis corresponding to specified prefix
      for Standard
  --then SQUID 2.50 assigns the name "StdHrs" to the cell range corresponding
      to column
  --"Hours" in sheet StandardData. Then:

    If pbSBMnorm = TRUE --i.e. if user requested SBM-normalisation
    --then ReDim the following double-precision vectors (not zero-addressed!)\

      ReDim SbmOffs[1 to Task.Npeaks]  --number of mass-stations in Task/XML
          file
      ReDim SbmOffsErr[1 to Task.Npeaks]
      ReDim SbmPk[1 to piaSpotCt[-pbStd] ]  --number of Standard analyses in
          XML file

    End If

    --Now calculate weighted mean calibration constants for Standard
    --(subroutine documented separately)
    WtdMeanAcalc BadSbm(), Adrift(), AdriftErr()

    --Then calculate and place robust means of 204-overcount columns for
        Standard:
    --(subroutine documented separately)
    OverCtMeans plaLastDatRw[-pbStd]
```

```
    Else --i.e. piaSpotCt[-pbStd] <= 1, i.e only one analysis with prefix
        matching that
    --defined for the Standard <<string StdPrefix>>, or none at all.
        Processing must quit,
    --but SQUID 2.50 tries to construct a helpful error-message first:

      If (pbFoundStdName = TRUE) AND (piaSpotCt[-pbStd] = 0)

        Msg = "Unable to parse the raw-data file."

      ElseIf piaSpotCt[-pbStd] = 1

        Msg = "Only 1 spot found with label similar to <<string StdPrefix>>
               - did you correctly specify the Standard name?"

      Else

        Msg = "No spots found with label similar to <<string StdPrefix>>
               - did you correctly specify the Standard name?"

      End If

      Exit Sub --quit processing

    End If

    --Since piaSpotCt[-pbStd] > 1, attend to StandardData SBM display:
    --2018-06-12: THIS SECTION SKIPPED; GRAPHICAL OUTPUT ONLY, ATTEND TO
        IT LATER
    If (pbSbmNorm = TRUE) AND (BadSBM[-pbStd] < piaSpotCt[-pbStd] *
        Task.Npeaks / 2)

      SBMdata SbmOffs(), SbmPk(), SbmOffsErr(), pdaSbmDeltaPcnt(), CanChart
      --subroutine to be documented separately

      If CanChart = TRUE

        AddSBMchart plHdrRw, SbmOffs(), SbmOffsErr()
        --subroutine to be documented separately

      End If --CanChart

    End If --(pbSbmNorm = TRUE, etc.)

  End if --pbStd = TRUE as per beginning of this Part 4.
```

The code then proceeds to the final stretch of the incomplete Loop A that commenced at the beginning of Part 1. That code is documented in Part 5.

SQUID 2.50 Sub: WtdMeanAcalc

This subroutine evaluates, **for the Standard**, the weighted mean (and associated parameters) of each relevant set of common-Pb corrected calibration constant values. This mean is the value to which all spot-by-spot calibration constants determined for the unknowns will be calibrated, and *all* spot-by-spot, 'directly-calculated daughter/parent dates' (i.e. 206Pb/238U and/or 208Pb/232Th as appropriate) are calculated from there, because SHRIMP (and secondary ion mass spectrometry in general) is an **indirect** dating technique. Thus this is a critical step.

Usage

WtdMeanAcalc BadSbm(), Adrift(), AdriftErr()

Optional variables

BadSbm: Array containing index numbers of spots for which 'bad' SBM values exist (e.g values lower than SBMzero).

Adrift: Array of double-precision values arising from 'correction' of the measured calibration-constant values for secular drift associated with the duration of the analytical session (i.e. "Hours"). Currently out of scope.

AdriftErr: Array of double-precision uncertainties associated with values arising from 'correction' of the measured calibration-constant values for secular drift associated with the duration of the analytical session (i.e. "Hours"). Currently out of scope.

Definition of variables

Values of type Boolean
NoReject, NoUPbConstAutoReject, pbCanDriftCorr, pbU, pbTh, [Task]Switch.DirectAltPD

Values of type Integer
c,DauParNum, ErN, h, i, j, LargeErRegN, Npts, Nrej, NtotRej, NumDauPar, r, rw1, rwn

Values of type Double
ExtPerr, ExtPtSigma, Lambda, MedianEr, MSWD, Nmadd, Prob, pscLm2, pscLm8, StdThPbRatio, StdUPbRatio, WtdMean, WtdMeanErr

Values of type String
s, Ele, t1, t2, t4, t5

Vectors of type Integer
LargeErRej, Rejected

Vectors of type Double
Acol, Aecol, Arange, Aerange, ErrVals

Arrays of type Double
Adatrange

Arrays of type Variant/Mixed
wW

The function commences with some setting up. NoUPbConstAutoReject is a user-defined Boolean that dictates whether SQUID 2.50 will be permitted to reject 'aberrant' calibration-constant values (according to its own conventions, and subject to its own rules) in order to calculate the 'best' weighted mean calibration-constant value. In practice, it is useful to allow SQUID 2.50 to identify and exclude its own rejections (i.e. user usually sets NoUPbConstAutoReject = FALSE), just to get an idea of which values it considers aberrant. Once the calculation is complete, the user has the authority to 'unreject' SQUID's outliers, and that option is frequently exercised. After all, the reference material is supposed to be isotopically homogeneous (as usually demonstrated by ID-TIMS data) and if the SHRIMP-derived population of calibration constants displays significant dispersion, then this is likely to be an instrumental effect which must be faithfully propagated to analyses of the unknowns. In some cases, rejections might be allowed to stand if there is independent evidence of (hopefully transient!) sub-optimal analytical conditions, such as instability in the primary beam or secondary beam, or poor focus of the beam on the sample.

pbCanDriftCorr is another user-defined Boolean that (if set to TRUE) instructs SQUID 2.50 to evaluate the spot-by-spot measured values of the calibration constants in terms of when they were acquired (during the course of the analytical session, typically described as "secular drift"). In practice, this can be quite a useful thing to be able to do, but at this stage of SQUID 3.0 development, it should be considered out-of-scope, and a working value of pbCanDriftCorr = FALSE should be assumed.

```
rw1 = plaFirstDatRw[-pbStd] --first row of StandardData
rwn = plaLastDatRw[-pbStd] --last row of StandardData

If NoUPbConstAutoReject = TRUE AND pbCanDriftCorr = FALSE
  NoReject = TRUE
```

```
Else
  NoReject = FALSE
End If
```

As before, catch non-meaningful cases where NumDauPar has been set to 2, without the proper specification of a direct calibration-expression for the 'second' DauPar, via Task.saEqns[-2]:

```
If NumDauPar = 2 AND Switch.DirectAltPD = TRUE AND Task.saEqns[-2] = ""
      --i.e. "null"
  NumDauPar = 1
End If
```

Now a long 'For' loop, which runs once for each calibration-constant mean that needs to be calculated:

```
For DauParNum = 1 to NumDauPar

  Acol = piaSacol[DauParNum] --select appropriate column of calib. constant
      values
  Aecol = piaSaEcol[DauParNum] --appropriate column of calib. constant
      uncertainties

  s = fsS(DauParNum) --string representation of DauParNum, used for
      labelling purposes

  If (pbU = TRUE AND NumDauPar = 1) OR (pbTh = TRUE AND NumDauPar = 2)
    --we are dealing specifically with a 206Pb/238U calib. constant:
    Lambda = pscLm8 --i.e. 238U decay constant in units of "Ma^-1"
    Ele = "U"
  Else --we are dealing specifically with a 208Pb/232Th calib. constant:
    Lambda = pscLm2 --i.e. 232Th decay constant in units of "Ma^-1"
    Ele = "Th"
  End If
```

Next, Name and define some Ranges: SQUID 2.50 uses the Ludwig custom functions "AddName" and "frSr" for this purpose.

"AddName" has 6 arguments (first 4 mandatory, last 2 optional) corresponding to (1) the string Name being defined, (2) a Boolean defining the sheet the Name applies to (TRUE = StandardData, FALSE = SampleData), (3) the address of the top row, (4) the address of the left column, (5) the address of the bottom row, and (6) the address of the right column, for a contiguous cell-range.

"frSr" has only the final 4 arguments of AddName, and can accept Names as input. The For loop continues:

```
AddName "Arr_" & s, TRUE, rw1, Acol, rwn, Acol
--Name is thus "Arr_1" or "Arr_2", for single Acol alone

AddName "Aer_" & s, TRUE, rw1, Aecol, rwn, Aecol
--Name is thus "Aer_1" or "Aer_2", for single Aecol alone

AddName "Adat_" & s, TRUE, rw1, Acol, rwn, Aecol
--Name is thus "Adat_1" or "Adat_2", for both columns

Set Arange = Range("Arr_" & s)
Set AerRange = Range("Aer_" & s)
Set AdatRange = Range ("Adat_" & s)

Npts = Count( AdatRange ) / 2 --seems indirect!
ErN = Count( AerRange )

ReDim ErrVals[1 to ErN] --apparently not zero-addressed
```

Now clean the uncertainty values in the Range, and the ErrVals calculation-input. Then calculate the median (and median absolute deviation) of the ErrVals, as inputs for (potential) filtering of calibration-constant value-uncertainty pairs from the eventual calculation of the weighted mean:

```
For i = 1 to ErN

  If (fbIsNum( AdatRange[i, 2] ) = TRUE) AND (IsEmpty( AdatRange[i, 2] )
        = FALSE)
    --function fbIsNum is in the Ludwig Library
    ErrVals[i] = AdatRange[i, 2]
  Else
    ErrVals[i] = 0
    AdatRange[i, 2] = ""
  End If

Next i

MedianEr = Median( AerRange ) --why clean col2 of AdatRange if not to use it here

GetMAD ErrVals, ErN, MedianEr, 0, 0 --GetMAD is a Ludwig Library function (I hope
--It is not obvious what (if anything) is achieved by this invocation of
--subroutine GetMAD, given that function fdNmad is invoked in the next line:

Nmadd = fdNmad( ErrVals ) --function fdNmad defined separately

LargeErRegN = 0
```

If the user has specified that they would like SQUID to attempt 'auto-rejection' of suspect calibration-constant measurements (i.e. NoReject = FALSE; unfortunately, very many of Ludwig's logic tests are double-negatives!), the next step is to assess the **uncertainties** of the calibration-constants, in a search for anomalous-looking values. Two criteria are used for ErrVal-based rejection:

1) ErrVals residuals (relative to MedianEr) that exceed Nmadd by more than an order of magnitude
2) ErrVals of 0

The unfinished 'For DauParNum…' clause continues:

```
If NoReject = FALSE
  ReDim LargeErRej[1 to 99] --won't have more than 99 rejections!

  For i = 1 to ErN

    If ( ABS( ErrVals[i] - MedianEr > 10 * Nmadd ) OR ( ErrVals[i] = 0 )

      --The following 'For' applies Strikethrough font to both the offending
      --ErrVals[i] value AND the associated calibration-constant *VALUE*
      --in the column immediately to its left. This is significant because
      --when SQUID invokes functions like WtdAv in a spreadsheet environment,
      --rows with Strikethrough are *EXCLUDED* from the calculation.

      For j = 0 to 1
        AerRange[i, j].Font.Strikethrough = TRUE
      Next j

      LargeErRegN = 1 + LargeErRegN

      LargeErRej[LargeErRegN] = i

    End If

  Next i

  If LargeErRegN > 0
    ReDim Preserve LargeErRej[1 to LargeErRegN]
  End If

End If --NoReject = FALSE
```

SQUID 2.50 then nominates a place for the weighted mean result to be places (the two-column output array will appear directly beneath the relevant calibration-constant value-uncertainty pair of columns, with a 3-row gap in between:

```
h = 3 + rwn

AddName "WtdMeanA" & s, TRUE, h, Acol --names a single cell
AddName "WtdMeanAPerr" & s, TRUE, 1 + h, Acol
--names the cell in the row directly BENEATH WtdMeanA
```

Now perform the actual weighted mean calculation (which might involve a detailed review of the operation of subroutine WtdAv). Remember that for the initial implementation of SQUID 3.0, we assign pbCanDriftCorr = FALSE. The unfinished 'For DauParNum…' clause continues:

```
If pbCanDriftCorr = TRUE

  [out-of-scope stuff]

Else

  wW = WtdAv Adatrange, TRUE, TRUE, 1, (NOT NoReject), TRUE, 1
```

The output wW of WtdAv is an array with 2 columns and **up to 7** rows. The left-hand column contains mostly numeric data (values, errors, MSWD, probability of fit, etc.), and the right-hand column contains text labels for the data in the left-hand column. **Remember** that analyses for which Font.Strikethrough was applied (above) are **excluded** from the calculation **in advance**, and so are not assessed as part of WtdAv's "CanReject" provisions - "CanReject" data-points are separate and additional, and the two rejection-sources are amalgamated into a single list of 'rejected points' in the code below.

Before that, however, Ludwig out elements of the array wW and gives them variable-names. Unfortunately, this simple-looking process is complicated by the fact the the output of WtdAv is **conditional**: the third row of the 7 x 2 array does not appear if it is not applicable (which is an awful 'feature'!), so this complicates addressing of the ensuing values in what might be a 6 x 2 array. The unfinished Else continues:

```
  WtdMean = wW[1, 1]
  WtdMeanErr = wW[2, 1]
```

Assigning the remaining values depends on whether the data are so dispersed (MSWD >> 1) that the subroutine WtdAv calculated a 'constant external error' (i.e. a **constant** additional uncertainty which, when added in quadrature to *EACH* of the data-point uncertainties, would yield a weighted mean value with an MSWD of ~1). The unfinished Else continues:

```
If wW[3, 2] = "MSWD" --then data were NOT dispersed, and NO constant
                     --external error was calculated:
  ExtPtSigma = 0
  MSWD = wW[3, 1]
  Prob = wW[5, 1]

Else --data WERE dispersed, a constant external error WAS calculated,
     --and the addresses of subsequent array-elements is different:
  ExtPtSigma = wW[3, 1]
  MSWD = wW[4, 1]
  Prob = wW[6, 1]

End If
```

In SQUID 2.50, when WtdAv is invoked with Boolean input CanReject = TRUE (as is the case here), the final row of the output array is a space-delimited list of the index-numbers of analyses rejected (remembering that, for the invocation being documented here, analyses identified for rejection by the "LargeErRej"-related criteria documented above were **never** assigned an index-number by WtdAv: those analyses were ignored before WtdAv even started applying its "CanReject" criteria).

The SQUID 2.50 code continues:

```
ParseLine wW[7, 1], Rejected(), Nrej, " " --subroutine not worth documenting
```

Subroutine ParseLine is basically a generalised text-editing subroutine, which transforms a delimited string into an array of the values (numeric, string, or mixed) separated by the delimiting character (specified above as a single space " "). In terms of Java implementation, all that is required here is:

1) generation of a vector array (Rejected) whose elements are the analysis index-numbers specified in wW[7, 1]
2) definition of integer Nrej = length(Rejected)

In SQUID 2.50, the upshot of all this is that when after the subroutine WtdAv has been invoked by subroutine WtdMeanAcalc, there exists up to two sets of rejected analyses: (1) those potentially generated by "LargeErRej" provisions above, and (2) those potentially generated by CanReject = TRUE within the WtdAv routine. With respect to the overall set of calibration constant value-uncertainty pairs, we need the union/superset of these two sets. The unfinished Else continues:

```
If LargeErRegN > 0
```

```
    For i = 1 To Nrej
      For j = 1 To LargeErRegN

        --If LargeErRej[j] < Rejected[i] --Ludwig original, which is incorrect.
        --The following line (Bodorkos 2018-02-25) is the correct replacement:

        If LargeErRej[j] < Rejected[i] OR LargeErRej[j] = Rejected[i]
          Rejected[i] = 1 + Rejected[i]
        End If

      Next j
    Next i

  End If --LargeErRegN > 0

  NtotRej = Nrej + LargeErRegN

  If NtotRej > 0
    ReDim Preserve Rejected[1 To NtotRej]

    For i = (Nrej + 1) To NtotRej
      Rejected[i] = LargeErRej[i - Nrej]
    Next i

    BubbleSort Rejected --subroutine Bubblesort is, according to Ludwig,
    --a "quick and dirty string-sorter" which can also be used for numbers.
    --I have not documented it; does not seem worthwhile. All that is
    --required here is that the elements of the vector Rejected are
    --rearranged into ascending order.

    Nrej = NtotRej

    t1 = Rejected[1] --Ludwig now constructs the *COMMA*-delimited string
    --of index-numbers of rejected analyses, for eventual placement at the
    --base of the "WtdMeanA" summary on StandardData sheet.

    For i = 2 To Nrej
      t1 = t1 & ", " & Rejected[i] --separate index nos with "comma-space"
    Next i

    wW[7, 1] = t1

  Else --i.e. NtotRej = 0

    t1 = "none"

  End If --NtotRej > 0

  --Finally, Ludwig assembles the various elements in their 'final' form,
  --as shown beneath the spot-rows on StandardData sheet:

  Cells[h, Acol] = WtdMean
  Cells[h + 1, Acol] = WtdMeanErr
  --Assignment of Cells[h + 2, Acol] is separate, outside this Else statement
  Cells[h + 3, Acol] = MSWD
```

```
      Cells[h + 4, Acol] = Prob
      Cells[h + 5, Acol] = wW[7, 1]

      --Finally, derive an element-dependent label-string:
      t2 = "Wtd Mean of Std Pb/" & Ele & " calibr."

    End If --(pbCanDriftCorr = TRUE)

    Set ExtPerr = Cells[h + 2, Acol] --identifies the cell to be known as ExtPerr
    ExtPerr.Name = "ExtPerr" & s --names it ExtPerr1 or ExtPerr2, based on DauParNum
    ExtPerr = ExtPtSigma --assigns the double-precision value calculated by WtdAv

    If pbCanDriftCorr = TRUE
      [out-of-scope stuff]
    End If

    If MSWD >= 100
      MSWD = Drnd(MSWD, 3)
      --Drnd is a function (Isoplot3 - Pub) that takes a double-precision number
      --(the first argument), and rounds it to a set number of significant figures
      --(the second argument). I don't understand the need for this If statement.
    End If

    If pbCanDriftCorr = TRUE

      [out-of-scope stuff]

    ElseIf Cells(h + 5, Acol).Value > 0
    --i.e. if there ARE some rejected spots,then show the rejected calib.-
    --constant values with strikethrough, and colour them yellow:

      For i = 1 To Nrej
        j = Adatrange[Rejected[i], 1].Row --selects the row

        Fonts j, Acol, , 3 + Acol, StrikeThrough=True
        --applies StrikeThrough to four columns in that row, the leftmost of
        --which is the calibration-constant value column

        IntClr vbYellow, j, Acol, , 3 + Acol
        --applies interior colour yellow to the same range of cells
      Next i

    End If
```

The next thing SQUID 2.50 does is define the cell-ranges which will contain the "Age(Ma)" data, the "Age(Ma)±1sigma" and the "Age(Ma)±2sigma". In all cases, these three columns are stored immediately to the right of their source calibration-constant value-uncertainty pairs, as defined below (recalling that rw1 and rwn are the first and last data-rows respectively on sheet StandardData):

```
AddName "Aadat" & s, TRUE, rw1, 1 + Aecol, rwn, 1 + Aecol
AddName "Aaerdat1_" & s, TRUE, rw1, 2 + Aecol, rwn, 2 + Aecol
AddName "Aaerdat2_" & s, TRUE, rw1, 3 + Aecol, rwn, 3 + Aecol
```

Next, assemble formulae for age-calculation, which naturally depend on the identity of the daughter-parent (i.e. 206Pb/238U or 208Pb/232Th), but also the identity of the index isotope (204-corr, 207-corr, etc.) under our "calculate everything" philosophy. So below, I have used 'X' as shorthand identifier for all 'permissible' index isotopes in a given permutation (recall the Cases outlined in the Introduction of Procedural Framework Part 3).

```
If (pbU = TRUE AND NumDauPar = 1) OR (pbTh = TRUE AND NumDauPar = 2)
  --we are dealing specifically with a 206Pb/238U calib. constant:

  t1 = "=LN( 1 + ["X-corr206Pb/238Ucalibr.const"] / ("WtdMeanA" & s)
    * StdUPbRatio ) / Lambda"

  --Now fill column ["X-corr206Pb/238U Age(Ma)"]:
  For i = rw1 To rwn
    PlaceFormulae t1, i, piaSAgeCol[DauParNum]
  Next i

  t4 = "EXP( Lambda * ["X-corr206Pb/238U Age(Ma)"] )"

  t5 = ["X-corr206Pb/238Ucalibr.const %err"] / 100 * ( t4 - 1 ) / Lambda / t4

 --Now fill columns ["X-corr206Pb/238U Age(Ma)±1sigma"] and
 --["X-corr206Pb/238U Age(Ma)±2sigma"]
  For i = rw1 To rwn
    PlaceFormulae t5, i, piaSAgeECol[DauParNum]  --±1sigma
    PlaceFormulae "2 * t5", i, 1 + piaSAgeECol[DauParNum]  --±2sigma
  Next i

Else
  --we are dealing specifically with a 208Pb/232Th calib. constant:
  t1 = "=LN( 1 + ["X-corr208Pb/232Thcalibr.const"] / ("WtdMeanA" & s)
      * StdThPbRatio ) / Lambda"

  --Now fill column ["X-corr208Pb/232Th Age(Ma)"]:
  For i = rw1 To rwn
    PlaceFormulae t1, i, piaSAgeCol[DauParNum]
  Next i

  t4 = "EXP( Lambda * ["X-corr208Pb/232Th Age(Ma)"] )"

  t5 = ["X-corr208Pb/232Thcalibr.const %err"] / 100 * ( t4 - 1 ) / Lambda / t4

 --Now fill columns ["X-corr208Pb/232Th Age(Ma)±1sigma"] and
 --["X-corr208Pb/232Th Age(Ma)±2sigma"]
  For i = rw1 To rwn
    PlaceFormulae t5, i, piaSAgeECol[DauParNum]  --±1sigma
```

```
      PlaceFormulae "2 * t5", i, 1 + piaSAgeECol[DauParNum]  --±2sigma
    Next i

  End If
```

The SQUID 2.50 code now embarks on a massive round of plotting and formatting, most of which seems irrelevant to the arithmetic. Buried inside it, however, is a call to the complex subroutine "ExtractGroup", which I will document separately and in detail, as it is also invoked when assessing populations of sample/unknown analyses. Its task is to find and extract the largest "statistically coherent" subgroup of analyses (i.e. with a weighted mean exceeding a specified probability-of-fit threshold), and this functionality is very useful to analysts trying to extract some sort of "rock age" from a high-n dataset complicated by lots of inherited zircon, Pb loss, etc.

```
  If pbCanDriftCorr = TRUE

    [out-of-scope stuff]

  Else --i.e. pbCanDriftCorr = FALSE

    ExtractGroup TRUE, 0, UPbConst, FALSE, TRUE, 0, [NULL], DauParNum, [NULL]
    --subroutine documented separately

  End If
```

Finally, I also found the code that manually over-rides a calculated external error that is "too small" (in the subjective opinion of the analyst) with a user-specified static minimum value. I have included some of the addressing code, because like everything else so far, this manipulation is theoretically applicable to every set of calibration-constant calculations:

```
  If pbCanDriftCorr = FALSE

    With Range("WtdMeanA" & s)
      r = Range.Row
      c = Range.Column
    End With

  End If

  Set extBox = frSr(r, c)

  With extBox
    extBox.Name = "ExtPerrA" & s --note that extBox.Name is the identity
    --of the external uncertainty that is actually propagated to
    --measurements of the unknowns
```

```
      extBox.Formula = "=MAX( "User-specified minimum external 1sigma %err", ExtPerr
      --user-specified value is set in Preferences: GA uses 0.75% based on
      --long-term observations
    End With

  Next DauParNum

  End Sub
```

SQUID 2.50 Function: fdNmad

This subroutine evaluates the double-precision value fdNmad (median absolute deviation) corresponding to an input vector v comprising double-precision numbers.

Usage

fdNmad(v)

Mandatory variables

v: Vector containing double-precision values.

Definition of variables

Values of type Integer
i, N, Nn

Values of type Double
med, medyr2, fdNmad

Arrays of type Double
v, yr2

The function utilises the Excel function UBound (http://www.excelfunctions.net/vba-ubound-function.html) to determine the length of the input vector:

```
N = UBound( v ) --like length(v)
med = Median( v )

Nn = Max( 3, N ) --minimum value of 3
ReDim yr2[1 to N] --apparently not zero-addressed

For i = 1 to N
  yr2[i] = ( v[i] - med )^2
Next i

medyr2 = Median( yr2 )

fdNmad = 1.4826 * ( 1 + 5 / (Nn - 2) ) * Sqrt( medyr2 )
```

```
End Function
```

The magic number 1.4826 is hard-coded (by Ludwig and by most other users of MAD-related functions: this value ensures that "± MAD" encompasses 50% (i.e. between 1/4 and 3/4) of the normal cumulative distribution function (e.g. https://en.wikipedia.org/wiki/Median_absolute_deviation).

SQUID 2.50 Sub: ExtractGroup

This documentation has been (temporarily) abandoned incomplete as of 2018-04-27, because it is not clear what effect (if any) these calculations can have on calibration constants. This subroutine and it's sub-subroutines will be revisited at the time it becomes necessary to deal with Grouping of Sample analyses.

This subroutine extracts a statistically coherent age group (i.e. one with probability of fit >= Mprob), and (in Excel-speak) constructs its range addresses, as well as placing the results on the active sheet. The SQUID 2.50 VBA code also constructs a weighted-average chart inset, but that aspect of its function is not documented here.

This procedure was probably written long ago; once again, it makes extensive use of Ranges in contexts where newer code-modules, executing similar operations, have explicitly defined and utilised double-precision arrays. VBA code dealing with arrays tends to be much more intelligible than that dealing with spreadsheet-ranges, but the latter is what we've got.

I have documented this subroutine in detail, because it is also invoked when assessing populations of sample/unknown analyses. Its functionality is very useful to analysts trying to extract some sort of "rock age" from a high-n dataset complicated by lots of inherited zircon, Pb loss, etc.

Usage

ExtractGroup Std, Mprob, t, RedoOnly, StdCalc, AgeResult, TypeCol, DpNum, Xrange

Mandatory variables

Std: Boolean input indicating whether the input to ExtractGroup is from the StandardData sheet or not.

Mprob: Double-precision value (0 <= Mprob <= 1) defining the minimum probability-of-fit required to satisfy the test of "statistical coherence" in the extracted group. Mprob = 0.05 is a common setting.

t: The cell-range containing the input data (can be thought of as a double-precision array). I think it usually has two columns: values and uncertainties

Optional variables

RedoOnly: Boolean input indicating whether this invocation of ExtractGroup is being run 'from scratch' (RedoOnly = FALSE), or whether it is a 're-run' of an pre-existing weighted-mean calculation (RedoOnly = TRUE) but aimed at investigating the effect of user-defined rejections and inclusions since the previous time ExtractGroup was run on the input data. **DEFAULT** value = FALSE.

StdCalc: Boolean input indicating whether this invocation of ExtractGroup is being run on calibration-copnstant value-uncertainty pairs on the StandardData sheet (StdCalc = TRUE) or not. **DEFAULT** value = FALSE.

AgeResult: Double-precision output value of the weighted average calculation.

TypeCol: Appears unused in the final implementation of the routine.

DpNum: For StandardData calibration-constants only, an integer value indicating whether ExtractGroup is being invoked for the 'primary' daughter-parent ratio (DpNum = 1) or the alternate ratio (DpNum = 2). **DEFAULT** value = 1.

Xrange: Unused in the final implementation of the routine.

Definition of variables - NEEDS UPDATING FOR EXTRACTGROUP

Values of type Boolean
NoReject, NoUPbConstAutoReject, pbCanDriftCorr, pbU, pbTh, [Task]Switch.DirectAltPD

Values of type Integer
c,DauParNum, ErN, h, i, j, LargeErRegN, Npts, Nrej, NtotRej, NumDauPar, r, rw1, rwn

Values of type Double
ExtPerr, ExtPtSigma, Lambda, MedianEr, MSWD, Nmadd, Prob, pscLm2, pscLm8, StdThPbRatio, StdUPbRatio, WtdMean, WtdMeanErr

Values of type String
s, Ele, t1, t2, t4, t5

Vectors of type Integer
LargeErRej, Rejected

Vectors of type Double

Acol, Aecol, Arange, Aerange, ErrVals

Arrays of type Double
Adatrange

Arrays of type Variant/Mixed
wW

The function commences with some definitions.

```
--Reinitialise AgeResult:
If fbNIM( AgeResult ) = TRUE --i.e. AgeResult exists
  AgeResult = 0
End If

--Define Boolean DoAll:
If RedoOnly = TRUE
  DoAll = FALSE
Else
  DoAll = TRUE
End If

--Define string version of DpNum:
If Std = TRUE
  Dp = fsS(DpNum) --fsS converts double-precision to string
Else
  Dp = "" --i.e. nothing
End If
```

Recalling that the input data t is defined as a single contiguous range (on a spreadsheet), it is necessary to define where on the spreadsheet the range is located, as well as the locations of the important components of the range:

```
With t
  ar1 = .Row --address of the FIRST row of the range
  ArN = .Rows.Count --number of rows in the range
  ArL = ar1 + ArN - 1 --derived address of the LAST row of the range
  arc = .Column --address of the LEFT (value/age) column of the range
End With
```

Ludwig now defines the Boolean Rad76age by searching the cell at [ar1 - 1, arc], which is the cell immediately **above** the top-left cell of the input range, on the off-chance that it is a column-header which contains the text-fragment "207Pb/206Pb". If it does, then Rad76age =

TRUE, else it is FALSE.

Next is some addressing of the column-header row, the first row of data and the last row of data on any given worksheet, bearing in mind that the addresses of these rows can differ between the StandardData and SampleData sheets:

```
HdrRowGrp = flHeaderRow[-Std]  --recalling Std = TRUE has numeric value -1, FALSE =
FirstGrpDatRw = plaFirstDatRw[-Std]
LastGrpDatRw = plaLastDatRw[-Std]

--Then define the range Hours, by finding the relevant range on the relevant sheet:
Set Hours = frSr(FirstGrpDatRw, piHoursCol, LastGrpDatRw)
```

Now populate the arrays rX and RxE (or X and Xerr), as well as vOK, according to the Boolean inputs of ExtractGroup, as well as the presence of numeric data in the 'values' and 'errors' columns of range t:

```
N = ArN

If RedoOnly = TRUE Or StdCalc = TRUE

  SetArrayVal TRUE, vOK
  --simply sets all elements of VOK to value TRUE

Else

  For i = 1 To ArN

    tB = TRUE
    vOK[i] = FALSE

    For j = 1 To 2
      tB = tB And ( IsEmpty(t[i, j]) = FALSE And IsNumeric(t[i, j] = TRUE )
    Next j

    If tB = TRUE
      k = 1 + k

      If StdCalc = TRUE
        rX[k] = t[i, 1]
        RxE[k] = t[i, 2]
      Else
        X[k] = t[i, 1]
        Xerr[k] = t[i, 2]
      End If

      vOK[i] = TRUE
    End If
```

```
  Next i

  Nn = k
  BadCt = N - Nn
  OkCt = Nn

End If

If N < 2 --should come immediately after "N = ArN", surely!
  Exit Sub
End If
```

How the code proceeds depends on what it is trying to do. If StdCalc = TRUE, then the business of identifying and rejecting outliers is covered elsewhere (i.e. in WtdmeanAcalc), and if RedoOnly = TRUE, then an attempt at extracting a group has already been made, and the code is merely assessing the effect of modifications manually applied by the user (i.e. rejections/unrejections defined by the presence/absence of Strikethrough font) since the last ExtractGroup attempt.

```
If RedoOnly = TRUE Or StdCalc = TRUE
  Nn = 0
  j = 0
  k = 0
  ReDim rX[1 To ArN] --values column, for Stds

  For i = 1 To ArN

    Set Tt = t[i, 1]
    --range Tt is the 'value', row-by-row, in range t

    If Len(Tt.Text) > 0 And IsNumeric(Tt) = TRUE And Val(Tt) > 0

      If Tt.Font.Strikethrough = FALSE And fbIsNumber(Tt.Text) = TRUE
        Nn = 1 + Nn
        rX[Nn] = Tt
        RxE[Nn] = t[i, 2] / 100 * Tt
      End If

    End If

  Next i

  OkCt = Nn
  BadCt = ArN - OkCt

ElseIf StdCalc = FALSE

  FindCoherentGroup N, Nn, BadCt, BadRwIndx, X(), Xerr(), rX(),
     RxE(), Mean, MeanErr, MSWD, Prob, Mprob, pdMinFract, OkVal
  --subroutine to be documented separately
```

```
  If OkVal = FALSE
    CFs LastGrpDatRw + 2, arc, "No coherent age group"
    --place string "No coherent age group" in the 'values' column,
    --two rows below the final row of the input range.

    Exit Sub
  End If

Else --i.e. StdCalc = TRUE And RedoOnly = FALSE

  If Nn > 2
    OkVal = TRUE
  Else
    OkVal = FALSE
  End If

End If
```

There follows a block of text aimed primarily at formatting (I think). Picking it up again at the point where a ±2sigma (absolute) error-column is inserted adjacent to (and to the right of) the ±1sigma (absolute) error column:

Code documentation temporarily abandoned at this point 2018-04-27

```
If StdCalc = TRUE

  Er2SigCol = 4 + t.Column
  --4 columns to the right of the calib. const. 'values' column, because
  --the intervening columns are calib. const. %err, Age (Ma), and
  --Age (Ma) ±1sigma, and Er2SigCol is the address for Age (Ma) ±2sigma.

Else --all other cases

  Er2SigCol = 2 + t.Column
  --2 columns to the right of the 'values' column, because the only
  --intervening column is ±1sigma, and Er2SigCol is the address for ±2sigma.

End If

If RedoOnly = FALSE And StdCalc = FALSE And Cells[HdrRowGrp, Er2SigCol] <> ""

  For i = FirstGrpDatRw  To LastGrpDatRw
    Cells[i, Er2SigCol] = "=2*" & Cells[i, Er2SigCol - 1].Address(0, 0)
  Next i

End If

If Nn = 0
  MsgBox "Can't reduce this data."
  End
```

```
End If

OkCt = 0
BadCt0 = BadCt
BadCt = 0
Tct = t.Rows.Count --can't see how this differs from ArN

If BadCt0 > 0
  ReDim TmpBadRwIndx[1 To BadCt0]
End If

NokGrps = 0
NbadGrps = 0
StartNewGrp = FALSE

For j = 1 To Tct

  OkCt0 = OkCt

  For i = 1 To UBound(rX) --i.e. length (rX)

    Set Tt = t[j, 1]

    If IsNumber(Tt.Text) = TRUE And Tt.Font.Strikethrough = FALSE

      If Tt <> 0 And Tt = rX[i]

        Set Ra1 = Tt[1, 0]
        Set Ra4 = Tt[1, 1]
        Set Ra5 = Tt[1, 2]

        If StdCalc = TRUE
          Set Ra2 = Tt[1, 3]
          Set Ra3 = Tt[1, 5]
        Else
          Set Ra2 = Tt[1, 1]
          Set Ra3 = Tt[1, 3]
        End If

        Set Ra6 = Range(Ra4, Ra5)
        Set Ra7 = Range(Ra1, Ra4)
        Set ur1 = Union(Ra1, Ra2)

        With Ra6
          .Font.Strikethrough = False
        End With

        OkCt = 1 + OkCt
        OK[OkCt] = rX[i]
        OKrwIndx[OkCt] = j

        If StdCalc = TRUE
          OkVals[OkCt, 1] = Ra1
          OkVals[OkCt, 2] = Ra4
          OkErrVals[OkCt] = Ra5 --1sigma absolute errors
        End If
```

```
        OkAgeVals[OkCt, 1] = Ra1
        OkAgeVals[OkCt, 2] = Ra2
        OkAgeErrVals[OkCt] = Ra3 --2sigma absolute errors

        If OkCt = 1
          NokGrps = 1
          ReDim OkAgeAddr[1 To 1], OkAgeErAddr[1 To 1], OkBreak[1 To 1]
          Set OKspots = Ra7
          Set OkErrs = Ra5
          Set OkAge = ur1
          Set OkAgeErrs = Ra3
        ElseIf StartNewGrp = TRUE
          NokGrps = 1 + NokGrps
          ReDim Preserve OkAgeAddr[1 To NokGrps], OkAgeErAddr[1 To NokGrps],
            OkBreak[1 To NokGrps]
          StartNewGrp = FALSE
          Set OkAge = ur1
          Set OkAgeErrs = Ra3
        Else
          Set OkAge = Union(OkAge, ur1)
          Set OkAgeErrs = Union(OkAgeErrs, Ra3)
        End If

        Set OKspots = Union(OKspots, Ra7)
        Set OkErrs = Union(OkErrs, Ra5)
        OkAgeAddr[NokGrps] = OkAge.Address
        OkAgeErAddr[NokGrps] = OkAgeErrs.Address
        OkBreak[NokGrps] = j

        Exit For

      End If --Tt <> 0 And Tt = rX[i]

    End If --IsNumber(Tt.Text) = TRUE And Tt.Font.Strikethrough = FALSE

  Next i

  If OkCt = OkCt0
    BadCt = 1 + BadCt
    BadRwIndx[BadCt] = j
  End If

Next j

If BadCt > 0

  k = Max( BadCt, BadCt0 )

  ReDim Preserve BadRwIndx[1 To k], TmpBadRwIndx[1 To k]
  ReDim TmpBadRwIndx[1 To k]

  NbadGrps = 0
  StartNewGrp = FALSE

  For i = 1 To BadCt
```

```
      TmpBadRwIndx[i] = BadRwIndx[i]
    Next i

    k = 0

    For i = 1 To BadCt
      m = TmpBadRwIndx[i]

      With Range( t[m, 1], t[m, 2] )
        .Font.Strikethrough = True
        .Interior.Color = vbYellow
      End With

      Set Tt = t[m, 1]

      If IsNumber(Tt.Text) = TRUE

        If Val(Tt) <> 0
          k = 1 + k

          Set Ra1 = Tt(1, 0)

          If StdCalc = TRUE
            Set Ra2 = Tt[1, 3]
            Set Ra3 = Tt[1, 5]
          Else
            Set Ra2 = Tt[1, 1]
            Set Ra3 = Tt[1, 3]
          End If

          Set Ra7 = Union(Ra1, Ra2)

          BadRwIndx[k] = m
          BadAgeVals[k, 1] = Ra1
          BadAgeVals[k, 2] = Ra2
          BadAgeErrVals[k] = Ra3

          If k = 1
            NbadGrps = 1
            ReDim BadAgeAddr[1 To 1], BadAgeErAddr[1 To 1], BadBreak[1 To 1]

            Set BadAge = Ra7
            Set BadAgeErrs = Ra3

          ElseIf StartNewGrp = TRUE
            NbadGrps = 1 + NbadGrps
            ReDim Preserve BadAgeAddr[1 To NbadGrps], BadAgeErAddr[1 To NbadGrps],
              BadBreak[1 To NbadGrps]

            StartNewGrp = TRUE
            Set BadAge = Ra7
            Set BadAgeErrs = Ra3

          Else
            Set BadAge = Union(BadAge, Ra7)
            Set BadAgeErrs = Union(BadAgeErrs, Ra3)
```

```
        End If

        BadAgeAddr[NbadGrps] = BadAge.Address
        BadAgeErAddr[NbadGrps] = BadAgeErrs.Address
        BadBreak[NbadGrps] = m

      End If --Val(Tt) <> 0

    End If --IsNumber(Tt.Text) = TRUE

  Next i

  BadCt = k

  If NbadGrps > 0

    If BadBreak[NbadGrps] = 0
      BadBreak[NbadGrps] = m
    End If

  End If

  If BadCt > 0
    ReDim Preserve BadRwIndx[1 To k]
  End If

End If --BadCt > 0
```

SQUID 2.50 Sub: OverCtMeans

This subroutine (which is solely **for the Standard**) does a bit more than the name implies. Firstly it places, row-by-row, formulae to calculate the 204-corrected 207Pb/206Pb **ratio** and its 1sigma percentage uncertainty, as well as invoking (row-by-row) the relevant LudwigLibrary functions to calculate the associated 204-corrected 207Pb/206Pb **date** and its 1sigma absolute uncertainty.

Secondly, it identifies which columns can usefully have robust means calculated, performs those calculations (using LudwigLibrary function TukeysBiweight, with tuning 9) and places the output of the expression as a 3 x 1 array beneath the relevant column. The SQUID 2.50 subroutine requires the index number of the last row of analytical data as an input, so it can determine in which rows the "summary" results should be placed so that the calculated biweights appear directly beneath the input data.

Usage

OverCtMeans plaLastDatRw

Mandatory variable

plaLastDatRw: Integer index number of the last row containing spot-by-spot data (for the Standard).

Definition of variables

Values of type Boolean
ShowOverCtCols

Values of type Integer
c, k, kk, plaFirstDatRw

Values of type String
t0, t1, t2, t3, t4, t5

The subroutine starts by calculating the 204Pb-corrected 207Pb/206Pb, its uncertainty, and the associated date and uncertainty.

```
If piaAgePb76_4Col[1] > 0 --i.e. if column ["4-corr207Pb/206Pb"] exists on Standard

  t0 = "=( ["207/206"] / ["204/206"] - sComm_74 ) / ( 1 / ["204/206"] - sComm_64 )"

  --Then place this formula in column ["4-corr207Pb/206Pb"] on StandardData:
  PlaceFormulae t0, plaFirstDatRw, piStdPb76_4Col, plaLastDatRw

  --Now calculate the associated %err:
  t1 = "( ( ["207/206"] * ["207/206%err"] )^2 +
       ( ["204/206"] * ( ["4-corr207Pb/206Pb"] * sComm_64 - sComm_74)
          * ["204/206%err"] )^2 )"
  t2 = "( ["207/206"] - ["204/206"] * sComm_74 )^2"
  t3 = "=sqrt( t1 / t2 )"

  --Then place this formula in column ["4-corr207Pb/206Pb%err"] on StandardData:
  PlaceFormulae t3, plaFirstDatRw, piStdPb76_4eCol, plaLastDatRw

  --Now invoke LudwigLibrary functions to calculate associated age and error:
  t4 = "=AgePb76( ["4-corr207Pb/206Pb"] )"

  --Then place this formula in column ["4-corr207Pb/206Pbage"] on StandardData:
  PlaceFormulae t4, plaFirstDatRw, piaAgePb76_4Col, plaLastDatRw

  --Convert ["4-corr207Pb/206Pb%err"] to absolute, to calculate age error:
  t5 = "=AgeErPb76( ["4-corr207Pb/206Pb"] , ["4-corr207Pb/206Pb"]
      * ["4-corr207Pb/206Pb%err"] / 100 )"

  --Then place this formula in column ["4-corr207Pb/206Pbage±1sigma"] on StandardDa
  PlaceFormulae t5, plaFirstDatRw, piaAgePb76_4eCol, plaLastDatRw

End If
```

The second part of the subroutine calculates the various biweight means, for the range of columns for which they are relevant. In practice, the "entry" Boolean is always TRUE, because Ludwig set the nominally user-defined Boolean ShowOverCtCols to TRUE, and then removed the relevant check-box from the user form! Note that a possibly unintended consequence of "locking in" this Boolean is that it formally requires *all* SQUID 2.50 U-Pb Geochronology Tasks to include 204Pb in the list of mass-stations.

__(There will come a time when it is necessary for us to formally assess the absolute minimum mass-stations in order to perform U-Pb (or Th-Pb) geochronology in SQUID 3.0. My feeling is that the absolute minimum lists are as follows:

- 206Pb/238U: 206Pb, 238U (or a proxy thereof), and ONE OF (204Pb or 207Pb)
- 208Pb/232Th: 208Pb, 232Th (or a proxy thereof), and ONE OF (204Pb or 207Pb)

At present, SQUID 2.50 enforces "BOTH OF". Certainly the "alternate" daughter-isotope (i.e.

208Pb for U-Pb, 206Pb for Th-Pb) should be optional, and probably Background should be optional too (even though including 204Pb in a run-table without including Background would be terrible practice. Something to revisit later, and the stakes are relatively low: it doesn't really matter exactly what the 'bare bones' list of mass-stations is, as long as we warn SQUID 3.0 users up-front… it would be an improvement on SQUID 2.50!)__

The subroutine proceeds:

```
If ShowOverCtCols = TRUE OR piaAgePb76_4Col[1] > 0 --i.e. if StandardData
  contains ["4-corr207Pb/206Pbage"]

  If piaOverCts4Col[7] > 0 Or piaOverCts4Col[8] > 0 Or piaAgePb76_4Col[1] > 0
  --i.e. if ANY of the columns ["204overcts/sec(fr. 207)"],
      ["204overcts/sec(fr. 208)"],
  --["4-corr207Pb/206Pbage"] exist on the StandardData sheet, then define
      the extent of calculations
  --to be performed:

    If ShowOverCtCols = TRUE
      kk = 1
    Else
      kk = 5
    End If

    For k = kk To 5

      Select Case k --all column-indices refer to StandardData sheet:
        Case 1: c = piaOverCts4Col[7] --index for ["204overcts/sec(fr. 207)"]
        Case 2: c = piaOverCts4Col[8] --index for ["204overcts/sec(fr. 208)"]
        Case 3: c = piacorrAdeltCol[7] --index for ["7-corr206Pb/238Uconst.delta%"]
        Case 4: c = piacorrAdeltCol[8] --index for ["8-corr206Pb/238Uconst.delta%"]
        Case 5: c = piaAgePb76_4Col[1] --index for ["4-corr207Pb/206Pbage"]
      End Select

      If c > 0

        --Define Range bw as 3 x 1, with the first row immediately following
            analytical data:
        Set bw = frSr(1 + plaLastDatRw, c, 3 + plaLastDatRw) --sets Range
            bw = 3 rows by 1 col

        --Invoke LudwigLibrary function Biweight (tuning 9) on the data in
            the column above bw.
        --Note that for k = 3 or 4, the values in this range have not yet
            been calculated!
        bw.FormulaArray = "=Biweight(" & frSr(plaFirstDatRw, c, plaLastDatRw).Addre

        --Finally, add Names (and explanatory Notes) to result-cells. These are
            relevant because
        --the Names can be used in Task-expressions, and the Notes matter
            because the columns are
        --quite abstract concepts for non-SHRIMP geochronologists (as well
```

```
      as SHRIMP beginners):

--Note that I have recast the following If to make it longer but more trans

If k = 1
  t1 = "StandardData!Pb204OverCts7corr"
  AddName t1, TRUE, 1 + plaLastDatRw, c --Name for the
      Biweight Value element
  t2 = "StandardData!Pb204OverCts7corrEr"
  AddName t2, TRUE, 3 + plaLastDatRw, c --Name for the
      Biweight 95% conf. element
  t3 = "Robust avg 204 overcts assuming 206Pb/238U-207Pb/235U age
      concordance"
ElseIf k = 2
  t1 = "StandardData!Pb204OverCts8corr"
  AddName t1, TRUE, 1 + plaLastDatRw, c --Name for the Biweight Value
      element
  t2 = "StandardData!Pb204OverCts8corrEr"
  AddName t2, TRUE, 3 + plaLastDatRw, c --Name for the Biweight
      95% conf. element
  t3 = "Robust avg 204 overcts assuming 206Pb/238U-208Pb/232Th age
      concordance"
ElseIf k = 3
  t1 = "OverCtsDeltaP7corr"
  AddName t1, TRUE, 1 + plaLastDatRw, c --Name for the Biweight Value
      element
  t2 = "OverCtsDeltaP7corrEr"
  AddName t2, TRUE, 3 + plaLastDatRw, c --Name for the Biweight
      95% conf. element
  t3 = "Robust avg of diff. between 207-corr. and 204-corr. calibr. const."
ElseIf k = 4
  t1 = "OverCtsDeltaP8corr"
  AddName t1, TRUE, 1 + plaLastDatRw, c --Name for the Biweight Value
      element
  t2 = "OverCtsDeltaP8corrEr"
  AddName t2, TRUE, 3 + plaLastDatRw, c --Name for the Biweight
      95% conf. element
  t3 = "Robust avg of diff. between 208-corr. and 204-corr. calibr. const."
Else --interestingly, SQUID 2.50 does not assign Names (t1, t2) to the
    Biweight
  --elements calculated for the ["4-corr207Pb/206Pbage"] column.
      Probably an oversight!
  t3 = "Robust average of 204-corrected 207/206 age"
End If --k = 1 to 5

--Now VBA function "Note" to add text t3 as Comment on the Biweight
    Value element:
Note 1 + plaLastDatRw, c, t3

--Corresponding Comment on the Biweight 95% conf. element is
    context-sensitive:
If k = 3 OR k = 4
  t3 = "95%-conf. error in above difference"
Else
  t3 = "95%-conf. error in above"
End If
```

```
        Note 3 + plaLastDatRw, c, t3

      End If --c > 0

    Next k

  End If --piaOverCts4Col[7] > 0 Or piaOverCts4Col[8] > 0 Or
      piaAgePb76_4Col[1] > 0

  ClearObj bw

End If --ShowOverCtCols = TRUE OR piaAgePb76_4Col[1] > 0

End Sub
```

Sq2.50 Procedure Pt 5

The final phase of Loop A "cleans up" the 232Th/238U calculations in the Perm2 and Perm4 cases (i.e. where piNumDauPar = 2): the subroutine ThUfromA1A2 can now be executed properly, because WtdMeanA1 and WtdMeanA2 now both exist. Unfortunately, this entails some really nasty code, much of which looks (to my eye) obsolete and pointless. But I can't be certain, so I have documented it below.

Recalling that pbHasTh is TRUE if the run-table contains at least one peak with mass 232, 248 or 264 (which it usually does, for geochronology), and pbHasU is TRUE if the run-table contains at least one peak with mass 238, 254 or 270 (which it almost always will, for geochronology), then the code for the Perm2 and Perm4 scenarios proceeds (from the end of Part 4) as follows:

```
If (pbHasTh = TRUE) AND (pbHasU = TRUE) AND (piNumDauPar = 2)

  --Then replace the dummy values of "1" for 232Th/238U with true formulae
  If (pbStd = TRUE) AND (piNumDauPar = 2)

    Set CalibConst1 = frSr(1 + plaFirstDatRw[1], [WtdMeanA1].Column, Lrw)
```

The previous line looks like it contains a bug: this frSr statement defines a one-column cell-range containing the calibration-constant values for the **primary** calibration constant (i.e. 206Pb/238U for Perm2, 208Pb/232Th for Perm4), but the use of "1 + plaFirstDatRw[1]" means that the cell-range **excludes** the first row! Having stepped through the code, I am convinced this is a slip, and that the "1 +" should be deleted. Let's leave it in place for now, because with respect to our SQUID-books of record, it's "what SQUID 2.50 does".

The next line of code invokes a SQUID 2.50 subroutine which Ludwig named "Clean", which is a bit confusing, because Microsoft VBA has a function of the same name, which does a different thing. For the sake of this documentation (and to allay my own considerable confusion!), I have renamed Ludwig's subroutine "SQUIDClean". In addition to the first two arguments (which are the input and output cell-ranges) and the third argument (which enumerates the "clean" data-rows), SQUIDClean has 7 optional Boolean arguments, the first 6 of which are assigned their default values. The last ("AddStrikeThru") is set to TRUE, where its default is FALSE:

```
    SQUIDClean CalibConst1, CleanedConst, 0, , , , , , , TRUE
    --subroutine documented separately
```

In this context, SQUIDClean takes the input range of 20 (not 21!) **primary** calibration-constant values (remember that the first row is ignored owing to the bug described above), and "cleans"

them to remove those that have already been excluded via the application of Strikethrough font as part of the WtdMeanAcalc subroutine earlier. CleanedConst comprises the 15 **primary** calibration-constant values that survive the cleaning process.

```
    AvCalibrConst = Average( CalibConst1 )

    If CleanedConst.Count < CalibConst1.Count --these "counts" refer to number of
      p = 12
    Else
      p = 0
    End If

  End If --(pbStd = TRUE) AND (piNumDauPar = 2)
```

I do not understand the relevance of the parameter p. As far as I can see, its sole purpose is to force multiple iterations of the subroutine ThUfromA1A2 in the code below, but the benefit of these iterations remains unclear, as I can't see any place that the value of p or its dependents actually influences the input to ThUfromA1A2… The code continues:

```
  Do

    p = p + 1
    AvCalibrConst0 = AvCalibrConst

    For DatRow = Frw To Lrw
      ThUfromA1A2 pbStd, (DatRow), FALSE
      --because WtdMeanA1 and WtdMeanA2 now exist
      --subroutine previously documented (Procedure Part 3)
    Next DatRow

    Recalculate

    If (p > 12) OR ( (pbStd = TRUE) AND (piNumDauPar = 2) ) = FALSE
      Exit Do
    End If

    AvCalibrConst = Average( CalibConst1 )
    vTmp = 100 * Abs( (AvCalibrConst - AvCalibrConst0) / AvCalibrConst )

  Loop Until (p > 1) AND (vTmp < 0.001)
```

I can't see any point to the iterated calculation of AvCalibrConst and vTmp. As far as I can see, the sole benefit of the above Do… loop is the invocation of subroutine ThUfromA1A2 for Perm2 and Perm4 Tasks, and even there, the subroutine need only be invoked once. The code proceeds by calculating ["ppmTh"] for the Perm2 and Perm4 cases:

```
If (piaPpmUcol[-pbStd] > 0) AND (piaPpmThcol[-pbStd] > 0)
--i.e. if the StandardData (or SampleData) sheets has columns
--for BOTH "ppmU" and "ppmTh":

  Term1 = " = ["232Th/238U"] * ["ppmU"] * 0.9678 "
  --"magic number" 0.9678 documented below

  --now use Term1 to populate column ["ppmTh"] for Perm2/Perm4
  PlaceFormulae Term1, Frw, {ppmTh-column}, Lrw

End If --(piaPpmUcol[-pbStd] > 0) AND (piaPpmThcol[-pbStd] > 0)
```

The "magic number" 0.9678 (4 decimal places) is hard-coded into the SQUID 2.50 code, which is a bit naughty because it actually is actually the product of several physical "constants" related to the masses and isotopic abundances of Th and U. Essentially it represents the product:

("Atomic mass of 232Th"/"Atomic mass of 238U") * (("238U"/"AllNaturalU") / ("232Th"/"AllNaturalTh"))

The atomic masses of 232Th and 238U **are** immutable physical properties: 238U comprises 92 protons and 146 neutrons by definition, so its atomic mass is 238 by definition. Similarly, 232Th is defined as the isotope containing 90 protons and 142 neutrons, so its atomic mass of 232 is an intrinsic property.

However, the ratios ("238U"/"AllNaturalU") and ("232Th"/"AllNaturalTh") have more scope for variation, and should certainly be modelled separately. Having scoured the internet and the literature, I believe Ludwig explicitly assumed exact values (4 decimal places) of 0.9928 for the former and 1.0000 for the latter. Both are perfectly reasonable values, but ought to be identified as explicit physical "constants" constraining the arithmetic. So the magic number 0.9678 represents the product:

(232/238) * (0.9928/1.0000)

rounded to 4 decimal places. For the present, we ought to retain the hard-coded "magic number" (0.9678). The If clause concludes with some tidying up for Standards, basically assembling a string of index-numbers of the Rejected calibration-constan:

```
If pbStd = TRUE

  --{Compile and report a list of index-numbers of Rejected calibration-constan
  --for each set of calibration constants}

End If
```

```
    End If --(pbHasTh = TRUE) AND (pbHasU = TRUE) AND (piNumDauPar = 2)
    --i.e. End of If that commenced Part 5

  Loop Until (pbStd = FALSE) OR (pbStdsOnly = TRUE) --End of Loop A that commenced at
```

This concludes the major "Standard-Sample" loop that has spanned **Procedural Framework Parts 1-5**.

The remainder of the master "SquidfGeochron" routine will be documented in Part 6.

I have named this subroutine "SQUIDClean". Ludwig named it "Clean" in his VBA code, but that is confusing, because there exists a Microsoft VBA function of the same name, and it took me a while to work out there was a difference!

Sub Clean(DatRange As Range, CleanedDat As Range, NumCleanRows%, _
Optional ZeroesOK As Boolean = False, Optional BlankOk As Boolean = False, _
Optional AllColsOK As Boolean = False, Optional BothNegPos As Boolean = True, _
Optional StrikeThruOK As Boolean = False, Optional AllComers As Boolean = False, _
Optional AddStrikeThru As Boolean = False)
' Returns Cleandat as array cleaned of all noncomplying rows.

Dim First As Boolean, OkCel As Boolean
Dim Nareas%, Ncols%, OKcol%, Col%, AreaIndx%, TempNum%, CleanedRowCt%
Dim Rw&, RowCt&, v#
Dim Cel As Range, Area As Range, Crow As Range

```
First = TRUE

With DatRange

  If AllColsOK = TRUE
    Ncols = DatRange.Columns.Count
  Else
    Ncols = 1
  End If

  CleanedRowCt = 0

  Nareas = DatRange.Areas.Count

  For AreaIndx = 1 To Nareas

    TempNum = 1 + CleanedRowCt

    Set Area = DatRange.Areas[AreaIndx]

    With Area
      RowCt = Area.Rows.Count

      For Rw = 1 To RowCt
        TempNum = 1 + CleanedRowCt
        OKcol = 0

        For Col = 1 To Ncols
          Set Cel = Area.Item[Rw, Col]
          OkCel = FALSE

          If (BlankOk = TRUE) OR (Cel.Formula <> "") OR (AllComers = TRUE) --Area1
```

```
            If (IsNumeric(Cel) = TRUE) Or (AllComers = TRUE) --Area2

              With Cel

                v = Cel.Value

                If (ZeroesOK = TRUE) OR (v <> 0) OR (AllComers = TRUE) --Cel1

                  If (BothNegPos = TRUE) OR (v > 0) OR (AllComers = TRUE) --Cel2

                    If (StrikeThruOK = TRUE) OR (Cel.Font.Strikethrough = FALSE) Or

                      OKcol = 1 + OKcol
                      OkCel = TRUE

                    End If --Cel3

                  End If --Cel2

                End If --Cel1

              End With --Cel

            End If --Area2

          End If --Area1

          If (OkCel = FALSE) AND (AddStrikeThru = TRUE)
            Cel.Font.Strikethrough = TRUE
          End If

          If OKcol < Col
            Exit For
          End If
        Next Col

        If OKcol = Ncols

          CleanedRowCt = 1 + CleanedRowCt
          Set Crow = Range( Area.Item[Rw, 1], Area.Item[Rw, Ncols] )

          If First = TRUE

            Set CleanedDat = Crow
            First = FALSE

          Else

            Set CleanedDat = Union( CleanedDat, Crow )

          End If --First = TRUE

        End If --OKcol = Ncols

      Next Rw
```

```
      End With --Area
    Next AreaIndx

  End With --DatRange

  NumCleanRows = CleanedRowCt

  End Sub
```

Sq2.50 Procedure Pt 6

Now that Loop A has finally been completed, the code finishes off with the calculation of the remaining columns (SQUID 2.50 performs these calculations separately for the StandardData and SampleData sheets). The code continues:

```
If (pbCanDriftCorr =TRUE) AND (pbStdsOnly = FALSE)

  [out-of-scope stuff]

End If
```

The next step is to calculate all LA-switched expression columns and cells. SQUID 2.50 does this via subroutine LastEquations, which has two Boolean inputs:

- StdCalc (TRUE = perform calculations on StandardData sheet; FALSE = SampleData sheet)
- IncludeSingleCells (obsolete; always TRUE. It was presumably replaced by Switch.SC at some point)

The code continues:

```
LastEquations TRUE, TRUE --subroutine not documented, probably obsolete
--This invocation calculates LA-switched expression on StandardData only.
```

"StdConcPlots" is a Boolean set buy the user in SQUID 2.50's preferences. In essence, it dictates whether SQUID 2.50 should attempt to portray analyses of the Standard on a conventional (Wetherill) concordia diagram, for visualisation of data quality. This is a somewhat artificial process, because in secondary ion mass spectrometry, calibrated 206Pb/238U is not independently determined for the Standard. Instead, the WtdMeanA value (determined from the calibration constants) is defined as a proxy for the IDTIMS-defined reference 206Pb/238U of the Standard. From there, 206Pb/238U "values" for individual analyses of the Standard can be derived by assessing the extent to which the analysis-specific calibration constant value is offset from the WtdMeanA value defined by the population, and offsetting the analysis-specific 206Pb/238U value from the IDTIMS-defined reference 206Pb/238U value in proportion.

It's not completely clear why StdConcPlots is user-controlled; I think maybe it could simply be set to TRUE, as it embodies calculations that are commonly useful, and which are not performed at all when StdConcPlots = FALSE. The only reason I can think of is that perhaps in geologically young reference materials (< 50 Ma), it might be possible to calculate spurious isotopic ratios that could not be properly represented on a concordia diagram. But it seems to me that in that case, error handling should be handled differently anyway. The code continues:

```
If StdConcPlots = TRUE

  StdRadiogenicCols plaFirstDatRw(1), plaLastDatRw(1)
  --subroutine documented separately

End If
```

The code now turns to the corresponding calculations on the SampleData sheet. It begins with a rather inelegant kluge on the SampleData set, essentially aimed at refreshing all the expression-results and calculations, and which I will not document.

```
If pbStdsOnly = FALSE --i.e. if processing loops includes Samples AND Standards

  [Activate SampleData sheet]
  Rows(plHdrRw).Font.Bold = TRUE

  If ( piaSpotCt[0] > 0 ) OR ( piaSpotCt[1] > 0 )
  --i.e. trivial check for the presence of ANY analyses

    [Kluge to "refresh formulas so will be current"]
    plOutputRw = plSpotOutputRw

    If piaSpotCt[0] > 0 --i.e. if there are ANY Sample analyses
    --then do SamRadiogenicCols from first to last row on SampleData

      SamRadiogenicCols plaFirstDatRw[0], plaLastDatRw[0]
      --subroutine being documented separately

    End If

  End If

  LastEquations FALSE, TRUE
  --This invocation calculates LA-switched expression on SampleData only.

End If --pbStdsOnly = FALSE
```

SQUID 2.50 Sub: StdRadiogenicCols

This subroutine (which is solely **for the Standard**) places, row-by-row, formulae to calculate radiogenic (i.e. corrected for common Pb) ratios for Pb/U (and Pb/Th, as a proxy) in the StandardData sheet.

Usage

StdRadiogenicCols plaFirstDatRw, plaLastDatRw

Mandatory variables

plaFirstDatRw: Integer index number of the first row containing spot-by-spot data (for the Standard).
plaLastDatRw: Integer index number of the last row containing spot-by-spot data (for the Standard).

Definition of variables

Values of type Boolean
pbTh, pbU

Values of type Integer
f, L, piNumDauPar

Values of type String
q, s, SA, sae, t, WtdMnA

Values of type Range
rw1, rw2, r4

The subroutine starts by determining the address of the column in which the row-by-row ["206Pb/238U calibr. const."] values will be found. In truth, this column occurs as the **primary** calibration constant for both Perm1 and Perm2, and as the **secondary** calibration constant in Perm4. However, Perm3 does not include a direct calculation of ["206Pb/238U calibr. const."], so this code incorporates an (isotopic) "kluge" where 208Pb/*232Th is substituted for 206Pb*/238U. It is important to note that this does not make sense in terms of portraying the

data (e.g. on a Concordia diagram), but at the same time, it is clear that Ludwig did this Perm3-specific substitution deliberately, as will become clear from the following code.

Note also that all of this SQUID 2.50 code is predicated on the calculation of **no more than one of each** calibration constant (206Pb/*238U and/or 208Pb*/232Th), because in SQUID 2.50, the index isotope has already been uniquely specified by the user. This means that in due course, some of this code will require additional generalisation, with particular respect to Perm1 and catering for the possibility of 208-corrected data. The following documents the SQUID 2.50 code as written, but also points out gaps to be addressed later.

The following "If" incorporates bug-fixes required in the SQUID 2.50 code (and implemented by me there, dated 2018-07-02). Ludwig realised that "SA" (which is the column of ["206Pb/238U calibr. const."] values) would be located in a different column for Perm4 than is the case for Perms1-3, as described above. However, he appears not to have realised that "WtdMnA" (i.e. the **value** of the weighted mean of ["206Pb/238U calibr. const."]) and "sae" (i.e. the location of ["206Pb/238U calibr. const. %err"] both **also** required analogous Perm4-specific addressing. Instead, the Perm1-3 addresses were used universally for WtdMnA and sae, with nonsensical results in Perm4.

```
If (pbTh = TRUE) and piNumDauPar = 2 --i.e. Perm4

  SA = " sA(2) "
  WtdMnA = " WtdMeanA2 "
  sae = " sAe(2) "
  --because 206/238 is the "secondary" calibr. const. in Perm4

Else --Perm1, Perm2, Perm3

  SA = " sA(1) "
  WtdMnA = " WtdMeanA1 "
  sae = " sAe(1) "
  --because 206/238 is the "primary" calibr. const. in Perm1 and Perm2, and
  --because for Perm3, 206/238 is being "kluged" using 208/232, which is the
  --"primary" calibr. const. in Perm3

End If

f = plaFirstDatRw
L = plaLastDatRw
```

Note that the following headers are not always an accurate representation of the data within the column. It is **true** that "207/*235*" = *["4corr-207Pb/*235U"] always, but the same is not always true for "206/*238*". *In fact, for Perm1, Perm2, and Perm4,* "*206*/238" = ["Xcorr-206Pb/*238U"], where X denotes the (user-specified) index isotope for common-Pb correction (i.e. one of 4, 7, or (in the case of Perm1 only) 8). In the case of Perm3,* "*206*/238" = ["Xcorr-

208Pb*/232Th"], where X denotes the (user-specified) index isotope for common-Pb correction (i.e. 4 or 7). This is definitely odd, but it seems clear in the code that it is deliberate…

```
If piStdPb7U5_4col > 0 --i.e. if column 204corr 207Pb*/235U exists on StandardData

  If (pi204PkOrder > 0) AND (pi207PkOrder > 0) AND (pbHasU = TRUE)
  --i.e. if mass stations 204Pb AND 207Pb AND at least one U-bearing mass-station
  --(e.g. 238U, 254UO, 270UO2) have ALL been measured, then define the headers:

    CFs plHdrRw, piStdPb7U5_4col, "207*|/235", -1 --header for 204corr 207Pb*/235U
    CFs plHdrRw, piStdPb7U5_4eCol, "%err", -1 --header for 204cor 207Pb*/235U %err

    --The following are so-called 204corr 206*/238 (and err corr) but see above!
    CFs plHdrRw, piStdPb6U8_4col, "206*|/238", -1 --header for 204corr 206Pb*/238U
    CFs plHdrRw, piStdPb6U8_4eCol, "%err", -1 --header 204corr 206Pb*/238U %err
    CFs plHdrRw, piStdPb7U5Pb6U8_4rhoCol, "err|corr", -1 --header 204c err correl.

  End If

End If
```

The next step is to specify the IDTIMS-determined reference isotopic ratio for the Standard (part of the Standard-specific "model", and derived in SQUID 2.50 simply by converting the reference "age" of the Standard (206Pb/*238U, or 208Pb*/232Th, or both) to the corresponding isotopic ratio value. As described previously, those age-specific reference isotopic ratios are stored as StdUPbRatio (206Pb/*238U), StdThPbRatio (208Pb*/232Th), and Std_76 (207Pb/*206Pb*), respectively. The code continues:

```
If piStdPb6U8_4col > 0 --i.e. if column 204corr 206Pb*/238U exists on StandardData

  If (pbTh = TRUE) AND (piNumDauPar = 1) --i.e. Perm3

    q = "StdThPbRatio"

  Else --i.e. Perm1, Perm2, Perm4

    q = "StdUPbRatio"

  End If

  s = "=" & SA & "/" & WtdMnA & "*" & q
  --so-called ["4corr-206Pb*/238U"], with Ludwig bugs repaired.

  --Now place the expression for so-called ["4corr-206Pb*/238U"]:
  PlaceFormulae s, f, piStdPb6U8_4col, L
```

```
    --The following line places the expression for so-called
    --["4corr-206*/238 %err"], but Ludwig's code contains a bug!

    --PlaceFormulae "= sAe(1) ", f, 1 + piStdPb6U8_4col, L --Ludwig original

    s = "=" & sae --Bodorkos 2018-07-02 addition
    PlaceFormulae s, f, 1 + piStdPb6U8_4col, L --Bodorkos 2018-07-02 revision

    --When primary D/P ratio = 208/232 and piNumDauPar = 2 (i.e. 206/238 calculated
    --directly as secondary ratio), corresponding to Perm4, Ludwig's original code
    --wrongly equates ["206*/238 %err"] with the %err of the 208/232 calibr. const.
    --This is obviously bad: instead, ["206*/238 %err"] should be equated with %err
    --of the 206/238 calibr. const. Easily fixed via Perm-dependent definition of
    --"sae" near the start of sub StdRadiogenicCols.

  End If

  If piStdPb7U5_4col > 0 --i.e. if column 204corr 207Pb*/235U exists on StandardData

    --Place the expression for ["4corr-207Pb*/235U"]:
    t = " = ["4corr-207Pb*/206Pb*] * ["4corr-206Pb*/238U"] * [Present 238U/235U] "
    --where SQUID 2.50 specifies [Present 238U/235U] = 137.88.
    PlaceFormulae t, f, piStdPb7U5_4col, L

    --Place the expression for ["4corr-207Pb*/235U %err"]:
    q = " =sqrt( ["4corr-206Pb*/238U %err"]^2 + ["4corr-207Pb*/206Pb* %err]^2 ) "
    PlaceFormulae q, f, piStdPb7U5_4eCol, L

    --Place the expression for ["4corr-errcorr"]:
    q = " = ["4corr-206Pb*/238U %err"] / ["4corr-207Pb*/235U %err"] "
    PlaceFormulae q, f, 2 + piStdPb6U8_4col, L
    --Note that "2 + piStdPb6U8_4col" corresponds to the "err corr" column.

  End If
```

Now all that is left in this subroutine is to perform a row-by-row assessment of the radiogenic ratios, to assess their usefulness for plotting on a Wetherill concordia diagram. Data-rows that have returned errors, or nonsensical numerical values, are marked for exclusion via application of Strikethrough font:

```
If (piStdPb7U5_4col > 0) AND (piStdPb6U8_4col > 0)
--i.e. if columns 204corr 207Pb*/235U and 204corr 206Pb*/238U both
--exist on StandardData:

  For i = f To L  --row-by-row assessment of recently calculated data
  ---with respect to its fitness for use on a Concordia diagram
    Set rw1 = Cells(i, piStdPb7U5_4col) --i.e. 207*/235 value
    Set rw2 = Cells(i, piStdPb6U8_4col) --i.e. so-called 206*/238 value
    Set r4 = Cells(i, 2 + piStdPb6U8_4col) --i.e. err corr value
```

```
    --frSr defines 5-column range in row i:
    With frSr(i, piStdPb7U5_4col, , 2 + piStdPb6U8_4col)

      If (IsError(rw1) = TRUE) OR (IsError(rw2) = TRUE) OR
      (rw1 < 1e-5) OR (rw2 < 1e-4) OR (r4 >= 1) OR (r4 <= 0)

        Font.Strikethrough = TRUE

      End If

    End With
  Next i

End If

End Sub
```

Note that this subroutine makes no attempt to calculate 208corr 207Pb/*235U (as could be legitimately required by Perm1 where index isotope = 208Pb, nor (by analogy) 208corr 207Pb*/206Pb*. Similarly, for the radiogenic daughter-parent pair, the focus is exclusively on 206Pb*/238U, whereas in Perm2 and Perm4, there ought to be an analogous calculation for 208Pb/*232Th (it's possible that this is done later, but there isn't much code left!). Furthermore, in Perm3, we see the calculation for 208Pb*/232Th (as the **sole** daughter-parent pair) deliberately inserted into the column labelled "206*/238", which really shouldn't happen, although I can see why Ludwig found it expedient.

So at some future point, it will be necessary to revisit these calculations to ensure that **all** the relevant calculations for each Perm are performed and tabulated. This should not require the invention of new arithmetic; it should rather just involve a wider range of analogous calculations in each Perm.

SQUID 2.50 Sub: SamRadiogenicCols

This subroutine (which is solely **for SampleData**) places, row-by-row, formulae to calculate radiogenic (i.e. corrected for common Pb) Pb/Pb, Pb/U, and Pb/Th ratios and ages in the SampleData sheet.

Usage

SamRadiogenicCols plaFirstDatRw, plaLastDatRw

Mandatory variables

plaFirstDatRw: Integer index number of the first row containing spot-by-spot data (for the Standard).
plaLastDatRw: Integer index number of the last row containing spot-by-spot data (for the Standard).

Definition of variables

Values of type Boolean
pbCalc8corrConcPlotRats, pbTh, pbU, Switch.DirectAltPD

Values of type Integer
f, L, m, piNumDauPar

Values of type String
d2, t5

Values of type Double
Alpha, Beta, Gamma, d1, d3, d4, d5, NetAlpha, NetBeta, NetGamma, radd6, radd8, sComm_68, t1, t3, TotPb6U8AbsErr, TotPb76AbsErr, TotPb8Th2AbsErr

---

The subroutine starts by calculating the Pb-isotope ratios used for the common-Pb correction. Note that these are not predicated on the user having specified "the right" numerator-denominator combinations as part of their Task definition: rather, SQUID 2.50 mandates that ratios ["204/206"], ["207/206"] and ["208/206"] are **always** calculated if the relevant mass-stations are present in the run-table, even if the user did not specifically request those ratios (i.e. they are evaluated 'in the background'). Define row-by-row values of Alpha, Beta and

Gamma:

```
Alpha = 1 / ["204/206"] --measured ["206/204"], i.e. total 206/204
Beta = ["207/206"] / ["204/206"] --measured ["207/204"], i.e. total 207/204
Gamma = ["208/206"] / ["204/206"] --measured ["208/204"], i.e. total 208/204
```

In each case, the measured (or total) Pb-isotope ratio can be considered as the sum of the **radiogenic** component and the **common** component. Using ["206/204"] as an example, R64 as shorthand for the ratio, and suffixes m, r, and c to denoted measured, common, and radiogenic components respectively: R64m = R64r + R64c. It is then possible to isolate the radiogenic component, by subtracting the common component from each measured value to generate row-by-row values of NetAlpha, NetBeta and NetGamma:

```
NetAlpha = Alpha - sComm_64 --i.e. R64r = R64m - R64c
NetBeta = Beta - sComm_74 --i.e. R74r = R74m - R74c
NetGamma = Gamma - sComm_68 --i.e. R84r = R84m - R84c
```

For the daughter isotopes of calibration-related interest (i.e 206Pb and 208Pb), it is possible to define, row-by-row, proportionality factors describing the radiogenic component as a fraction of the total measured value. The calculated values of radd6 and radd8 are used later in this subroutine:

```
radd6 = NetAlpha / Alpha --i.e. R64r / R64m
radd8 = NetGamma / Gamma --i.e. R84r / R84m
```

Start by calculating one or both of ["Total 206Pb/238U"] and/or ["Total 208Pb/232Th"] **where calibration constants are available**, as required by Perms1-4. Note that as per previously-documented subroutine StdRadiogenicCols, this arithmetic is predicated on having **only one** calibration constant per daughter-parent pair (following the user-defined index isotope for the common Pb correction), whereas SQUID 3.0 will be calculating multiple calibration constants corresponding to all the candidate index-isotope values. The following expressions will therefore require generalisation:

```
For i = 1 To piNumDauPar --recall piNumDauPar = 2 for Perm2 and Perm 4, else 1

  If ( (i = 1) AND (pbU = TRUE) ) OR ( (i = 2) AND (pbTh = TRUE) )
  -- first part of the If covers Perm1 and Perm2; the second part covers Perm4

    m = piPb6U8_totCol --index number for ["Total 206Pb/238U"] column
```

```
  --NOTE that SQUID 3.0 will need to key this to index-isotope, no matter how
  --counter-intuitive that seems!

ElseIf ( (i = 1) AND (pbTh = TRUE) ) OR ( (i = 2) AND (pbU = TRUE) )
-- first part of the If covers Perm3 and Perm4; the second part covers Perm2

  m = piPb8Th2_totCol --index number for ["Total 208Pb/232Th"] column
  --NOTE that SQUID 3.0 will need to key this to index-isotope, no matter how
  --counter-intuitive that seems!

Else

  m = 0 --should never happen

End If

--Now place formulae for so-called ["Total 206Pb/238U"] and/or so-called
--["Total 208Pb/232Th"]. Note that in SQUID 3.0, because we are performing
--calculations for ALL candidate index-isotopes, these calculations need to
--be generalised to give results like ["Xcorr Tot 206Pb/238U"] (where X can
--be 4 or 7 (Perm1,2,4) or 8 (Perm1 only)), or ["Ycorr Total 208Pb/232Th"]
--(where Y can be 4 or 7 (Perm2,3,4)).

--We will need to evaluate the Xcorr... results carefully, because it seems
--to me that there should not be much variation, and it occurs to me that
--the calculation would be more efficiently AND more correctly done by direct
--reference to our ["Uncorr Pb/U const."] and/or [Uncorr Pb/Th const."]
--columns that we calculated long ago (note that this approach would also
--yield the intuitively expected "single" ["Total 206Pb/238U"] and/or
--["Total 208Pb/232Th"]). But for the present, I will document Ludwig's
--calculations in SQUID 2.50.

--Calculate ["Total 206Pb/238U"] or ["Total 208Pb/232Th"]
If m = piPb6U8_totCol --Place row-by-row expression for ["Total 206Pb/238U"]

  If i = 1 --Perm1 and Perm2

    --Place row-by-row expression for ["Total 206Pb/238U"]
    PlaceFormulae " = ["Xcorr 206Pb/238U calibr. const."] / WtdMeanA1 *
      StdUPbRatio ", f, m, L --i.e. place expression in col m, rows f to L

    --Place row-by-row expression for ["Total 206Pb/238U %err"]
    PlaceFormulae " = SQRT( ["Xcorr 206Pb/238U calibr. const. %err"]^2
      + ExtPerrA1 ^ 2 ) ", f, m + 1, L
      --i.e. place expression in col m + 1, rows f to L
      --Recall that ExtPerrA is defined as max(R, S) where S is the ExtPerr
      --value (at 1-sigma, as a percentage) returned by the relevant
      --WtdMeanAcalc, and R is an arbitrary "minimum" value for S (R = 0.75%
      --at Geoscience Australia, by convention). There is more detail on this
      --at the very end of the documentation of subroutine WtdMeanAcalc: it
      --is described under extBox.

  ElseIf i = 2 --Perm4

    --Place row-by-row expression for ["Total 206Pb/238U"]
    PlaceFormulae " = ["Xcorr 206Pb/238U calibr. const."] / WtdMeanA2 *
```

```
        StdUPbRatio ", f, m, L --i.e. place expression in col m, rows f to L

      --Place row-by-row expression for ["Total 206Pb/238U %err"]
      PlaceFormulae " = SQRT( ["Xcorr 206Pb/238U calibr. const. %err"]^2
        + ExtPerrA2 ^ 2 ) ", f, m + 1, L
        --i.e. place expression in col m + 1, rows f to L

    End If --i = 1 or 2

  ElseIf m = piPb8Th2_totCol --Place expression for ["Total 208Pb/232Th"]

    If i = 1 --Perm3 and Perm4

      --Place row-by-row expression for ["Total 208Pb/232Th"]
      PlaceFormulae " = ["Xcorr 208Pb/232Th calibr. const."] / WtdMeanA1 *
        StdThPbRatio ", f, m, L --i.e. place expression in col m, rows f to L

      --Place row-by-row expression for ["Total 208Pb/232Th %err"]
      PlaceFormulae " = SQRT( ["Xcorr 208Pb/232Th calibr. const. %err"]^2
        + ExtPerrA1 ^ 2 ) ", f, m + 1, L
        --i.e. place expression in col m + 1, rows f to L

    ElseIf i = 2 --Perm2

      --Place row-by-row expression for ["Total 208Pb/232Th"]
      PlaceFormulae " = ["Xcorr 208Pb/232Th calibr. const."] / WtdMeanA2 *
        StdThPbRatio ", f, m, L --i.e. place expression in col m, rows f to L

      --Place row-by-row expression for ["Total 208Pb/232Th %err"]
      PlaceFormulae " = SQRT( ["Xcorr 208Pb/232Th calibr. const. %err"]^2
        + ExtPerrA2 ^ 2 ) ", f, m + 1, L
        --i.e. place expression in col m + 1, rows f to L

    End If --i = 1 or 2

  End If --m = piPb8Th2_totCol

Next i
```

Now calculate the corresponding ["Total 238U/206Pb"], via simple inversion:

```
If piPb6U8_totCol > 0 --i.e. if ["Total 206Pb/238U"] exists

  --Fill column ["Total 238U/206Pb"]:
  PlaceFormulae " = 1 / ["Total 206Pb/238U"] ", f, piU8Pb6_totCol, L

  --Fill column ["Total 238U/206Pb %err"]:
  PlaceFormulae " = ["Total 206Pb/238U %err"] ", f, piU8Pb6_TotEcol, L

End If
```

Now calculate the "secondary" daughter-parent isotopic ratio, for the permutations where no calibration constant is available. Specifically, this means calculating ["Total 208Pb/232Th"] for Perm1, and ["Total 206Pb/238U"] for Perm3. Note that the previously documented analogue of this subroutine for Standard data (Sub StdRadiogenicCols in SQUID 2.50) does **not** contain this step, but Squid3 should be revised to include it.

```
If (pbU = TRUE) AND (Switch.DirectAltPD = FALSE) --i.e. Perm1 only

  If piPb8Th2_totCol > 0  --i.e. if column ["Total 208Pb/232Th"] exists

    --Fill column ["Total 208Pb/232Th"] on SampleData:
    PlaceFormulae " = ["Total 206Pb/238U"] * ["208/206"] / ["232Th/238U"] ",
      f, piPb8Th2_totCol, L

    --Fill column ["Total 208Pb/232Th %err"] on SampleData:
    PlaceFormulae " = SQRT( ["208/206 %err"]^2 + ["Total 206Pb/238U %err"]^2 +
      ["232Th/238U %err"]^2 ) ", f, piPb8Th2_totEcol, L

  End If

ElseIf (pbTh = TRUE) AND (Switch.DirectAltPD = FALSE) AND (piPb6U8_totCol > 0)
--i.e. Perm3 only, and even then, only if column ["Total 206Pb/238U"] exists

  --Fill column ["Total 238U/206Pb"] on SampleData:
  PlaceFormulae " = ["Total 208Pb/232Th"] / ["208/206"] * ["232Th/238U"] ",
    f, piPb6U8_totCol, L

  --Fill column ["Total 238U/206Pb %err"] on SampleData:
  PlaceFormulae " = SQRT( ["208/206 %err"]^2 + ["Total 208Pb/232Th %err"]^2 +
      ["232Th/238U %err"]^2 ) ", f, piPb6U8_totEcol, L

End If
```

Now, if a uranium concentration reference material has been defined (i.e. column ["ppmU"] contains data), it is time to calculate and populate the columns containing the ppm values of radiogenic Pb, according to the various index isotope. There are a total of five permutations, comprising ["X-corr ppm 206*"], where X = 4, 7 or 8, and [Y-corr ppm 208*"], where Y = 4 or 7.

Note that SQUID 2.50 uses the 'magic number' 0.859 hard-coded, and for initial testing, this should be replicated. In terms of models of physical constants, however, the value 0.859 is derived via:

```
0.859 = ( 1 - ( 1 / Present238U235U ) * 206 / 238
```

where 206 is the atomic mass of 206Pb, 238 is the atomic mass of 238U, and Ludwig

assumes a value of Present238U235U of 137.88. More recently, other values of Present238U235U have been measured via EARTHTIME (e.g. 137.818 by Hiess et al., 2012), so the hard-coded value should ultimately be replaced by the expression containing Present238U235U above.

With this in mind, the SQUID 2.50 code populates each of these 5 column in turn:

```
If piaPpmUcol > 0 --i.e. ["ppmU"] exists on SampleData

  For p = 1 To 5

    Select Case p

      Case 1 --populate column ["4-corr ppm 206*"]
      PlaceFormulae " = ["Total 206Pb/238U"] * ["ppmU"] * 0.859 *
        ( 1 - ["204/206"] * sComm_64 ) ", f, piaRadDauCol_46, L

      Case 2 --populate column ["7-corr ppm 206*"]
      PlaceFormulae " = ["Total 206Pb/238U"] * ["ppmU"] * 0.859 *
        ( 1 - ["204/206 fr. 207"] * sComm_64 ) ", f, piaRadDauCol_76, L

      Case 3 --populate column ["8-corr ppm 206*"]
      PlaceFormulae " = ["Total 206Pb/238U"] * ["ppmU"] * 0.859 *
        ( 1 - ["204/206 fr. 208"] * sComm_64 ) ", f, piaRadDauCol_86, L

      Case 4 --populate column [4-corr ppm 208*"], noting that this calculation
          draws
      --on the row-by-row results determined in Case 1
      PlaceFormulae " = ["4-corr ppm 206*"] * ["4-corr 208Pb*/206Pb*"] * 208/206 ",
        f, piaRadDauCol_48, L
      --where 208 is the atomic mass of 208Pb and 206 is the atomic mass of 206Pb.

      Case 5 --populate column [7-corr ppm 208*"], noting that this calculation
          draws
      --on the row-by-row results determined in Case 2
      PlaceFormulae " = ["7-corr ppm 206*"] * ["7-corr 208Pb*/206Pb*"] * 208/206 ",
        f, piaRadDauCol_78, L
      --where 208 is the atomic mass of 208Pb and 206 is the atomic mass of 206Pb.

    End Select

  Next p

End If
```

If column ["4corr 206/*238"] exists, calculate 204Pb-corrected 206Pb*/238U values (and their uncertainties), and place them in columns ["4corr 206/*238"] and ["4corr 206*/238 %err"], along with their corresponding dates, which are placed in columns ["204corr 206Pb/238U Age"] and

["204corr 206Pb/238U Age 1serr"]. Strictly, these calculations are predicated on the presence of the column ["204/206"], but this code permits the calculation to proceed even if ["204/206"] does not exist (albeit by instead calculating an ["Total 206Pb/238U Age"]). Squid3 could (and probably should) calculate BOTH ["204corr 206Pb/238U Age"] and ["Total 206Pb/238U Age"] as a matter of course, just because we can:

```
If piPb6U8_4col > 0 --i.e. if column ["4corr 206*/238"] exists

  --Populate column ["4corr 206*/238"]:
  PlaceFormulae = " = ["Total 206Pb/238U"] * radd6 ", f, piPb6U8_4col, L

  --Now populate column ["4corr 206*/238 %err"]. Ludwig commented the expression:
  --Var(Rad6/8) = Var(Tot6/8) + (sComm_64/(Alpha-sComm_64))^2 * Var(Alpha)
  --which is correct in detail, although easier to read if rearranged:
  --Var(Rad6/8) = Var(Tot6/8) + Var(Alpha) * ( (sComm_64/(Alpha-sComm_64))^2 )
  --Defining ["4corr 206*/238 %err"] = sqrt[Var(Rad6/8)]:

  PlaceFormulae " = SQRT( ["Total 206Pb/238U %err"]^2 + ( sComm_64 *
    ["204/206 %err"] / ( 1 / ["204/206"] - sComm_64) )^2 ) ", f, piPb6U8_4ecol, L

  If piU8Pb6_4col > 0 --i.e. if column ["4corr 238/206*"] exists

    --Populate column ["4corr 238/206*"] via simple inversion:
    PlaceFormulae " = 1 / ["4corr 206*/238"] ", f, piU8Pb6_4col, L

    --Populate column ["4corr 238/206* %err"]:
    PlaceFormulae "= ["4corr 206*/238 %err"] ", f, piU8Pb6_4ecol, L

  End If

  --Now populate column ["204corr 206Pb/238U Age"]:
  PlaceFormulae " = LN( 1 + ["4corr 206*/238"] ) / Lambda238Ma ",
    f, piAgePb6U8_4col, L
  --where Lambda238Ma has units "Ma^-1", so Age will be in Ma.

  --Now calculate and populate column ["204corr 206Pb/238U Age 1serr"]:

  d1 = ( NetAlpha * ["Total 206Pb/238U %err"] / 100 )^2
  d3 = ( ["204/206 %err"] * sComm_64 / 100 )^2
  d4 = ( ["Total 206Pb/238U"] * ["204/206"] )^2
  d5 = ( 1 / Lambda238Ma /
         EXP( Lambda238Ma * ["204corr 206Pb/238U Age"] ) )^2
  --where Lambda238Ma has units "Ma^-1".

  PlaceFormulae " = SQRT( d5 * d4 * (d1 + d3) ) ", f, piAgePb6U8_4ecol, L

ElseIf (piPb46col = 0) AND (piAgePb6U8_4col > 0) --this curious ElseIf
--covers the possibility that a column ["204corr 206Pb/238U Age"] might
--exist even in the absence of column ["204/206"], which would
--ordinarily be a prerequisite for the calculation. In this scenario,
--Ludwig simply calculates the Age from the **uncorrected** 206/238,
--and modifies the relevant column-header accordingly.
```

```
    --Populate the column **designated** ["204corr 206Pb/238U Age"] with
    --an expression **reflecting** ["Total 206Pb/238U Age"]:
    PlaceFormulae " = LN( 1 + ["Total 206Pb/238U"] ) / Lambda238Ma,
      f, piAgePb6U8_4col, L
    --where Lambda238Ma has units "Ma^-1", so Age will be in Ma.

    --...and then change its header to ["Total 206Pb/238U Age"]!!
    CFs plHdrRw, piAgePb6U8_4col, "total|206Pb|/238U|age", TRUE

    --Now populate column **designated** ["204corr 206Pb/238U Age 1serr"]
    --with an expression **reflecting** ["Total 206Pb/238U Age 1serr"]:
    PlaceFormulae " = ["Total 206Pb/238U"] / Lambda238Ma /
      (1 + ["Total 206Pb/238U"] ) * ["Total 206Pb/238U %err"] / 100 ",
      f, piAgePb6U8_4ecol, L
    --where Lambda238Ma has units "Ma^-1", so Age 1serr will be in Ma.

  End If --piPb6U8_4col > 0 --i.e. if column ["4corr 206*/238"] exists.
```

The next step is to perform the superset of calculations enabled by the measurement of ["207/206"]. These start with 4corr 207/*206* analogous to the 206/*238 calculations described above, but also encompass 4corr 207*/235, plus 207Pb-corrected 206*/238. The code continues:

```
  If piPb76col > 0 --i.e. if column ["207/206"] exists

    If piU8Pb6_totCol > 0 --i.e. if column ["Total 238U/206Pb"] exists

      --Place ["Total 207Pb/206Pb"]; used for Tera-Wasserburg plots and calculations
      PlaceFormulae "= ["207/206"] ", f, piPb76_totCol, L

      --Place ["Total 207Pb/206Pb %err"]
      PlaceFormulae " = ["207/206 %err"] ", f, piPb76_totEcol, L

    End If --column ["Total 238U/206Pb"] exists

    If (piPb46col > 0) AND (piU8Pb6_4col > 0) --i.e. if BOTH columns
    --["204/206"] and ["4corr 238/206*"] exist

      --Populate column ["4corr 207*/206*"]
      PlaceFormulae " ABS( NetBeta / NetAlpha ") ",
        f, piPb76_4col, L

      --Populate column ["4corr 207*/206* %err"]
      --Complicated expression; note that it uses ["4corr 207*/206*"]:

      t1 = ( ( ["207/206"] - ["4corr 207*/206*"] ) * ["204/206 %err"]
             / 100 / ["204/206"] )^2
      t3 = ( ["207/206 %err"] / ["204/206"] / 100 * ["207/206"] )^2

      PlaceFormulae " = ABS( SQRT( t1 + t3 ) / NetAlpha * 100 /
```

```
      ["4corr 207*/206*"] ) ",
     f, piPb76_4eCol, L

  If piaAgePb76_4Col > 0 --i.e. if column ["204corr 207Pb/206Pb Age"] exists,
  --the form of the next calculations depend on whether column ["204/206"]
  --exists, analogous to the previous calculation of 206*/238 (i.e. if **NO**
  --["204/206"] exists, a ["Total 207Pb/206Pb Age"] will be calculated; else
  --["204corr 207Pb/206Pb Age"] will be calculated as would be expected.

    If piPb46col > 0 --i.e. if column ["204/206"] exists

      --Place LudwigLibrary expression for ["204corr 207Pb/206Pb Age"]:
      PlaceFormulae " = AgePb76( ["4corr 207*/206*"] ) ",
        f, piaAgePb76_4Col, L

      --Place LudwigLibrary expression for ["204corr 207Pb/206Pb Age 1serr"]:
      PlaceFormulae " = AgeErPb76( ["4corr 207*/206*"],["4corr 207*/206* %err"]
        ,,,, TRUE ) ", f, piaAgePb76_4eCol, L

    Else --i.e. if column ["204/206"] does *NOT* exist. Note that this Else
    --appears to be redundant, as this If loop cannot be traversed unless BOTH
    --["204/206"] and ["4corr 238/206*"] exist, as per If clause near the top
    --of this block:

      --Place LudwigLibrary expression for ["Total 207Pb/206Pb Age"]
      --in the column **designated** ["204corr 207Pb/206Pb Age"]...
      PlaceFormulae " = AgePb76( ["Total 207/206"] ) ",
        f, piaAgePb76_4Col, L

      --...and then change the column-header to reflect the contents!
      CFs plHdrRw, piaAgePb76_4Col, ["Total 207Pb/206Pb Age"], TRUE

      --Place LudwigLibrary expression for ["Total 207Pb/206Pb Age 1serr"]
      --in the column **designated** ["204corr 207Pb/206Pb Age 1serr"]:
      PlaceFormulae " = AgeErPb76( ["Total 207*/206*"],["Total 207*/206* %err"]
        ,,,,TRUE ) ", f, piaAgePb76_4eCol, L

    End If --piPb46col > 0 --i.e. if column ["204/206"] exists

  End If --piaAgePb76_4Col > 0 --i.e. if ["204corr 207Pb/206Pb Age"] exists

  --Now place expressions for ["4corr 207*/235"] and ["4corr 207*/235 %err"]:

  If (piPb7U5_4col > 0) AND (piPb6U8_4col > 0) --i.e. if BOTH columns
  --["4corr 207*/235"] and ["4corr 206*/238"] exist

    --Calculate and place ["4corr 207*/235"]:
    PlaceFormulae " = ["4corr 207*/206*"] * ["4corr 206*/238"] * Present238U235U
      f, piPb7U5_4col, L

    --Calculate and place ["4corr 207*/235 %err"]:
    PlaceFormulae " = SQRT( ["4corr 207*/206* %err"]^2 +
      ["4corr 206*/238 %err"]^2 ), f, piPb7U5_4ecol, L

    --Calculate and place rho (error correlation) value ["4corr err corr"]:
    PlaceFormulae " = ["4corr 206*/238 %err"] / ["4corr 207*/235 %err"] ",
```

```
        f, piPb7U5Pb6U8_4rhoCol, L

    End If --BOTH columns ["4corr 207*/235"] and ["4corr 206*/238"] exist

  End If --BOTH columns ["204/206"] and ["4corr 238/206*"] exist

  --Now calculate and place ["204corr Discordance"], noting that SQUID 2.50
  --column-headers do not specify the index isotope prefix:

  If (piDiscordCol > 0) AND (piPb46col > 0) --i.e. if BOTH columns
  --["204corr Discordance"] and ["204/206"] exist

    --SQUID 2.50 defines Discordance as 100 * ( 1 - R68m / R68i ) where
    --R68i is the concordant 206*/238 RATIO (corresponding to the measured
    --207*/206* RATIO) and R68m is the measured 206*/238 RATIO. First calculate
    --the row-by-row values of (204corr) R68i:

    R68i = EXP( Lambda238Ma * ["204corr 207Pb/206Pb Age"] ) - 1

    --Then place the expression for ["204corr Discordance"]:
    PlaceFormulae " = 100 * ( 1 - ["4corr 206*/238"] / R68i ) ",
      f, piDiscordCol, L

  End If --BOTH columns ["204corr Discordance"] and ["204/206"] exist

  --Now calculate and place ["207corr 206Pb/238U Age"] and its uncertainty.
  --Note that this code does *NOT* encompass the ["7corr 206*/238"] RATIO, but it
  --SHOULD; the relevant code is misplaced a little further along. Below I have
  --marked the spot at which ["7corr 206*/238"] and ["7corr 206*/238 %err"]
  --SHOULD be evaluated!

  If piAgePb6U8_7col > 0 --i.e. if column ["207corr 206Pb/238U Age"] exists

    --Place LudwigLibrary expression for ["207corr 206Pb/238U Age"]:
    PlaceFormulae " = Age7corr( ["Total 206Pb/238U"], ["207/206"], sComm_76 ) ",
      f, piAgePb6U8_7col, L

    --Place LudwigLibrary expression for ["207corr 206Pb/238U Age 1serr"], noting
    --that Ludwig function AgeEr7corr has a total of seven arguments. Two of these
    --arguments are *ABSOLUTE* uncertainties, which I have evaluated in advance,
    --for clarity. Calculate TotPb6U8AbsErr and TotPb76AbsErr as:

    TotPb6U8AbsErr = ["Total 206Pb/238U %err"] / 100 * ["Total 206Pb/238U"]
    TotPb76AbsErr = ["Total 207Pb/206Pb %err"] / 100 * ["Total 207Pb/206Pb"]

    PlaceFormulae " = AgeEr 7corr( ["207corr 206Pb/238U Age"],
      ["Total 206Pb/238U"], TotPb6U8AbsErr, ["207/206"], TotPb76AbsErr,
      sComm_76, 0 ) ", f, piAgePb6U8_7ecol, L

    --This is the spot at which ["7corr 206*/238"] and ["7corr 206*/238 %err"]
    --SHOULD be evaluated! The relevant lines are marked in the next code-block.

  End If --piAgePb6U8_7col > 0 --i.e. if ["207corr 206Pb/238U Age"] exists

End If --piPb76col > 0 --i.e. if column ["207/206"] exists
```

The next step is to calculate Xcorr 208Pb/232Th ratios, dates and uncertainties (where X = 4 or 7). The 204Pb-corrected calculations are analogous to those for 206/*238 and 207*/206. *Note also that the following code contains an out-of-place block relating to [*"*7corr 206*/238"] and ["7corr 206*/238 %err"]. This is undesirable at two levels: (1) it should have been evaluated in the previous code-block, subject to the If conditions that are relevant to the calculation, and (2) The If conditions employed below are not appropriate to the calculation: they could result in the evaluation being skipped for no good reason. The code continues:

```
If (piPb8Th2_totCol > 0) AND (piPb86col > 0) --i.e. if BOTH columns
--["Total 208Pb/232Th"] and ["208/206"] exist

  If piPb46col > 0 --i.e. if column ["204/206"] exists

    --Place the expression for ["4corr 208*/232"]:
    PlaceFormulae " = ["Total 208Pb/232Th"] * radd8 ", f, piPb8Th2_4col, L

    --Place the expression for ["4corr 208*/232 %err"]. Ludwig notes that this
    --expression "neglects the 208/206 error":
    PlaceFormulae " = SQRT( ["Total 208Pb/232Th %err"]^2 +
      ( sComm_84 / NetGamma )^2 * ["204/206 %err"]^2 )", f, piPb8Th2_4eCol, L

  End If

  If piPb46col > 0 --i.e. if column ["204/206"] exists

    --Place expression for ["204corr 208Pb/232Th Age"]:
    PlaceFormulae " = LN( 1 + ["4corr 208*/232"] ) / Lambda232Ma ",
      f, piAgePb8Th2_4col, L
    --where Lambda232Ma has units "Ma^-1", so Age will be in Ma.

    --Place expression for ["204corr 208Pb/232Th Age 1serr"]:
     PlaceFormulae " = ["4corr 208*/232"] ) / Lambda232Ma /
       (1 + ["4corr 208*/232"] ) * ["4corr 208*/232 %err"] / 100 ",
       f, piAgePb8Th2_4eCol, L
    --where Lambda232Ma has units "Ma^-1", so Age 1serr will be in Ma.

  Else --i.e. if column ["204/206"] does *NOT* exist

    --Place expression for ["Total 208Pb/232Th Age"] in the column
    --**designated** ["204corr 208Pb/232Th Age"]...
    PlaceFormulae " = LN( 1 + ["Total 208Pb/232Th"] ) / Lambda232Ma ",
      f, piAgePb8Th2_4col, L

    --...and then change the column-header to reflect the contents!
    CFs plHdrRw, piAgePb8Th2_4col, ["Total 208Pb/232Th Age"], TRUE

    --Place expression for ["Total 208Pb/232Th Age 1serr"] in the column
    --**designated** ["204corr 208Pb/232Th Age 1serr"]:
    PlaceFormulae " = ["Total 208Pb/232Th"] ) / Lambda232Ma /
```

```
      (1 + ["Total 208Pb/232Th"] ) * ["Total 208Pb/232Th %err"] / 100 ",
      f, piAgePb8Th2_4eCol, L

End If --piPb46col > 0 --i.e. if column ["204/206"] exists

--Ludwig's next If block is superfluous: it repeats the evaluation of
--["4corr 208*/232 %err"], adding an unnecessary Boolean, and an absolute
--value of NetGamma, which is moot as the NetGamma-bearing term is squared
--anyway. So I have commented-out the following lines:
--
--If (pbTh = TRUE) AND (piPb46col > 0) --i.e. if (Perm3 or Perm4) AND column
----["204/206"] exists:
--
--  PlaceFormulae " = SQRT( ["Total 208Pb/232Th %err"]^2 +
--    ( sComm_84 / ABS( NetGamma ) )^2 * ["204/206 %err"]^2 )",
--    f, piPb8Th2_4eCol, L
--
--End If

--Ludwig's next If block is OUT OF PLACE; it should have appeared at the base
--of the previous code-block, in the location I have marked there. I have
--included the code below, but I have inserted some comments of my own.
--**NOTE ALSO** that Ludwig's code below also contained an arithmetical bug:
--he used the wrong Lambda at one point, which results the value being too small
--by a factor of about 3!!

If piAgePb6U8_7col > 0 --i.e. if column ["207corr 206Pb/238U Age"] exists. This
--criterion on its own is appropriate for the following calculation, and it is
--also consistent with the If criterion already in place at the proper
--destination (shown in the previous code-block) for the following expressions.
--But note that this criterion is **NOT** appropriate in combination with the
--2-column If criteria applied at the top of the current code-block. As written,
--the following calculation would not be performed if ["208/206"] was not
--measured, which is ridiculous.

  --CODE TO BE TRANSPLANTED TO PREVIOUS CODE-BLOCK STARTS **HERE**
  --Place expression for ["7corr 206*/238"]:
  PlaceFormulae " = EXP ( Lambda238Ma * ["207corr 206Pb/238U Age"] ) - 1 ",
    f, piPb6U8_7col, L
  --where Lambda238Ma has units "Ma^-1".

  --Place expression for ["7corr 206*/238 %err"]. Ludwig's code contains a bug
  --in the following line (commented out), where his expression incorrectly
  --commences with Lambda232Ma:
  --PlaceFormulae " = Lambda232Ma * EXP( Lambda238Ma *
        ["207corr 206Pb/238U Age"] ) *
  --  ["207corr 206Pb/238U Age 1serr"] / ["7corr 206*/238"] * 100 ",
  --  f, piPb6U8_7ecol, L

  --The corrected expression commences with Lambda238Ma:
  PlaceFormulae " = Lambda238Ma * EXP( Lambda238Ma *
        ["207corr 206Pb/238U Age"] ) *
    ["207corr 206Pb/238U Age 1serr"] / ["7corr 206*/238"] * 100 ", f,
      piPb6U8_7ecol, L
  --CODE TO BE TRANSPLANTED TO PREVIOUS CODE-BLOCK ENDS **HERE**
```

```
    End If --piAgePb6U8_7col > 0 i.e. if column ["207corr 206Pb/238U Age"] exists.
    --Code transplantation should result in abolition of the previous If.

    --Now perform calculations related to ["7corr 208*/232"]:
    If piAgePb8Th2_7col > 0 --i.e. if column ["207corr 208Pb/232Th Age"] exists:

      --Place LudwigLibrary expression for ["207corr 208Pb/232Th Age"], which has a
      --total of 7 arguments:
      PlaceFormulae " = Age7corrPb8Th2( ["Total 206Pb/238"], ["Total 208Pb/232Th"],
        ["208/206"], ["207/206"], sComm_64, sComm_76, sComm_86 ) ",
        f, piAgePb8Th2_7col, L

      --Place LudwigLibrary expression for ["207corr 208Pb/232Th Age 1serr"], which
      --has a total of 11 arguments (i.e. includes %err values for the first four
      --arguments of the previous function). Note also that the input order of
      --["207/206"] and ["208/206"] is *REVERSED* in this second function!
      PlaceFormulae " = AgeErr7corrPb8Th2( ["Total 206Pb/238"],
        ["Total 206Pb/238 %err"],["Total 208Pb/232Th"], ["Total 208Pb/232Th %err"],
        ["207/206"], ["207/206 %err"], ["208/206"], ["208/206 %err"], sComm_64,
        sComm_76, sComm_86 ) ", f, piAgePb8Th2_7ecol, L

      --Place expression for ["7corr 208*/232"]:
      PlaceFormulae " = EXP ( Lambda232Ma * ["207corr 208Pb/232Th Age"] ) - 1 ",
        f, piPb8Th2_7col, L
      --where Lambda232Ma has units "Ma^-1".

      --Place expression for ["7corr 208*/232 %err"]:
      PlaceFormulae " = Lambda232Ma * EXP( Lambda232Ma *
            ["207corr 208Pb/232Th Age"] ) *
        ["207corr 208Pb/232Th Age 1serr"] / ["7corr 208*/232"] * 100 ",
        f, piPb8Th2_7ecol, L

    End If --piAgePb8Th2_7col > 0 i.e. if column ["207corr 208Pb/232Th Age"] exists
```

The next thing SQUID 2.50 does is calculate ["208corr 206Pb/238U Age"] and its uncertainty. As for the 207corr equivalent, this is a hangover from SQUID 1, where these two ages and their errors were presented *without* the actual source ratios! SQID 2.50 rectifies that, but only "in reverse" (i.e. it still calculates the Age first, and then derives the ratios from the Ages). *Note* that the following If condition seems irrelevant at best (incorrect at worst), as neither ["204/206"] nor any of its dependents are required in order to perform the LudwigLibrary calculations 'Age8Corr' or 'AgeEr8Corr'. I think it is possible that the condition should read "If piPb86col > 0" (i.e. if column ["208/206"] exists), because that would make more sense, but *note* that piPb86col > 0 is already a precondition of the entire loop, and so would be redundant here.

After some deliberation, I have left the If condition (piPb4col > 0) as it was specified by Ludwig. One possibility is that the If condition should be removed entirely; perhaps these particular calculations should be mandatory if the foregoing If conditions have been satisfied.

For use as arguments in ensuing LudwigLibrary functions, first define sComm_68 as the inverse of sComm_86, and TotPb8Th2AbsErr as the Th-Pb equivalent of the previously defined TotPb6U8AbsErr, and then proceed:

```
sComm_68 = 1 / sComm_86
TotPb8Th2AbsErr = ["Total 208Pb/232Th %err"] / 100 * ["Total 208Pb/232Th"]

--If piPb46col > 0 --i.e. if ["204/206"] exists, although this doesn't make sense

  --Place LudwigLibrary expression for ["208corr 206Pb/238U Age"]:
  PlaceFormulae " = Age8Corr( ["Total 206Pb/238U"], ["Total 208Pb/232Th"],
    ["232Th/238U"], sComm_68 ) ", f, piAgePb6U8_8col, L

  --Place LudwigLibrary expression for ["208corr 206Pb/238U Age 1serr"], noting
  --that the function has a total of 9 arguments:
   PlaceFormulae " = AgeEr8Corr( ["208corr 206Pb/238U Age"],["Total 206Pb/238U"],
     TotPb6U8AbsErr, ["Total 208Pb/232Th"], TotPb8Th2AbsErr, ["232Th/238U"], 0,
     sComm_68, 0 ) ", f, piAgePb6U8_8ecol, L

End If --piPb46col > 0 --i.e. if ["204/206"] exists.
```

SQUID 2.50 has a user-controlled Boolean (pbCalc8corrConcPlotRats) which controls whether the 208corr isotopic ratios required for Wetherill and Tera-Wasserburg concordia plots are calculated (possibly because they frequently give results that are ridiculous or impossible in the case of high-Th analyses). But in Squid3, I would think that errors in calculations would be handled separately, and that this Boolean ought to at least be TRUE by default, if not hard-coded TRUE. Then, if column ["208corr 206Pb/238U Age"] exists, the code continues:

```
If (pbCalc8corrConcPlotRats = TRUE) AND (piAgePb6U8_8col > 0)
--i.e. if user has requested 208corr Concordia ratios AND the column
--["208corr 206Pb/238U Age"] exists:

  If piU8Pb6_8col > 0 --i.e. if column ["8corr 238/206*"] exists

    --PLace LudwigLibrary expression for ["8corr 206*/238"] first:
    PlaceFormulae " = Pb206U238rad( ["208corr 206Pb/238U Age"] ) ",
      f, piPb6U8_8col, L

    --Then place expression for ["8corr 206*/238 %err"]:
    PlaceFormulae " = Lambda238Ma * ( 1 + ["8corr 206*/238"] ) *
      ["208corr 206Pb/238U Age 1serr"] * 100 / ["8corr 206*/238"] ",
      f, piPb6U8_8ecol, L

    --Then place expression for ["8corr 238/206*"] by simple inversion:
    PlaceFormulae " = 1 / ["8corr 206*/238"] ", f, piU8Pb6_8col, L

    --Finally, ["8corr 238/206* %err"] is equivalent to ["8corr 206*/238 %err"]:
```

```
      PlaceFormulae " = ["8corr 206*/238 %err"] ", f, piU8Pb6_8ecol, L

    End If --piU8Pb6_8col > 0 --i.e. if column ["8corr 238/206*"] exists

    If piPb7U5_8col > 0  --i.e. if column ["8corr 207*/235"] exists

      --Place LudwigLibrary expression for ["8corr 207*/235"]:
      PlaceFormulae " = Rad8corPb7U5(["208corr 206Pb/238U Age"],["Total 206Pb/238U"
        ["207/206"], sComm_76 ) ", f, piPb7U5_8col, L

      --Define row-by-row TotPb7U5 as follows (noting that SQUID 2.50 hard-codes a
      --numeric value of 137.88 in lieu of physical constant Present238U235U):
      TotPb7U5 = ["Total 206Pb/238U"] * ["207/206"] / Present238U235U

      --Now place LudwigLibrary expression for [8corr 207*/235 %err"], noting that
      --the function has a total of 12 arguments:
      PlaceFormulae " = Rad8corPb7U5perr( ["Total 206Pb/238U"],
        ["Total 206Pb/238U %err"], ["8corr 206*/238"], TotPb7U5, ["232Th/238U"],
        ["232Th/238U %err"], ["207/206"], ["207/206 %err"], ["208/206"],
        ["208/206 %err"], sComm_76, sComm_86 ) ", f, piPb7U5_8ecol, L

      --Now place LudwigLibrary expression for [8corr err corr"], noting that the
      --function has a total of 11 arguments (as per previous, without TotPb7U5):
      PlaceFormulae " = Rad8corConcRho( ["Total 206Pb/238U"], ["Total 206Pb/238U %e
        ["8corr 206*/238"], ["232Th/238U"], ["232Th/238U %err"], ["207/206"],
        ["207/206 %err"], ["208/206"], ["208/206 %err"], sComm_76, sComm_86 ) ",
        f, piPb7U5Pb6U8_8rhoCol, L

    End If --piPb7U5_8col > 0  --i.e. if column ["8corr 207*/235"] exists
```

The final step in this loop is to evaluate ["8corr 207/*206*"] and its uncertainty, and then use those to generate ["208corr 207Pb/206Pb Age"] and its error:

```
    If piPb76_8col > 0 --i.e. if column ["8corr 207*/206*"] exists

      --Place expression for ["8corr 207*/206*"], noting that once again SQUID 2.50
      --uses the hard-coded numeric value 137.88 in lieu of Present238U235U::
      PlaceFormulae " = ["8corr 207*/235"] / ["8corr 206*/238"] / Present238U235U "
        f, piPb76_8col, L

      --Place expression for ["8corr 207*/206* %err"]:
      PlaceFormulae " = SQRT( ["8corr 207*/235 %err"]^2 +
          ["8corr 206*/238 %err"]^2 -
        2 * ["8corr 207*/235 %err"] * ["8corr 206*/238 %err"]
            * ["8corr err corr"] ) ",
        f, piPb76_8ecol, L

    End If --piPb76_8col > 0 --i.e. if column ["8corr 207*/206*"] exists

    If piAgePb76_8col > 0 --i.e. if column ["208corr 207Pb/206Pb Age"] exists
```

```
        --Place LudwigLibrary expression for ["208corr 207Pb/206Pb Age"]:
        PlaceFormulae " = AgePb76( ["8corr 207*/206*"] ) ", f, piAgePb76_8col, L

        --Place LudwigLibrary expression for ["208corr 207Pb/206Pb Age 1serr"]:
        PlaceFormulae " = AgeErPb76( ["8corr 207*/206*"], ["8corr 207*/206* %err"]
          ,,,, TRUE ) ", f, piAgePb76_8ecol, L

        --PLACE EXPRESSION FOR 208CORR DISCORDANCE (SQUID 2.50 DOES NOT EVALUATE IT)
        --INSERT IDENTICAL EXPRESSION TO 204CORR DISCORDANCE, BUIT WITH 8CORR RATIOS
        --AND AGES SUBSTITUTED!

      End If --piAgePb76_8col > 0 --i.e. column ["208corr 207Pb/206Pb Age"] exists

    End If --(pbCalc8corrConcPlotRats = TRUE) AND (piAgePb6U8_8col > 0)
    --i.e. if user has requested 208corr Concordia ratios AND the column
    --["208corr 206Pb/238U Age"] exists

End If --If (piPb8Th2_totCol > 0) AND (piPb86col > 0) --i.e. if BOTH columns
--["Total 208Pb/232Th"] and ["208/206"] exist

--The subroutine finishes with some formatting to colour the rows corresponding to
--the Uranium concentration reference material pale yellow.

End Sub
```