Implementation of Unevenly Spaced Resultants in Pandeia

S. Gomez

October 12, 2023

Abstract

Observations from the Nancy Grace Roman Space Telescope (Roman) will be able to have a variable number of reads for each resultant and a variable number of skips between resultants, within a single exposure. This document outlines the process to implement this feature in the Pandeia exposure time calculator, a feature that is not currently supported. We include an updated parameter nomenclature that will need to be implemented to be consistent with Roman observations, as well as a Python-based example to implement unevenly spaced resultants in Pandeia to calculate the variance of the measured count rate of an observation.

1 Introduction

The Wide Field Instrument (WFI) on board the Nancy Grace Roman Space Telescope (Roman) has 18 near-infrared-sensitive Teledyne H4RG-10 sensor chip assemblies (SCAs) able to non-destructively read out the accumulated charge every few seconds. For every readout of the Roman SCAs during an exposure (similar to an “integration” in JWST terminology), the readout may be sent to a buffer (“read”) or discarded (“skip”). The instructions for every readout of the SCAs during an exposure (also called the “readout pattern”) are stored in a multi-accum (MA) table, including which readouts are skipped and how the reads in the buffer are averaged into “resultants”. Resultants are the combination of individual reads that are down-linked to the ground system, and are analogous to the JWST “groups”.

The Pandeia exposure time calculator (ETC) was originally developed for JWST, and is now being used to also predict the signal-to-noise ratio (S/N) of Roman observations. The fundamental problem we address here is that, unlike JWST, Roman observations will be able to have unevenly spaced resultants. While each JWST group in an integration has the same number of reads and skips, every Roman resultant can have a different number
of reads within an exposure, and the number of skips between resultants need not be a constant during the exposure. This readout pattern allows for a higher dynamic range of observable targets that can recover dim sources while reducing the saturation of bright ones. Additionally, users will have the option to select at which resultant to truncate an MA table to stop observing if desired. An example of a readout pattern with unevenly spaced resultants is shown in Figure 1.

Figure 1: Representation of a Roman exposure with six resultants. Note that the number of reads per resultant and the number of skips between resultants is not a constant during the exposure. Roman refers to these as unevenly spaced resultants. (Image credit: Tyler Desjardins).

The functions used to calculate the S/N in the current version of Pandeia are based on the equations from Robberto (2009), which are only valid for the case of evenly spaced resultants. Here, we outline the process to update the equations in Pandeia to instead use the methods of Casertano (2022), which not only allow for unevenly spaced resultants, but also introduce flux-dependent weights to improve the estimate of the variance. This method should be applicable to any telescope supported by Pandeia and provide more accurate S/N estimates across missions.

2 Front-end Schema

Roman observations have a distinct nomenclature from what JWST uses to describe different components of an observation. In order for Pandeia to be consistent with the nomenclature used in the Roman APT, these keywords will need to be updated or implemented in Pandeia. When loading in an observation using the build_default_calc function
from the `roman/wfi/config.json` reference file, the output dictionary should contain the keywords listed in Table 1. The number of exposures `nexp` can remain the same, the instrument should always be `wfi`, and if the user attempts to specify a number of groups or integrations this should return an error or warning, since these are JWST-specific terms. Instead, the number of groups and skips in an observation will be defined by specifying the name of the MA table to be read in. The user can then select at what point to truncate the MA table with the `nresultants` parameter. Finally, the `filter` parameter should be reworded to `optical_element` to account for filters, the grism, and prism.

The readout patterns that will be input into Pandeia come from MA tables that are formatted as a list of lists, where each existing value represents a frame that will be read, and any missing value is a frame that will be skipped. For example, the readout pattern shown in Figure 1 would be represented as the following list: `[[1], [3,4], [6,7,8], [11,12,13,14,15,16], [19,20,21,22,23,24], [27,28,29,30,31,32]]`. This list can be stored in the equivalent of the `readout_pattern_config` key in the `roman/wfi/config.json` reference file, a list that currently stores the individual names a readout patterns for JWST. These names can be replaced with the names of the Roman MA tables.

<table>
<thead>
<tr>
<th>Description</th>
<th>Current</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of exposures</td>
<td><code>nexp</code></td>
<td><code>nexp</code></td>
</tr>
<tr>
<td>Number of groups</td>
<td><code>ngroup</code></td>
<td><code>N/A</code></td>
</tr>
<tr>
<td>Number of integrations</td>
<td><code>nint</code></td>
<td><code>N/A</code></td>
</tr>
<tr>
<td>Number of resultants</td>
<td><code>N/A</code></td>
<td><code>nresultants</code></td>
</tr>
<tr>
<td>Name of MA table</td>
<td><code>readout_pattern</code></td>
<td><code>ma_table_name</code></td>
</tr>
<tr>
<td>WFI optical element</td>
<td><code>filter</code></td>
<td><code>optical_element</code></td>
</tr>
<tr>
<td>Instrument</td>
<td><code>instrument</code></td>
<td><code>instrument</code></td>
</tr>
</tbody>
</table>

Table 1: Keywords currently used in Pandeia and the equivalent names that will be required for Roman observations.

## 3 Method Implementation

The functions currently implemented in Pandeia use the equations from [Robberto (2009)](Robberto2009) to estimate the read-noise variance $V_r$, shot-noise variance $V_s$, and total variance $V$. These equations are stored in the `slope_variance` and `rn_variance` functions of the `Detector` module in the current version of Pandeia. An abridged version of pseudo-code that shows only the relevant portions that will need to be modified is shown in Listing 1. The relevant equations used to calculate $V_r$ and $V_s$ are below, where we use $\Delta t$ to denote the read time $tframe$. All examples listed in this document correspond to a single pixel, these calculations should be expanded for the entire array.
\[
V_r = \frac{12 \times rn^2}{m n(n^2 - 1) t_{\text{group}}^2}
\]

\[
V_s = \frac{6}{5} \frac{n^2 + 1}{b} \frac{m^2 - 1}{3 m(n^2 + 1) t_{\text{group}}^2} \left(1 - \frac{5}{3} \frac{m^2 - 1}{m(n^2 + 1) t_{\text{group}}} \Delta t \right)
\]

rn = Read noise
m = Number of frames
n = Number of unsaturated groups
tframe = Duration of one frame, (e.g. 3.04s)
rate['fp_pix_variance'] = Pixel rate in counts / s

# Total readout time
tgroup = self.exposure_spec.tgroup

# Variance due to read noise only
var_rn = 12. * rn ** 2. / (m * n * (n ** 2. - 1.) * tgroup ** 2.)

# Variable rewording in Pandeia
slope_rn_var = var_rn

# Pixel rate before IPC convolution
variance_per_pix = rate['fp_pix_variance']

# Variance due to the shot noise AND read noise
slope_var = (6. / 5.) * (n ** 2. + 1.) / (n * (n ** 2. - 1.)) * \n(variance_per_pix / self.exposure_spec.tgroup) * \n(1. - (5. / 3.) * (m ** 2. - 1.) / (m * (n ** 2. + 1.))) * \n(tframe / self.exposure_spec.tgroup)) + slope_rn_var

Listing 1: Current Pandeia implementation used to calculate the variance of an observation.

To support unevenly spaced resultants, the equations and methods in Casertano (2022) must be implemented. This modification requires several extra steps before the computation of the variance due to the read noise and shot noise components can be calculated. The readout pattern is specified in the form of a list of lists, described in §2. This list can be converted into two arrays: one with the total number of reads per resultant n_reads, and one with the total number of frames per resultant n_total, including reads and skips. An example of how this could be implemented in Pandeia is shown in Listing 2. Note that skips may only occur between resultants, and not follow the last read.

import numpy as np

# Readout pattern read from MA table
readout_pattern = [[1], [3, 4], [6, 7, 8], [11, 12, 13, 14, 15, 16],
                   [19, 20, 21, 22, 23, 24], [27, 28, 29, 30, 31, 32]]
# Get number of reads in each resultant
n_reads = np.array([len(i) for i in readout_pattern])

# Get total number of frames in each resultant
first = np.array([i[0] for i in readout_pattern])
n_total = np.append(np.diff(first), len(readout_pattern[-1]))

Listing 2: Example of input readout pattern format converted into number of reads and total frames in each resultant.

Once the number of reads and frames per resultant have been extracted from the MA table, we can use these in combination with the known count rate in electrons per second to estimate the variance. The optimal algorithm for the slope determination described in Casertano (2022) requires the inclusion of a flux-dependent weight matrix, \( W_{ij} \) in Casertano (2022). Accounting for these weights can have an effect in the order of \( \sim 10\% \) for the measurement of the variance, this is particularly noticeable for count rates \(< 1 \) electron / s. Equations 44 and 45 in Casertano (2022) are shown below and describe how to calculate these weights based on the maximum flux \( s_{\text{max}} \). The value of a flux-dependent exponent \( P \) can be determined from the value of \( s \) using the reference table in Table 2 of Casertano (2022), also provided in the \texttt{power_value} function shown in Listing 3.

\[
\begin{align*}
  s &= \frac{s_{\text{max}}}{\sqrt{rn^2 + s_{\text{max}}}} \\
  w_i &= \frac{(1 + P) \times N_i}{1 + P \times N_i} |t_i - t_{\text{mid}}|^P
\end{align*}
\]

Here \( N_i \) is defined as the number of reads in each resultant, \( t \) is the mean time of each resultant, and \( t_{\text{mid}} \) is the midpoint of the exposure. We show an example of how these calculations could be implemented in Pandeia in Listing 3. This example only applies to a single pixel, but the methodology can be expanded to the entire detector array with a matrix of equal size to the detector. The “optimal” example described in the paragraph above uses the values from Table 2 in Casertano (2022) to determine the optimal exponent \( P \), while the “simple” example assumes \( P = 0 \), and the “unweighted” example sets all weights equal to 1. As mentioned before, the choice of weighing scheme has an effect in the order of \( \sim 10\% \) on the measured variance. In this example \texttt{variance_per_pix}, \texttt{tframe}, and \texttt{rn} are the count rate, frame time, and read noise already used in Pandeia.

```python
import numpy as np

def power_value(s):
    ""
    Determine the optimal exponent \( P \) given a signal \( S \) based on
    ""
```
if s < 5:
    return 0
elif (5 <= s < 10):
    return 0.4
elif (10 <= s < 20):
    return 1.0
elif (20 <= s < 50):
    return 3.0
elif (50 <= s < 100):
    return 6.0
elif s >= 100:
    return 10

# Calculate beginning time of each read
t0 = np.append(0, np.cumsum(n_total)[::-1])

# Calculate mean time
tmean = tframe * t0 + tframe * (n_reads - 1) / 2

# Resultant flux
flux = variance_per_pix * tmean.reshape(-1, 1)

# Calculate power from maximum flux
s_max = np.max(flux)
s = s_max / np.sqrt(rn**2 + s_max)
P = power_value(s)

if weights == 'optimal':
    # Calculate optimal weights using Equation 45 in Casertano et al.
    tmid = (tmean[0] + tmean[-1]) / 2
    W_ij = ((1 + P) * n_reads) / (1 + (P * n_reads)) * \n          np.abs(tmean - tmid) ** P
elif weights == 'simple':
    # Calculate weights based on number of reads, or P = 0
    W_ij = n_reads
elif weights == 'unweighted':
    # Set all weights equal to 1
    W_ij = np.ones(len(n_reads))

Listing 3: Example of proposed weight calculation using three different weighing schemes.

The method presented in Casertano (2022) provides a way to ease the calculation of variance by introducing the auxiliary quantities $F_0$, $F_1$, $F_2$, $D$, and $K_i$ in Equations 35, 36, and 37 and $\tau$ in Equation 15 of that document. These equations are shown below, where we
use $\Delta t$ to denote the read time $t_{\text{frame}}$:

$$F_0 = \sum_{i=0}^{N-1} W_i$$

$$F_1 = \sum_{i=0}^{N-1} W_i t_i$$

$$F_2 = \sum_{i=0}^{N-1} W_i t_i^2$$

$$D = F_2 \times F_0 - F_1^2$$

$$K_i = \frac{(F_0 \times t_i - F_1) W_i}{D}$$

$$\tau_i = t_i - (n_i - 1)(n_i + 1) \frac{\Delta t}{6n_i}$$

Finally, we can use these auxiliary quantities in combination with the observation pattern in the MA table and the input flux values to calculate the read-noise variance, shot-noise variance, and total variance. An example of how this could be implemented in Pandeia is shown in Listing 4. The final `slope_var` parameter is equivalent to the one currently implemented in Pandeia, and this can then be fed into the remainder of the functions in Pandeia that require it.

```python
import numpy as np

# Calculate the auxiliary quantities
F0 = np.sum(W_ij, axis=0)
F1 = np.sum(W_ij*tmean, axis=0)
F2 = np.sum(W_ij*tmean**2, axis=0)
D = F2 * F0 - F1**2
K_i = (F0 * t_i - F1) * W_i / D

# Calculate variance-based resultant time tau
tau = tmean - (n_reads - 1) * (n_reads + 1) * tframe / 6 / n_reads

# Get number of resultants
nresultants = len(n_reads)

# Variance associated with the read noise only
V_r = rn**2 * np.sum(K_i**2 / n_reads, axis=0)

# Variance associated with the shot-noise
V_s1 = np.sum(K_i**2 / tau)
V_s2 = np.sum([np.sum([K_i[i] * K_i[j] * tmean[i]] for j in range(i + 1, n_reads)])
```

7
\[
V_s = V_{s1} + 2 \times V_{s2}
\]
\[
slope_{\text{var}} = V_r + V_s \times \text{variance\_per\_pix}
\]

Listing 4: Example of proposed calculation of variance.

References

Casertano, S. 2022, Determining the best-fitting slope and its uncertainty for up-the-ramp sampled images with unevenly distributed resultants, Technical Report Roman-STScI-000394