Poisson Uncertainty of JADE data

Poisson statistics: 1-standard deviation uncertainty is the square root of the total number of counts. But if zero counts were measured, the uncertainty is ±1 count (technically $^{+1}_{-0}$ as can not be negative, but level 3 files only provide a single number, hence ±1).

Yet we have two further sources of uncertainty on total number of counts (and all these are independent of each other):

- Variance in counts due to Look Up Table (LUT) compression VAR_{LUT}:
  If zero counts there is no lossy compression in the LUT, so $VAR_{LUT} = 0$.
- A variance in number of views, m, of VAR_m. If m is known then $VAR_m = 0$.

Counts per Accumulation product.

DATA product returns total counts, compressed by a LUT: $VAR_{Total\_Counts} = DATA$

Variance of VAR_{Total_Counts} is just the uncertainty on the total number of counts, which is only due to the LUT compression, $DATA \pm \sqrt{VAR_{LUT}}$

Variances are summed together to give:

$$DATA = DATA \ \pm \ \sqrt{VAR_{Counts} + VAR_{LUT}} = DATA \ \pm \ \sqrt{DATA + VAR_{LUT}}$$

=> The combined standard uncertainty for a 'counts per accumulation' product is:

$$\frac{\text{Counts}}{\text{Accumulation}} = \begin{cases} DATA \pm \sqrt{\text{DATA}+VAR_{LUT}} & \text{if } DATA > 0 \\ 0 \pm 1 & \text{if } DATA = 0 \end{cases}$$

Counts per View product. (Rate product)

DATA product returns counts per view, compressed by a LUT. There are m views.
Total number of counts (required for Poisson Statistics) is the counts per view, multiplied by number of views: $VAR_{Total\_Counts} = m\ DATA$
Uncertainty per view requires dividing the standard deviation by m, so divided the variance by m^2:

$$VAR_{Views} = \frac{m\ DATA}{m^2} = \frac{DATA}{m}$$

Variance of VAR_{Views} requires propagating the uncertainty of $DATA$ and m, with uncertainty of VAR_{LUT} and VAR_m respectively. Variance of $DATA/m$ is then:

$$VAR_{DATA/m} = \frac{DATA^2}{m^2}\left(\frac{VAR_{LUT}}{DATA^2} + \frac{VAR_m}{m^2}\right) = \frac{VAR_{LUT}}{m^2} + \frac{DATA^2\ VAR_m}{m^4}$$

If zero counts, then for total counts for Poisson statistics, $m\ DATA = 1$ so: $VAR_{Views} = \frac{1}{m^2}$
Variance of $VAR\__{Views}$ requires propagating the uncertainty of m^{-2}.

$$\frac{\text{Counts}}{\text{View}} = \begin{cases} DATA \pm \sqrt{\frac{DATA}{m} + \frac{VAR_{LUT}}{m^2} + \frac{DATA^2\ VAR_m}{m^4}} & \text{if } DATA > 0 \\ 0 \pm \sqrt{\frac{1}{m^2} + \frac{2\ VAR_m}{m^6}} & \text{if } DATA = 0 \end{cases}$$

When it is known that $m = 1$ (=> $VAR_m = 0$) then this collapses to the Counts/accumulation equation.
For this version 1 approach, it is assumed that $VAR_m = 0$, simplified equations are shown later.

See https://en.wikipedia.org/wiki/Propagation_of_uncertainty for propagation equations.

If DATA object contains MISSING_CONSTANT values

If the DATA value is a MISSING_CONSTANT (fill) value, then the uncertainty should be returned as a MISSING_CONSTANT too. This makes coding more complicated, as there are several reasons why the DATA object could contain MISSING_CONSTANT values. E.g. HRS_ELC_ALL contains data from all three electron sensors, however when E300 is off, then the E300 entries are all MISSING_CONSTANT values, but the E060 and E180 entries are not. Alternatively, for JADE-I records, there may be a ping without a pong, or vice versa, hence half the 64 energy steps must remain MISSING_CONSTANT. As such, be careful not to treat a MISSING_CONSTANT value (which differs depending on product) as if it was valid DATA.

VAR_{LUT}

The VAR_{LUT} term relates to the onboard lossy compression and for each look up table is provided in the LUT_*m*_*mm*_COMPRESSION.CSV files as one of the following 4 columns:

Object in file (form L3_*n*_*nn*_??? in file LUT_*m*_*mm*_COMPRESSION.CSV)	**Data Mode**	**Info**
L3_16_1_VAR	HRS	16 to 8 bit variance for compression for electrons
L3_16_2_VAR	HRS	16 to 8 bit variance for compression for ions
L3_32_1_VAR	LRS/CAL	32 to 8 bit variance for compression for electrons
L3_32_2_VAR	LRS/CAL	32 to 8 bit variance for compression for ions

Which LUT_*m*_*mm*_COMPRESSION.CSV files to use may be found from the level 2 data files in the TABLES_VERSION object for each level 2 record. If TABLES_VERSION = 3.06 then use LUT_3_06_COMPRESSION.CSV, etc. Note that TABLES_VERSION is a value to two decimal places, but due to rounding errors may show up in IDL as 3.0599999, which obviously means 3.06. At time of writing there are LUTS 3.00, 3.01, 3.02, 3.03, 3.04, 3.05, 3.06, 3.07, 3.08 and 3.09 – which so far the LUT_*m*_*mm*_COMPRESSION.CSV files are identical for all, however it is possible later LUTs may contain different values to optimize them once we've got a feeling for general plasma conditions at Jupiter.

However it is not as simple as taking the DATA value and cross-referencing the L3_*n*_*nn*_VAR from the corresponding L3_nn_n_REP column. The onboard data has a MIN_SUBTRACTED_VALUE removed before lossy compression, which is also reported in the JADE Level 2 files. For JADE-I, there is a different value for each Level 2 ping and pong record (32 energy steps each). In addition, values onboard for LRS_ELC_ANY, LRS_ION_ANY and CAL_ION_ANY are multiplied by 512 before lossy compression. All these effects must be taken in to account in order to get a value equivalent to that when underwent lossy compression onboard and thus provided the VAR_{LUT} term.

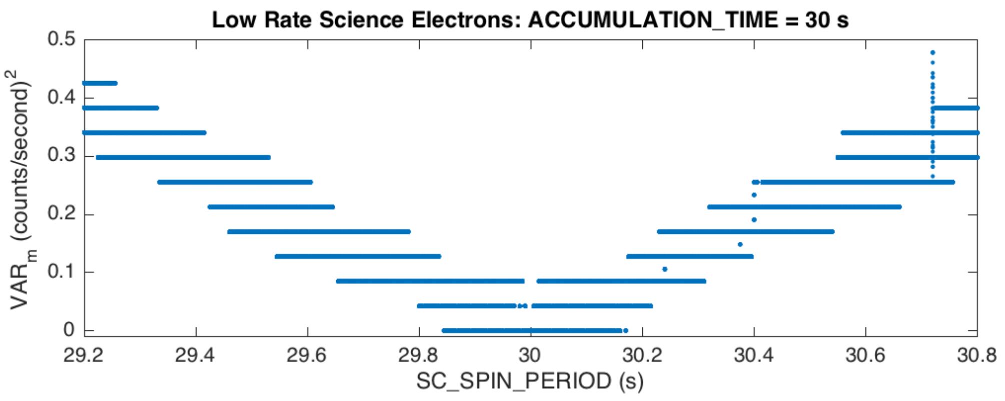
IDL code is shown later and describes exactly how the uncertainties were generated for the level 3 products for this version 1 method of uncertainties; that is the ultimate documentation source.

VAR_m

The uncertainty on number of views for a given spin-phase sector of a low rate science product is surprisingly tricky to work out. It is dependent on three items: the total accumulation period, the current Juno spin period for that record, and the spin-phase at the start of the record. The former is provided in the level 2 data record as ACCUMULATION_TIME. However the spin period can vary from 29.2 to 30.8 seconds for the Juno propulsion folk to consider it an acceptable "2 RPM", and its value must be calculated from reconstructed SPICE kernels. The start spin-phase is based on the onboard reported spin-phase to JADE, which is an unsigned 32-bit value representing 0 to 2π, that is 4294967296 different values, each 83.82 nano-degrees apart. The start spin-phase may also be calculated using reconstructed SPICE kernels, however it is not known if that accurately reflects the reported onboard spin-phase to the same 32-bit resolution – as such it also has an unknown uncertainty. The other option would be to use the SHK housekeeping data file that lists an onboard spin-phase (spin-phase to next 'north' crossing rather than from the last 'north' crossing) and time when it was registered. However the cadence of these records are different to that of the science products, and even when plotting the values against time it does not flow linearly but with a slight wave, suggesting that although the onboard value is to 32-bit resolution, it is not reported to that.

A possible solution is two-fold:
- If ACCUMULATION_TIME was truncated (Level 2 object ACCUM_TRUNCATION = 1) then we cannot calculate VAR_m, therefore set returned uncertainty to the MISSING_CONSTANT value for all elements.
- Otherwise, use the worst-case VAR_m independent of start spin-phase and spin-period.

Calculating the worst-case VAR_m may be done by brute force. For a fixed spin period and ACCUMULATION_TIME, altering the start spin-phase value tended to provide a few unique VAR_m values, of which we take the largest. Such calculations take time, therefore should be done in advance. The following plot shows an early test by way of example for LRS electrons; trying spin periods from 29.2s to 30.8s in steps of 0.005s, and start spin phase angles from 0° to 359.9° in steps of 0.1°, providing over a million sample calculations for VAR_m to be plotted. The figure clearly shows distinct quantization of values, with larger VAR_m values as you get further away from the perfect 30.000s spin period. The figure also shows that even if the spin period is known exactly, if the start spin phase is unknown then VAR_m could be any one of 3 or 4 different values.



For reference, SC_SPIN_PERIOD for when JADE has been on in 2015 and 2016 has varied from ~60.17s (early 2015), ~30.12s to ~30.15s, ~30.19s to ~30.20s and ~30.68s after JOI to ~30.74s at PJ1 (last known value at time of writing this document).

However as mentioned before, this is still very hard to get right. Onboard spin-phase used to select which look direction each anode falls in may well be different (to 32-bit resolution) to the spin-phase calculated on the ground with SPICE kernels.

Essentially – solving this problem to accurately get VAR_m will require a major investigation project, and is not done for this iteration of uncertainty calculation.

Instead, for this version 01 iteration, we shall assume m is known perfectly based on a perfect 30s (or multiples of) spin-period, and as such $VAR_m = 0$. This simplifies the equations considerably.

Should a users wish to add back in a VAR_m term they may take the DATA_SIGMA value from the level 3 files (generated by the equations in the next section without the VAR_m term by setting VAR_m=0), square it to get a variance, add the users desired VAR_m term to that variance, and square root it to return to the standard deviation.

[Note: Returning the onboard average is better than returning the onboard sum. Had the onboard code returned the sum only, we'd still have the uncertainty in number of views, so a VAR_m term, but now we'd also have an uncertainty in the average counts per view too – which we do not get from the onboard average – it knew what m was to make the average.]

Equations used for uncertainties in Version 01

Counts per Accumulation product.

Accumulation products do not have an unknown number of views – so the equations are unchanged from those originally postulated. However we add an extra condition, the cases when DATA is the MISSING_CONSTANT (FILL for short) value:

The combined standard uncertainty for a 'counts per accumulation' product is:

$$\frac{\text{Counts}}{\text{Accumulation}} = \begin{cases} DATA \pm \sqrt{\text{DATA}+VAR_{LUT}} & \text{if } DATA > 0 \\ 0 \pm 1 & \text{if } DATA = 0 \\ FILL \pm FILL & \text{if } DATA = FILL \end{cases}$$

Counts per View product. (Rate product)

For the three Rate products (low rate science electrons and low rate (and calibration) ion species) we assume we known m perfectly, hence $VAR_m = 0$, and the earlier equations simplify, except we have two new cases. If the DATA is the MISSING_CONSTANT value (FILL for short) then both the data and uncertainty are set to the MISSING_CONSTANT value. The second is if the level 2 object ACCUM_TRUNCATION equals 1 (rather than zero) then we know that the record was ended before the expected 30 (or multiples of) seconds. In that case we do not know which look directions got how many views, as such we report the value as DATA, but set the uncertainty to the MISSING_CONSTANT value.

The combined standard uncertainty for a 'counts per accumulation' product is:

$$\frac{\text{Counts}}{\text{View}} = \begin{cases} DATA \pm \sqrt{\dfrac{DATA}{m} + \dfrac{VAR_{LUT}}{m^2}} & \text{if } DATA > 0 \\ 0 \pm \sqrt{\dfrac{1}{m^2}} & \text{if } DATA = 0 \\ FILL \pm FILL & \text{if } DATA = FILL \\ DATA \pm FILL & \text{if } ACCUM\_TRUNCATION = 1 \end{cases}$$

However, there is a further complication in that the VAR_{LUT} values reported in the CALIB/LUT tables are for the cases after the onboard DATA was multiplied by 512. This must be accounted for and is shown in the following outline.

Background value uncertainties.

The uncertainties for any background value are calculated as above for accumulation or rate background products. If using the background anodes within some of the Level 2 files they are compressed the same way as the regular data, with the exception of JAD_L20_LRS_ELC_ANY where the Level 2 data has its own BACKGROUND_COUNTS object that is total accumulated counts over the ACCUMULATION_TIME without any lossy LUT compression (nor 512 multiplier), despite the DATA object being a rate product. Therefore if using the LRS electron background anode (object BACKGROUND_COUNTS) for a background value, then for JAD_L20_LRS_ELC_ANY only:

$$\frac{\text{BACKGROUND\_COUNTS}}{\text{Accumulation}} = \begin{cases} \text{BACKGROUND\_COUNTS} \pm \sqrt{\text{BACKGROUND\_COUNTS}} & \text{if BACKGROUND\_COUNTS} > 0 \\ 0 \pm 1 & \text{if BACKGROUND\_COUNTS} = 0 \\ \mathit{FILL} \pm \mathit{FILL} & \text{if BACKGROUND\_COUNTS} = \mathit{FILL} \end{cases}$$

e.g. same as the equation for any accumulation product, but with $VAR_{LUT} = 0$.

If no background is removed, then the level 3 objects BACKGROUND and BACKGROUND_SIGMA (both the same size as object DATA) will be all zeros.

To tell if a background has been removed from the data (or not) in the level 3 files, the SOURCE_BACKGROUND object in every level 3 record provides a number representing if a background has been removed, and if so, with what method. SOURCE_BACKGROUND = 0 means that no background was removed and that the BACKGROUND and BACKGROUND_SIMGA level 3 objects are all zeros, which is usually the case for the first iteration of data files.

Process to calculate uncertainties in code – an outline.

Implementing the above equations is not trivial – but here's the outline. The IDL code that follows is the true source – the following is trying to put the basics in to words.

- Start with Level 2 data, record by record
  For ions this is per ping record and per pong record, before merging them.
- If packet is from a rate species, `mult = 512`, else `mult = 1`
  Rate species are packets from JAD_L20_LRS_ELC_ANY, JAD_L20_LRS_ION_ANY or JAD_L20_CAL_ION_ANY.
  - For electrons, there are 10 views per look direction for a 30 s spin-period with 30s ACCUMULATION_TIME. This is true for all 48 look directions, all have 10 views. Therefore number of views, `m`, is:
    `m = ACCUMULATION_TIME * 10 / 30`
  - For ion species (both LRS and CAL) there are 78 look directions, for a given 30 s spin-period with 30s ACCUMULATION_TIME, the number of views may be 5, 3, 2 or 1. Therefor if
    ```
    LRS_species_views =  [5,5,5,3,2,3,2,3,2,1,1,1,1,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,1,1,1,1,1,1,1,2,3,2,3,2,3,5,5,5]
    ```
    then for look direction i (of 0 to 77):
    `m[i] = ACCUMULATION_TIME*LRS_species_views[i]/30`
  - Note, both the above equations work for a spin period of 1 or 2 revolutions per minute (RPM), e.g. for electrons with 2 RPM the equation would be `m = ACCUMULATION_TIME * 20/ 60`,
    however that is equivalent to the original equation.
    [Main mission is at ~2 RPM, but 2015 cruise data also had ~1 RPM.]
- Identify any element of DATA that is a MISSING_CONSTANT value. For Level 2 data this is either 4294967295 or 65535, depending on which product.
  This will be required later to reset any values back to MISSING_CONSTANT.

- If any DATA element is a MISSING_CONSTANT, set the corresponding uncertainty to MISSING_CONSTANT and do not alter that value with anything from the following steps. The remaining bullets only apply to non-MISSING_CONSTANT elements of DATA.
- Copy the DATA value, and remove the record's MIN_SUBTRACTED_VALUE from it, then multiply by `mult`.
  ```
  D = (DATA - MIN_SUBTRACTED_VALUE) * mult
  ```
- Identify which of the 4 compression LUTs to use: 16_1, 16_2, 32_1 or 32_2.
- Using the LUT, replace the level 2 count representative value (integers) with the more appropriate Level 3 count representative value (floats), and rename it D_L3. E.g. if LUT is 16_1, then find the index of object L2_16_1_REP that matches D with the corresponding index in L3_16_1_REP to get D_L3:
  ```
  D_L3    = LUT3_nn_n_REP[ WHERE( LUT2_nn_n_REP = D ) ]
  ```
  (NOTE : should not be doing this (or below) if D = MISSING_CONSTANT.)
- With that same index, find the VAR_{LUT} term. E.g. is LUT is 16_1, then the index of object L3_16_1_VAR value:
  ```
  VAR_LUT = LUT3_nn_n_VAR[ WHERE( LUT2_nn_n_REP = D ) ]
  ```
- Now reverse the earlier onboard adjustment equations:
  ```
  D_L3 = (D_L3 / mult) + MIN_SUBTRACTED_VALUE
  ```
- The VAR_{LUT} term may also need adjusting depending on if it's a rate product:
  ```
  VAR_LUT = VAR_LUT / mult / mult
  ```
- Now one of the following 4 cases, starting from the top
  - If `mult = 1` (i.e. an accumulation product)
    ```
    VAR_CNTS = D_L3
    VAR      = VAR_CNTS + VAR_LUT
    ```
    Finally, if any element of VAR is 0, replace it with a value of 1:
    ```
        j = WHERE( VAR = 0 )
        VAR[j] = 1
    ```
  - Else if `ACCUM_TRUNCATION = 1`
    Set all uncertainty elements to fill values and move to next record.
    ```
        VAR[ {all} ] = MISSING_CONSTANT
    ```
  - Else if `mult != 1` (i.e. a rate product) then the following equations apply.
    [Note that for LRS electrons, `m` is a scalar so use `m` rather than `m[i]` .]
    ```
    VAR_CNTS[i] = DATA_L3[i] / m[i]
    VAR_LUT[i]  = VAR_LUT[i] / m[i] / m[i]
    VAR[i]      = VAR_CNTS[i] + VAR_LUT[i]
    ```
    Finally, if any element of VAR is 0, replace it with a value of $1/m^2$:
    ```
        j = WHERE( VAR = 0 )
        VAR[j] = 1 / m[j] / m[j]
    ```
- As a final check – if any element of DATA = MISSING_CONSTANT, the corresponding element of VAR should also be a MISSING_CONSTANT, and if not then set it to MISSING_CONSTANT.
- The uncertainty for DATA is now the square root of VAR,
  unless VAR was a MISSING_CONSTANT, in which case the uncertainty becomes a MISSING_CONSTANT.

Process to calculate uncertainties in code – Actual IDL code used

The following two functions are the actual IDL code used (at time of writing this document when LUT version 3.09 was the latest) to calculate the uncertainties for Level 3 JADE data, version 01 of uncertainty method.
The first function puts the LUT from CALIB/LUT_3_00_COMPRESSION.CSV in to an IDL script; rather than 256 entries per array, which would take up many pages to list, we show the first two elements and last element of each.
The second function does all the calculation, taking an input of a Level 2 file (as an IDL structure). As opposed to what was said in the outline in the previous section, this code takes ion data with ping and pongs already merged, so there are two MIN_SUBTRACTED_VALUES to worry about, one for the first half of energy steps, and one for the second half.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
FUNCTION _jade_load_compression_LUT, TABLES_VERSION

  ON_ERROR,2
  COMPILE_OPT HIDDEN

  ;There are 4 compression tables for FSW4:
  ;16_1 : 16 to 8 bit compression for electrons
  ;16_2 : 16 to 8 bit compression for ions
  ;32_1 : 32 to 8 bit compression for electrons
  ;32_2 : 32 to 8 bit compression for ions
  ;High Rate science data use the 16 to 8 bit tables.
  ;Low Rate science (and CAL) data use the 32 to 8 bit
  ;tables (with some rate products first multiplied
  ;by 512)."

  ; LUTS 3.00 to 3.09 are identical, which are all the LUTs at time of coding.
  ; Values from CALIB/LUT_3_00_COMPRESSION.CSV
  ; Each array has 256 elements
  CASE 1 OF ; main code already checked LUT > 2.995, i.e. 3.00 with rounding
    (TABLES_VERSION LT 3.095) : LUT = { $ ; LUT 3.00 Compression table
      STEP       : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},             255.000d],$
      L2_16_1_MIN : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},           63243.000d],$
      L2_16_1_MAX : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},           65535.000d],$
      L2_16_1_REP : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},           64389.000d],$
      L3_16_1_REP : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},           64389.000d],$
      L3_16_1_VAR : [    0.000d,     0.000d, {253 ELEMENTS CUT FOR SPACE},          438345.167d],$
      L2_16_2_MIN : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},           63243.000d],$
      L2_16_2_MAX : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},           65535.000d],$
      L2_16_2_REP : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},           64389.000d],$
      L3_16_2_REP : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},           64389.000d],$
      L3_16_2_VAR : [    0.000d,     0.000d, {253 ELEMENTS CUT FOR SPACE},          438345.167d],$
      L2_32_1_MIN : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},      3773127747.000d],$
      L2_32_1_MAX : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},      4294967295.000d],$
      L2_32_1_REP : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},      4034047521.000d],$
      L3_32_1_REP : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},      4034047520.500d],$
      L3_32_1_VAR : [    0.000d,     0.000d, {253 ELEMENTS CUT FOR SPACE}, 22693042951859276.000d],$
      L2_32_2_MIN : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},      3773127747.000d],$
      L2_32_2_MAX : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},      4294967295.000d],$
      L2_32_2_REP : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},      4034047521.000d],$
      L3_32_2_REP : [    0.000d,     1.000d, {253 ELEMENTS CUT FOR SPACE},      4034047520.500d],$
      L3_32_2_VAR : [    0.000d,     0.000d, {253 ELEMENTS CUT FOR SPACE}, 22693042951859276.000d]}
    ELSE : MESSAGE,'ERROR: unrecognized LUT for compression table'
  ENDCASE

  RETURN, LUT
END
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
FUNCTION jade_counts_for_l3_v01, L2_innnnnn
;FUNCTION jade_counts_for_l3_v01, L2_innnnnn , SC_SPIN_PERIOD; spin period no longer used in this code

  ; This code calculates uncertainties based on Poisson Statistics and
  ; the compression LUT table.
  ; It assumes perfect 30s spins (or 60s) with exactly known number of
  ; views for rate species (i.e. VAR_m = 0 assumed and not even coded for)


  ON_ERROR,2

  L2 = L2_innnnnn ; so as not to change original structure.

  t = TAG_NAMES(L2)
  IF MAX(L2.DATA_UNITS) GT 1 THEN $ ; $ = continue on next line
    MESSAGE,'ERROR: Input structure does not look like Level 2, DATA_UNITS look wrong'
  IF TOTAL(STRCMP(t,'UTC')) NE 1 THEN $
    MESSAGE,'ERROR: Input structure does not look like Level 2, no UTC field'
  IF (L2.PACKET_SPECIES[0] GE 0) THEN IF (MAX(STRCMP(t,'T_PING_PONG')) EQ 0) THEN $
    MESSAGE,"ERROR: This structure must already be merged for pings and pongs" ; only if ion
  ; Must run CAL modes separate to LRS/HRS
  MAX_MODE = MAX(L2.PACKET_MODE, MIN = MIN_MODE)
  IF (MIN_MODE EQ 0) AND (MAX_MODE NE 0) THEN $
    MESSAGE,'ERROR: For conversion of counts, must do CAL modes separately to HRS and/or LRS.'


  ;TABLES_VERSION has a MISSING_CONSTANT  = -99.99
  ind = WHERE(L2.TABLES_VERSION LT -99.00,COMPLEMENT=notind,/NULL)
  IF N_ELEMENTS(ind) GT 0 THEN BEGIN
    IF N_ELEMENTS(ind) EQ N_ELEMENTS(L2.TABLES_VERSION) THEN BEGIN
      PRINT,'WARNING: No records in file have a valid TABLES_VERSION (LUT) number, '+$
        'returning an empty structure of -1'
      RETURN,-1
    ENDIF
    PRINT,'WARNING: Removing records with a fill value for TABLES_VERSION'
    L2 = jade_st_shrink(TEMPORARY(L2),INDEX = notind)
    ; jade_st_shrink shrinks structure to only keep records specified by INDEX
    ; e.g. here is keeping records with L2.TABLES_VERSION GE -99.00,
    ; and removeing records with L2.TABLES_VERSION LT -99.00
  ENDIF
  ; L2.TABLES_VERSION is definitely not fill by here
  IF MIN(L2.TABLES_VERSION) LT 2.995 THEN $
    MESSAGE,'ERROR: Input structure needs a TABLES_VERSION 3.00 or greater'

  CASE L2.PACKETID[0] OF
    96 : FMT = 'JAD_L20_LRS_ION_ANY'
    97 : FMT = 'JAD_L20_LRS_ION_ANY'
    98 : FMT = 'JAD_L20_LRS_ION_ANY'
    99 : FMT = 'JAD_L20_LRS_ION_ANY'
    100: FMT = 'JAD_L20_LRS_ION_ANY'
    101: FMT = 'JAD_L20_LRS_ION_ANY'
    102: FMT = 'JAD_L20_LRS_ION_ANY'
    103: FMT = 'JAD_L20_LRS_ION_ANY'

    112: FMT = 'JAD_L20_CAL_ION_ANY'
    113: FMT = 'JAD_L20_CAL_ION_ANY'
    114: FMT = 'JAD_L20_CAL_ION_ANY'
    115: FMT = 'JAD_L20_CAL_ION_ANY'
    116: FMT = 'JAD_L20_CAL_ION_ANY'
    117: FMT = 'JAD_L20_CAL_ION_ANY'
    118: FMT = 'JAD_L20_CAL_ION_ANY'
    119: FMT = 'JAD_L20_CAL_ION_ANY'

    128: FMT = 'JAD_L20_HRS_ION_ANY'
    129: FMT = 'JAD_L20_HRS_ION_ANY'
    130: FMT = 'JAD_L20_HRS_ION_ANY'
    131: FMT = 'JAD_L20_HRS_ION_ANY'
    132: FMT = 'JAD_L20_HRS_ION_ANY'
    133: FMT = 'JAD_L20_HRS_ION_ANY'
    134: FMT = 'JAD_L20_HRS_ION_ANY'
    135: FMT = 'JAD_L20_HRS_ION_ANY'

    137: FMT = 'JAD_L20_HLS_ION_TOF'
    105: FMT = 'JAD_L20_HLS_ION_TOF'
    121: FMT = 'JAD_L20_CAL_ION_TOF'

    140: FMT = 'JAD_L20_HLS_ION_LOG'
    108: FMT = 'JAD_L20_HLS_ION_LOG'
    124: FMT = 'JAD_L20_CAL_ION_LOG'

    142: FMT = 'JAD_L20_HRS_ELC_ALL'
    104: FMT = 'JAD_L20_LRS_ELC_ANY'
    106: FMT = 'JAD_L20_LRS_ELC_ANY'
    107: FMT = 'JAD_L20_LRS_ELC_ANY'
    126: FMT = 'JAD_L20_CAL_ELC_ALL'

    ELSE  :   MESSAGE,'ERROR: PacketID '+STRTRIM(STRING(L2.PACKETID[0]),2)+ $
      ' packets should not have counts converted in preparation for level 3.'
  ENDCASE
  ; Should not get here if you try a wrong packet

  CASE FMT OF ; 3 products are multiplied x 512 onboard (never TOF nor LOG)
    'JAD_L20_LRS_ELC_ANY' : mult = 512d ; a rate species
    'JAD_L20_LRS_ION_ANY' : mult = 512d ; a rate species
    'JAD_L20_CAL_ION_ANY' : mult = 512d ; a rate species
    ELSE : mult = 1d
  ENDCASE

  ; set up number of views constants for rate species
  IF ROUND(mult) EQ 512 THEN BEGIN
    ; For LRS electrons
    ; ASSUMING A PERFECT MULTIPLE OF 30 second spins,
    ; where each look direction would get 10 views on a perfect 30s spin period
    LRS_electron_views = 10d ; for a 30s spin period
    ; And for LRS/CAL ion species
    ; ASSUMING A PERFECT MULTIPLE OF 30 second spins, in one 30 second spin,
    ; there are 15 2-second sweeps
    LRS_species_views = DOUBLE([     $
      5,         5,         5,         $
      3,     2,  3,     2,  3,     2,  $
      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,$
      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,$
      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,$
      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,$
      2,  3,     2,  3,     2,  3,     $
      5,         5,         5          ])
    ;LRS_species_views2 = DBLARR(1,64,78,/NOZERO)
    ;FOR vz = 0,77 DO LRS_species_views2[0,*,vz] = LRS_species_views[vz]
    ;LRS_species_views = LRS_species_views2
    ;; Next line does the above 3 commented out lines faster
    LRS_species_views = REBIN( REFORM(LRS_species_views,1,1,78) ,1,64,78)
    ismult = 1
  ENDIF ELSE BEGIN
    ismult = 0
  ENDELSE

  CASE FMT OF
    'JAD_L20_HRS_ELC_ALL' : key = ['_16_1_']
    'JAD_L20_LRS_ELC_ANY' : key = ['_32_1_']
    'JAD_L20_CAL_ELC_ALL' : key = ['_32_1_']
    'JAD_L20_HRS_ION_ANY' : key = ['_16_2_']
    'JAD_L20_LRS_ION_ANY' : key = ['_32_2_']
    'JAD_L20_CAL_ION_ANY' : key = ['_32_2_']
    'JAD_L20_CAL_ION_TOF' : key = ['_32_2_']
    'JAD_L20_CAL_ION_LOG' : key = ['_32_2_']
    'JAD_L20_HLS_ION_TOF' : key = ['_32_2_','_16_2_']
    'JAD_L20_HLS_ION_LOG' : key = ['_32_2_','_16_2_']
    ELSE : MESSAGE,'ERROR: unrecognized '+FMT
  ENDCASE

  CASE FMT OF ; 3 products are multiplied x 512 onboard (never TOF nor LOG)
    'JAD_L20_HRS_ELC_ALL' : MISSING_CONSTANT = 65535LL ; make LONG64 with LL
    'JAD_L20_HRS_ION_ANY' : MISSING_CONSTANT = 65535LL ; make LONG64 with LL
    ELSE : MISSING_CONSTANT = 4294967295LL ; make LONG64 with LL
  ENDCASE

  L3_MISSING_CONSTANT = -1d ; Set Missing Constant to use for L3 files.

  ; If any MISSING_CONSTANT values in DATA, set to -1 instead of what they where
  L2.DATA[WHERE((ROUND(L2.DATA,/L64) EQ MISSING_CONSTANT) OR (L2.DATA EQ -1), /NULL)] =
L3_MISSING_CONSTANT ; 1 line for L2.DATA to L3_MISSING_CONSTANT

  L2 = CREATE_STRUCT(L2,'DATA_SIGMA', L2.DATA)
  L2.DATA_SIGMA[*] = L3_MISSING_CONSTANT ; preallocate with -1 as fill value

  DATA_VAR    = L2.DATA ; start preallocation by matching size
  DATA_VAR[*] = 0d ; preallocate with 0 at the same size as DATA
  empty_zeros    = DATA_VAR[0,*,*,*] ; pre-allocate for size, set to zero above

  prev_LUT = -99
  prev_mode = -3
  nm1 = N_ELEMENTS(L2.T) -1L
  FOR rec = 0L,nm1 DO BEGIN
    ; Don't get new LUT if no change
    IF (prev_LUT NE L2.TABLES_VERSION[rec]) OR (prev_mode NE L2.PACKET_MODE[rec]) THEN BEGIN
      prev_LUT  = L2.TABLES_VERSION[rec] ; update prev_lut
      prev_mode = L2.PACKET_MODE[   rec] ; update prev_mode

      CASE FMT OF
        ; For these two FMTS:
        ; packet_mode of 2 means HRS means 16_2 table (index 1), while ...
        ; ...packet_mode of 1 means LRS means 32_2 table (index 0).
        'JAD_L20_HLS_ION_TOF' : key_dim = ROUND(L2.PACKET_MODE[rec]) - 1
        'JAD_L20_HLS_ION_LOG' : key_dim = ROUND(L2.PACKET_MODE[rec]) - 1
        ELSE : key_dim = 0
      ENDCASE
      LUT = _jade_load_compression_LUT(prev_LUT) ; and now get new one.
      LUT_TAGS = TAG_NAMES(LUT)
      L2_rep = WHERE(STRCMP(LUT_TAGS,'L2'+key[key_dim]+'REP',/FOLD_CASE) EQ 1, NULL=1)
      ; NULL = 1 faster than /NULL
      IF N_ELEMENTS(L2_rep) EQ 0 THEN MESSAGE,'ERROR: Failed to find LUT tag l2 rep',key_dim
      L3_rep = WHERE(STRCMP(LUT_TAGS,'L3'+key[key_dim]+'REP',/FOLD_CASE) EQ 1, NULL=1)
      IF N_ELEMENTS(L3_rep) EQ 0 THEN MESSAGE,'ERROR: Failed to find LUT tag l3 rep',key_dim
      L3_var = WHERE(STRCMP(LUT_TAGS,'L3'+key[key_dim]+'VAR',/FOLD_CASE) EQ 1, NULL=1)
      IF N_ELEMENTS(L3_var) EQ 0 THEN MESSAGE,'ERROR: Failed to find LUT tag l3 var',key_dim

      L2_rep_values = ROUND(LUT.(L2_rep),/L64) ; make integer of type LONG64 - used in for loop below
    ENDIF
    ; Can use too many dimensions - if it's an unused dimension it's simply a dimension of size 1
    ; e.g. an array of size n x 64 x 13 is the same as an array of size n x 64 x 13 x 1
    ; so I will use 4 dimensions for all, as TOF goes up to 4 dimensions, although most are only to 3
    DATA_L2 = L2.DATA[rec,*,*,*]
    ; Find fills now so we can replace them later
    FILLS = WHERE(DATA_L2 EQ L3_MISSING_CONSTANT, nFILLS, NULL=1) ; NULL = 1 faster than /NULL

    ; Deal with min-subtracted value and the products with the 512 multiple
    IF N_ELEMENTS(L2.MIN_SUBTRACTED_VALUE[rec,*]) EQ 1 THEN BEGIN ; ELC
      DATA_L2[0,    *,*,*] -= L2.MIN_SUBTRACTED_VALUE[rec]
    ENDIF ELSE BEGIN ; ION, has two values - one for the ping, one for the pong.
      DATA_L2[0, 0:31,*,*] -= L2.MIN_SUBTRACTED_VALUE[rec,0]
      DATA_L2[0,32:63,*,*] -= L2.MIN_SUBTRACTED_VALUE[rec,1]
    ENDELSE
    ; Account for products thare are multiplied by 512
    IF ismult EQ 1 THEN BEGIN ; IF statment fore speed, don't bother if mult = 1
      DATA_L2 = DATA_L2 * mult
    ENDIF
    ; Replace fill values to a fill value of -1
    IF nFILLS GT 0 THEN DATA_L2[FILLS] = L3_MISSING_CONSTANT

    DATA_L2 = ROUND(TEMPORARY(DATA_L2),/L64) ; make DATA_L2 an integer of type LONG64
    DATA_L3 = DATA_L2 ; preallocate array of integers of right size.
    uni = jade_unique(DATA_L2) ; jade_unique extracts array of unique values in DATA_L2.
    ; check there's more than just fill, if it it just fill, set record to -1s and continue
    IF (N_ELEMENTS(uni) EQ 1) AND (uni[0] EQ L3_MISSING_CONSTANT) THEN BEGIN
      CONTINUE ; nothing to do as nothing but fill, go to next FOR loop iteration
    ENDIF
    uni = uni[WHERE(uni NE L3_MISSING_CONSTANT,n_uni,NULL=1)] ; remove any fill values from list
    ; uni can not be empty/NULL due to IF statement above.

    VAR_LUT = empty_zeros ; preallocate with 0 at the same size as DATA_L2
    FOR z = 0L, (n_uni - 1L) DO BEGIN
      index = WHERE(L2_rep_values EQ uni[z], NULL=1) ; NULL = 1 faster than /NULL
      IF N_ELEMENTS(index) EQ 0 THEN $
        MESSAGE,'ERROR: Could not find value in LUT L2 table to convert to an L3 equivalent'
      ind = WHERE(DATA_L2 EQ uni[z],NULL=1) ; can't be NULL as value in uni, but kept for safety
      DATA_L3[ind] = LUT.(L3_rep)[index]
      VAR_LUT[ind] = LUT.(L3_var)[index]
    ENDFOR

    ; Reverse deal with min-subtracted value and the products with the 512 multiple
    IF ismult EQ 1 THEN BEGIN ; IF statment fore speed, don't bother if mult = 1
      DATA_L3 = DATA_L3 / mult        ; Account for products thare are multiplied by 512
      VAR_LUT = VAR_LUT / mult / mult ; since this is variance, need to divide twice,
      ; e.g. (stddev/mult) squared is (sqrt(var)/mult)^2 = var/mult^2
    ENDIF
    IF N_ELEMENTS(L2.MIN_SUBTRACTED_VALUE[rec,*]) EQ 1 THEN BEGIN ; ELC
      DATA_L3 += L2.MIN_SUBTRACTED_VALUE[rec]
    ENDIF ELSE BEGIN ; ION
      DATA_L3[0, 0:31,*,*] += L2.MIN_SUBTRACTED_VALUE[rec,0]
      DATA_L3[0,32:63,*,*] += L2.MIN_SUBTRACTED_VALUE[rec,1]
    ENDELSE
    IF (nFILLS GT 0) THEN DATA_L3[FILLS] = L3_MISSING_CONSTANT ; replace fills

    ;;; typical_spin_period cancels out below! Doesn't matter if 60 or 30! Do not need this section!
;    ; If a rate product, we need to know the spin period for later.
;    ; Don't need if an accumulation product.
;    IF L2.DATA_UNITS[rec] EQ 1 THEN BEGIN
;      CASE 1 OF
;        ; Engineering consider 2 RPM to be 29.2 to 30.8 seconds.
;        (SC_SPIN_PERIOD[rec] GE 29.2) AND (SC_SPIN_PERIOD[rec] LE 30.8)          : $
;          typical_spin_period = 30d ; We expect this at Jupiter.
;        (ROUND(SC_SPIN_PERIOD[rec]) EQ 60) AND (L2.TIMESTAMP_WHOLE LT 481935731) : $
;          typical_spin_period = 60d ; Cruise in early 2015 only, SCLK 481935731 = 2015-100T11:00:00
;        ELSE : MESSAGE,'ERROR: unrecognized standard spin for rate product'
;      ENDCASE
;      nspins = DOUBLE(ROUND( L2.ACCUMULATION_TIME[rec] / typical_spin_period))
;      ; rounding for safety - but not required if both numbers are multiples of 30
;    ENDIF

    ; Now calculate variance... using Poisson it's the size of the counts,
    ; plus variance due to unknowing the counts due to the compression table
    CASE 1 OF
      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      L2.DATA_UNITS[rec] EQ 0: BEGIN
        ; ACCUMULATION product - nice and easy!
        ; Does not matter if ACCUM_TRUCATION EQ 1, we know the accumulation time
        VAR_CNTS = DATA_L3 ; This is the Poisson stats variance term
        VAR      = VAR_CNTS + VAR_LUT ; Combined Standard uncertainty
        VAR[ WHERE(VAR EQ 0, NULL=1) ] = 1d ; Deal with any case of variance = 0
      END
      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      ; If a rate product but was accumulation truncated - return fill values for
      ; uncertainties, after all, we really don't know how many views (m)...
      L2.ACCUM_TRUNCATION[rec] EQ 1:  VAR[*] = L3_MISSING_CONSTANT
      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      ; Case statements that follows must have L2.DATA_UNITS[rec] = 1 and L2.ACCUM_TRUNCATION[rec] = 0
      L2.PACKET_SPECIES[rec] LT 0: BEGIN
        ; Rate product, and electrons,  must be LRS electrons
        ; ASSUMING A perfect MULTIPLE OF 30 second spins
        ; m = number of views per spin sector
        ;m = LRS_electron_views * nspins * typical_spin_period / 30d
        ; This is independent of spin rate if assumed spin rate is a multiple of 30
        m = LRS_electron_views * L2.ACCUMULATION_TIME[rec] / 30d
        ; note, m is a scalar for LRS Electrons.
        ; We are ignoring any variance in m itself.
        VAR_CNTS = DATA_L3 / m ; This is the Poisson stats variance term
        VAR_LUT  = VAR_LUT / m / m ; This is the term based on the uncertainty in the LUT
        VAR      = VAR_CNTS + VAR_LUT   ; Combined Standard uncertainty, no VAR_m term
        VAR[ WHERE(DATA_L3 EQ 0, NULL=1) ] = 1d / m / m ; Deal with zero count cases
      END
      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      ELSE: BEGIN
        ; Rate product, and ion species data for LRS or CAL
        ; ASSUMING A perfect MULTIPLE OF 30 second spins
        ; m = number of views per spin sector
        ;m = LRS_species_views * nspins * typical_spin_period / 30d
        ; This is independent of spin rate if assumed spin rate is a multiple of 30
        m = LRS_species_views *  L2.ACCUMULATION_TIME[rec] / 30d
        ; note, m is an array of size 1 x 64 x 78 for ion species LRS/CAL.
        ; We are ignoring any variance in m itself.
        VAR_CNTS = DATA_L3 / m ; This is the Poisson stats variance term
        VAR_LUT  = VAR_LUT / m / m ; This is the term based on the uncertainty in the LUT
        VAR      = VAR_CNTS + VAR_LUT ; Combined Standard uncertainty
        ; now deal with zero counts
        iz = WHERE(DATA_L3 EQ 0, NULL=1) ; NULL = 1 faster than /NULL
        IF N_ELEMENTS(iz) GT 0 THEN VAR[iz] = 1d / m[iz] / m[iz]
      END
      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ENDCASE

    ; Replace fills in original L2.DATA array with fills for data and uncertainty
    DATA_L3[FILLS] = L3_MISSING_CONSTANT ; not original MISSING_CONSTANT
    VAR[    FILLS] = L3_MISSING_CONSTANT ; not original MISSING_CONSTANT
    ; Put back in structure
    L2.DATA[ rec,*,*,*] = DATA_L3
    DATA_VAR[rec,*,*,*] = VAR
  ENDFOR

  ; Finally sqrt(Variance) = std dev, but only those values that are not -1
  ; note L2.DATA_SIGMA was set to -1 initially
  ind = WHERE(DATA_VAR NE L3_MISSING_CONSTANT, NULL=1) ; NULL = 1 faster than /NULL
  IF N_ELEMENTS(ind) GT 0 THEN L2.DATA_SIGMA[ind] = SQRT( DOUBLE( DATA_VAR[ind] ) )

  RETURN,L2
END
```