UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

Simulated Data and Exposure Time Calculator for the NOT Transient Explorer

Master's Thesis

Written by

Mads Nymann-Lynggaard (gbc493)

Cecilie Valet Henneberg (xhf876)

May 22, 2023

Supervised by

Johan Peter Uldall Fynbo

Joonas Kari Markku Viuho

University of Copenhagen

UNIVERSITY OF COPENHAGEN

Faculty:	Faculty of Science
Institute:	Niels Bohr Institute
Research Center:	The Cosmic Dawn Center (DAWN)
Authors:	Mads Nymann-Lynggaard (gbc493)
	Cecilie Valet Henneberg (xhf876)
Email:	gbc493@alumni.ku.dk
	xhf876@alumni.ku.dk
Supervisors:	Johan Peter Uldall Fynbo
	Joonas Kari Markku Viuho
Handed in:	22.05.2023
Defended:	20.06.2023

Acknowledgements

First and foremost, we would like to express our gratitude to our supervisor, Prof. Johan Peter Uldall Fynbo, for his invaluable guidance, support, and continuous encouragement throughout the entire process of our thesis. His expertise and patient explanations have significantly contributed to our understanding of numerous topics.

We would also like to thank our co-supervisor Joonas Kari Markku Viuho, who proved to be an infinite source of knowledge. His generous sharing of expertise, patient mentoring, and willingness to go above and beyond to help us when needed were essential to the successful completion of our thesis.

Furthermore, we are grateful to the entire NTE team for their willingness to assist and for providing crucial data for the project. Their collaboration has been invaluable.

Abstract

This thesis presents the development of an exposure time calculator and the generation of simulated data to test a data pipeline for the NOT Transient Explorer (NTE), a new instrument designed for the NOT (Nordic Optical Telescope). The exposure time calculator serves as a tool for telescope users to determine optimal exposure times for different observing conditions. The exposure time calculator derives a slightly lower signal-to-noise ratio in the UV and VIS arms, but a slightly higher signal-to-noise ratio in the IR arm when compared to direct computation "by hand".

The simulated data was generated by taking ray-tracing input fluxes through an optical model and taking the efficiency of the telescope-instrument combination into account. These semi-realistic simulated data can then be used to test and prepare a data pipeline for the NTE. Additionally, the simulated data also helped fine-tune some parameters of the setup of the spectrograph before the final construction. Finally, the initial output from testing on the pipeline has also been compared to the output of the ETC. This showed a good similarity between the results and is therefore promising. The quality of the simulated data could also be improved by introducing more realistic effects, such as detector effects and cosmic rays, by using the detector simulation framework Pyxel. Overall, this thesis contributes to future work with the NTE by providing an exposure time calculator and creating simulated data to help the creation of a data pipeline, thereby enabling efficient observations from the moment the NTE is operational.

Contents

1 Introduction

Ground-based telescopes like the Nordic Optical Telescope (NOT) are limited in their observational capabilities compared to space-based telescopes since the light has to pass through the atmosphere of the Earth. However, there are a lot of advantages of ground-based telescopes. They are a lot cheaper to make and it is possible to continuously repair and add instruments to the telescope. The ability to add new instruments is currently being utilized at NOT. The NOT Transient Explorer (NTE) is a new instrument inspired by X-Shooter at VLT. [1] The main improvements it adds to NOT are simultaneous visual and near-infrared (NIR) imaging and spectroscopy. Additionally, the sensitivity in the NIR is improved for both imaging and spectroscopy. [2]

NTE is supposed to be available at all times for observations of transient events. Some of the transient events that will be observed with NTE are gamma-ray burst afterglows, type Ia supernovae, and core-collapse supernovae.
Gamma-ray bursts (GRBs) are highly energetic events with a short duration. GRBs that last longer than 2 s, are usually associated with powerful supernovae, while GRBs that last shorter than 2 s are associated with neutron star mergers. However, for long GRBs, there are cases where a bright supernova is not found, suggesting a different origin. Short GRBs are more uncertain as they are difficult to locate in the sky. Therefore it is important to study short GRBs more to explore their origin.
There already exists a lot of data from type Ia supernovae and therefore the most important thing when observing these supernovae is getting very detailed data. Since NTE will cover a wide wavelength range, the ends of the spectrum will not be cut off, and it will therefore be possible to study the near-UV and near-IR parts of the supernova spectrum.
Core-collapse supernovae are a type of supernovae that creates neutron stars or black holes. These events play a crucial role in the creation of elements heavier than iron, as well as many lighter elements such as oxygen and magnesium. Because of this, core-collapse supernovae have a big impact on star formation and it is therefore of great interest to study these supernovae. Additionally, they are candidates for dust production, and it could be possible to measure the Hubble Constant based on these. [3, 4]

Some non-transient objects that are of interest for NTE are active galactic nuclei

Figure 1: Picture of the NOT (image from [2]).

(AGN) and exoplanets. For exoplanets that pass in front of their star, it is possible to find out which elements their atmosphere consists of by studying the light from the star that passes through their atmosphere. This is interesting since the evolution and the habitability of the planet are dependent on the elements in the atmosphere. Supermassive black holes with an accretion disk are called active galactic nuclei, but only few supermassive black holes have an accretion disk. The emission caused by this accretion disk makes AGN the brightest non-transient source in the universe, but AGNs are not well understood today. Questions such as typical lifetime, role in galaxy interactions, and fuelling mechanisms remain unanswered. [3] The emission from the accretion disk covers a wide wavelength range, which makes NTE ideal for the study of AGN. [4]

This project consists of two parts, both of which contribute to the preparation for NTE. The final aim is pre-commissioning the software to a level where the majority of the software will be written before the instrument itself has been built.

One of the things that are needed when running NTE, is a pipeline that can process the raw data taken by the instrument. However, preparing a pipeline without data to test it on is challenging, and one of the goals of this project is therefore to produce a set of simulated data the pipeline can be tested on. Everything from bias frames to science spectra and cosmic rays should be included for a thorough test of the

pipeline.
In addition to testing that all the features of the pipeline are working, this simulated data can also be used for validating and optimizing the optical design of NTE.

The initial step in simulating data from NTE is using the optics design software ZEMAX OpticStudio [5], which is a ray tracing software used to trace the light through the detector. Once the arrival positions for the photons are obtained from ZEMAX, other software can be used to determine the fluxes. This is done using a modified version of Pyechelle [6].

Another important tool when using the NTE is an exposure time calculator [7]. This will help the user determine the optimal exposure time for the object they are observing, and in the process of creating NTE, it is a great tool for demonstrating the possibilities with this new instrument.
For this purpose, an existing exposure time calculator for X-Shooter is translated into Python and modified to work for the NTE.

2 Theory

2.1 Echelle spectrographs

Even though different existing tools are going to be used for the simulation of data, and the simulation therefore will not be done from scratch, it is still very useful to know the basics of how echelle spectrographs work. This is important in order to understand the final simulated data, as well as being able to ensure the simulations give the expected output.

Echelle spectrographs are a kind of spectrographs that can achieve high spectral resolutions with more efficient use of detector area than other spectrographs. Some of the most important components of an echelle spectrograph are an echelle grating and a cross disperser. The echelle grating will split the light into multiple orders. The higher of these orders will physically overlap on the detector unless cross dispersed. An example of this can be seen in figure 5.

Figure 2 shows a simple model of an echelle spectrograph. Further detail will be provided in the next sections, as the setup of the echelle spectrograph is explained.

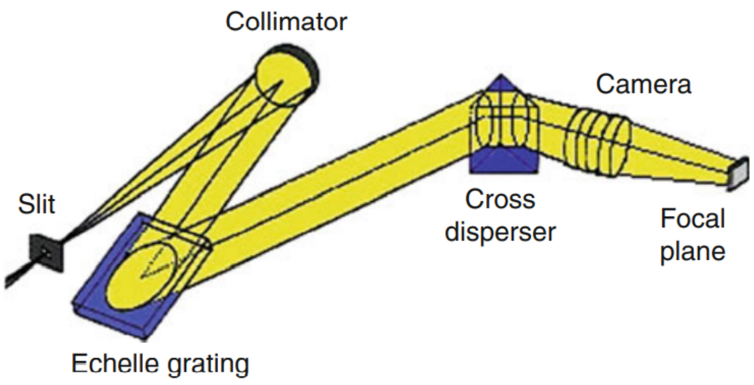


Figure 2: General example of an echelle spectrograph. The echelle grating and the cross disperser are described in section 2.1.1, while the slit, the collimator, the camera, and the focal plane are described in section 2.1.2 (figure from [8, p. 197]).

2.1.1 Dispersion of light

The most significant part of an echelle spectrograph is how it disperses light. Echelle spectrographs use a dispersion grating, more specifically an echelle grating, to disperse light into individual wavelengths. The grating has a pattern of grooves in the surface, so that when light is incident on the grating, is it diffracted according to the ruling density and angle of incidence, producing a spectrum of wavelengths.

This is central to the dispersion of light, as the refracted angle of the light depends on the wavelength of the light.

The echelle grating is characterized by a low groove density, relative to other gratings, and also by a large angle of incidence. This will do so that the echelle spectrograph is able to cover a large wavelength range.

The dispersion of the light can be described by the grating equation. The grating equation can be seen on eq. 1.

$$n\lambda = d\cos(\gamma)(\sin(\alpha) + \sin(\beta)) \tag{1}$$

The equation relates the angle of diffraction (β), the angle of incidence (α), the distance between grooves (d), the off-axis angle (γ), and the wavelength of the incident light (λ). This equation describes the basic idea that the echelle spectrographs work on. The general setup and illustration of angles can be seen in figure 3.

As a quick mathematical example, a constant diffraction angle can be considered

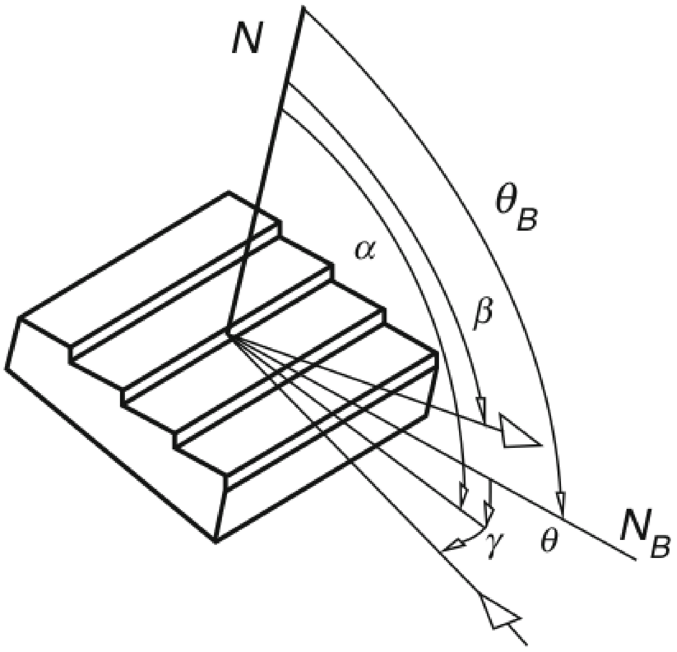


Figure 3: Illustration explaining the angles found in the grating equation, eq. 1 (figure from [8, p. 194]).

$\beta = \beta_c$, then the grating equation is as on eq. 2. [8]

$$\lambda_c(n) = \frac{1}{n} d\cos(\gamma)(\sin(\alpha) + \sin(\beta_c)) \tag{2}$$

Here λ_c is inversely proportional to the order number n, and the change in central wavelength will therefore be $\frac{d\lambda_c}{dn} \sim \frac{1}{n^2}$. [8]

To illustrate this the simple example of $\alpha = \gamma = 0°$ is considered. This gives for the grating equation $n\lambda = d\sin(\beta)$. Then for $n = 1$ and $d = \frac{1}{150}$mm the diffraction angle for a spectrum can be calculated. Here the range of visible light, from 400 nm to 700 nm, is considered. [8]

The angle of diffraction is therefore calculated as $\beta = \arcsin\frac{n\lambda}{d}$ and the results are shown in table 1. For orders 1 through 6 the result will look as in figure 4.

From figure 4 it is clear that higher orders overlap in the diffraction angle, but also that they are slightly shifted from each other. This means that a large range of the spectrum can actually be contained in a certain small geometric window $\Delta\lambda$, as shown in figure 4. [8]

λ [nm]	β [deg]
400	3.4
450	3.9
500	4.3
550	4.7
600	5.2
650	5.6
700	6.0

Table 1: Table of the diffraction angle at different wavelengths for the order n=1.

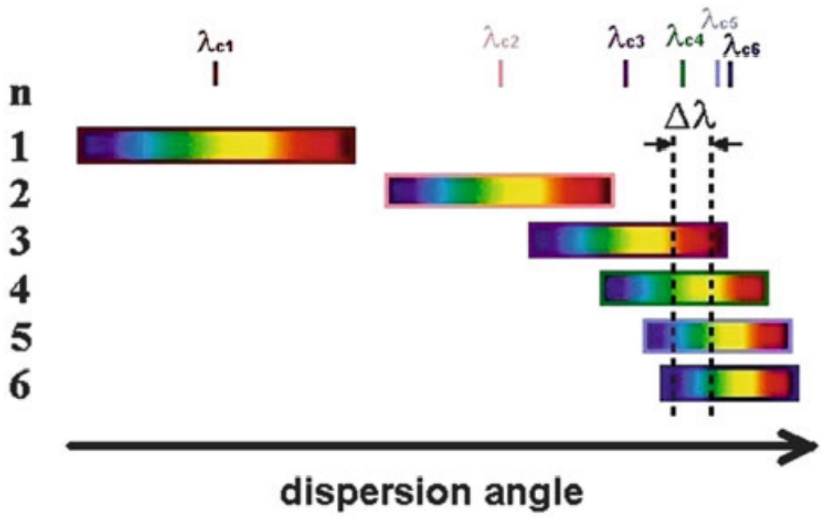


Figure 4: Illustration of the dispersion angle depending on the order number. $\Delta\lambda$ denotes a small physical area where a large wavelength range can be obtained if multiple orders are considered (figure from [8, p. 195]).

With multiple orders overlapping, a second grating (or prism) with a dispersion direction that is perpendicular to that of the first grating can be included to separate the different orders. This second grating or prism is called the cross disperser. An example setup of an echelle spectrograph can be seen in figure 2. [8] The resulting image, after the area $\Delta\lambda$ goes through a cross disperser, on the detector is shown in figure 5.

2.1.2 Optics

An echelle spectrograph will use a combination of lenses and mirrors to focus the light onto the detectors. The choice of optics and coating will also impact the overall efficiency of the spectrographs.

A key component is the entrance slit. It is a narrow opening that determines the

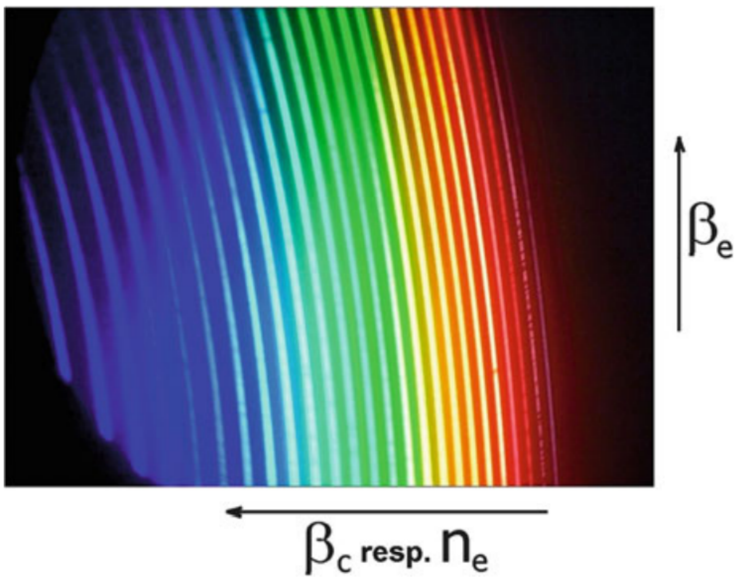


Figure 5: Example of a possible output from an echelle spectrograph. In this illustration, the main dispersion direction is on the y-axis, while the cross-dispersion is on the x-axis (figure from [8, p. 123]).

angular size of the observation on the sky. It will also, together with the grating, determine the resolution of the spectrograph. A narrower slit will yield a higher resolution. For the NTE there are different widths of slits to choose from, but they all have the same length. There is also a pinhole in addition to the other rectangular slits. The available slit widths on the NTE are as follows: 0.5", 0.8", 1.0", 1.2", 1.5", 1.7", 2.0", and 5.0". All slits have the same height of 22.8". In addition to these rectangular slits there is also a 0.5" diameter pinhole.

Another important part is the collimator. It is a lens or mirror system that parallelizes the light, that converts the diverging light from the slit, into a parallel beam.

After the light has been diffracted by the echelle grating and the cross disperser, it also has to be focused onto the detector by camera optics. The camera optics will usually consist of multiple lenses that form an image of the light onto the detector.

2.1.3 Layout of the NTE

The NTE is not only a spectrograph but also an imager. A complete layout of the NTE can be seen in figure 6. However, this project will only focus on the

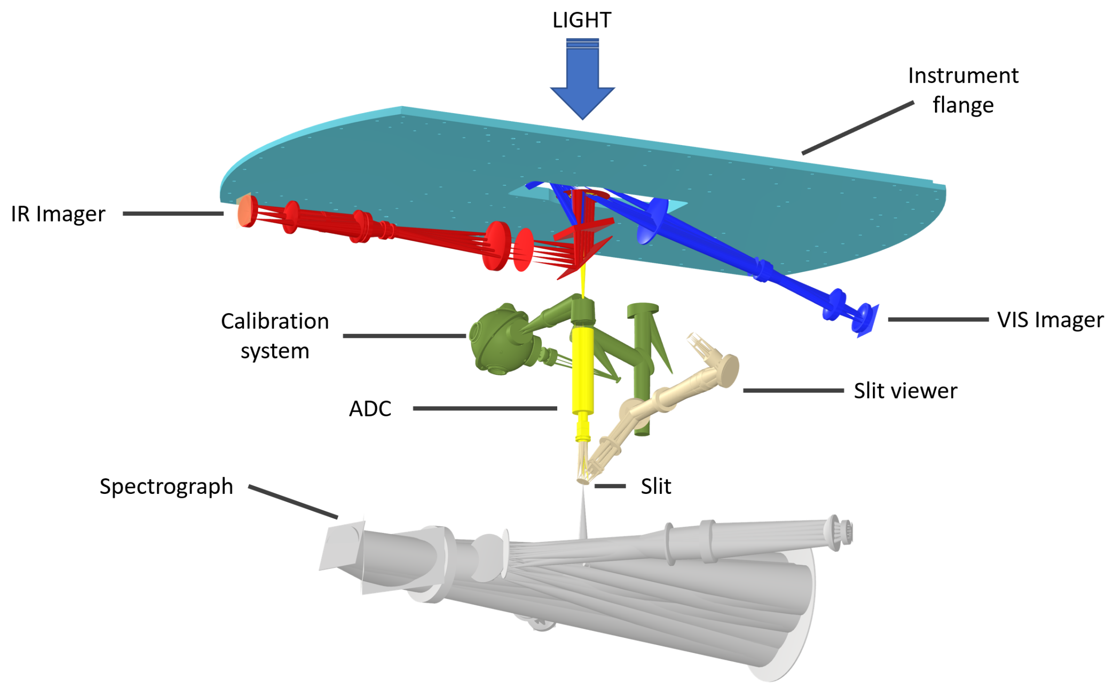


Figure 6: Illustration of the NTE optical path. Figure provided by Joonas Kari Markku Viuho.

spectrograph, which can be seen in figure 7. The slit wheel is above the fold mirror. The fold mirror turns the light 90° to be in the plane of the spectrograph. From the fold mirror, the beam is led to the collimator. Then the beam goes through the corrector lens before hitting the cross disperser, the grating, and back through the cross disperser. Passing through the cross disperser twice provides additional cross dispersion. From there it goes back to the collimator and then onto the field focus. All orders 3-21 are drawn in the field focus, where the appropriate orders are reflected to the individual UVB, VIS, and NIR cameras with field slicing mirrors.

2.1.4 Detectors

Detectors are extremely important for the performance of the telescope. The detector, usually a charge-coupled device (CCD) or a complementary metal-oxide-semiconductor (CMOS) sensor, is what captures the light and converts it to a digital signal that can be analyzed on the computer.

There are several considerations that should be made when selecting a detector for a spectrograph, such as its ability to detect a specific range of wavelengths, its sensitivity, and its spatial resolution to minimize the loss of information.

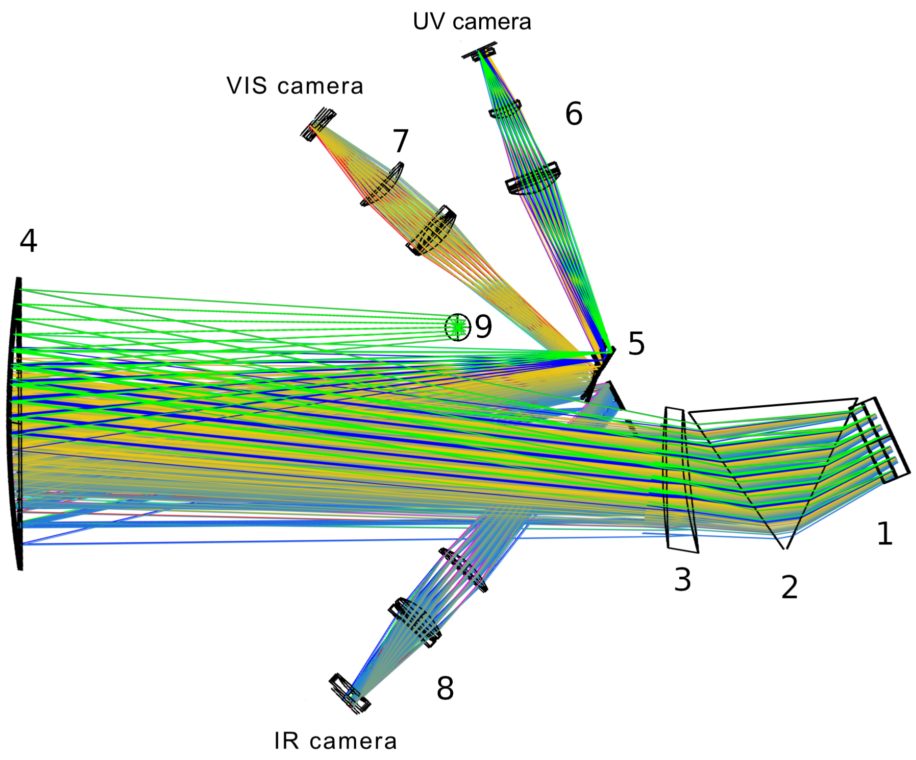


Figure 7: Layout of the spectrograph on the NTE. The components of NTE are numbered where 1 is the echelle grating, 2 is the cross disperser prism, 3 is the corrector lens, 4 is the collimator, 5 is the field focus, 6 is the UV camera lens setup, 7 is the VIS camera lens setup, 8 is the IR camera lens setup and 9 is the fold mirror. Figure provided by Joonas Kari Markku Viuho.

Three different detectors are used in the NTE. They are an Electron Multiplication CCD (EMCCD) for the UV arm, a Skipper CCD for the VIS arm, and an H2RG Murcury-Cadmiun-Telluride detector for the NIR arm. These detectors will be explained in the next sections.

2.1.4.1 General CCD

The general idea for a CCD is to have an array of light-sensitive cells, or pixels, that can convert the incoming photons to electrical charge. Each pixel contains a metal-oxide-semiconductor (MOS) capacitor, which is the core of observing the photons and storing them as electric charge.

When the charge is stored in the capacitor, and the detector is read out, the charges are shifted line by line to a serial register from which the pixels will be moved one by one to the output amplifier. The charge is then measured and converted to a digital unit that can be stored in memory and seen as an image on the computer. [9]

The NTE has two different kinds of CCDs, an EMCCD and a Skipper CCD, used for the UV arm and the VIS arm respectively. They will be described in the following sections.

2.1.4.2 EMCCD

For the UV arm of the spectrograph, an EMCCD is used. EMCCD is short for Electron-Multiplying CCD. One of the advantages of EMCCDs is their ability to increase signal compared to noise. Since the readout noise of the amplifier is in a number of electrons, then with the Electron-Multiplication the readout noise is insignificant compared to the signal after multiplication. Additional to increasing signal compared to noise, it is also possible to run the readout with higher bandwidth due to the increase in noise, from increasing speed of the readout, being insignificant. It is typical for EMCCDS to be a frame transfer CCD with a storage area, from where the image can be read out while the next frame is being exposed. This means that it is possible to transfer the image to the storage area and start exposure again while the image is being read out, increasing the speed of the detector further. Figure 8 shows a setup of an EMCCD. [10]

To read out the CCD, the charge is shifted through the readout register, and then through the multiplication register, where the amplification happens prior to readout by the charge amplifier. What happens in the multiplication register is something, that also happens in a regular CCD, called 'Clock-Induced Charge'. When clocking a charge, there is a small chance to create additional charges by impact ionization. [10]

Impact ionization is a result of the charge having enough energy to create an electron-hole pair, and so another charge is created. The chance of impact ionization is small, but it can be increased by increasing the energy of a charge by clocking the charge with a higher voltage. Additionally, increasing the number of register stages will give more opportunities for impact ionization and increase the overall gain. Therefore there are hundreds of register stages, so even if the probability of impact ionization is small, over the whole register the gain will be high. [10]

One thing to note about EMCCDs is that given an output number of electrons X, then there is no way of knowing exactly how many electrons were in the input signal.

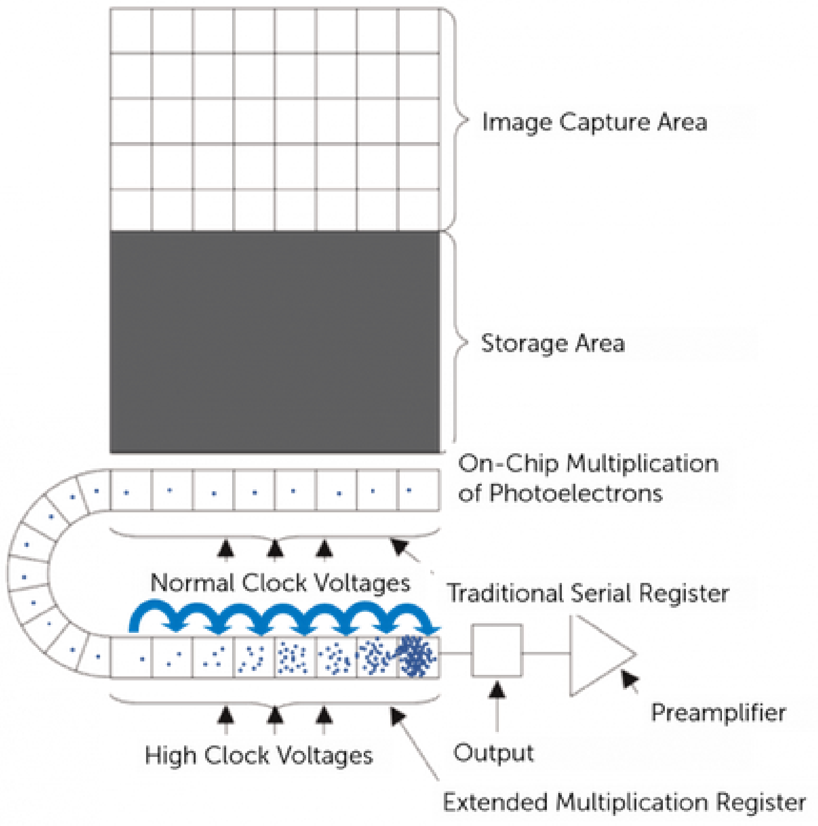


Image Capture Area
Storage Area
On-Chip Multiplication of Photoelectrons
Normal Clock Voltages
Traditional Serial Register
Preamplifier
High Clock Voltages
Output
Extended Multiplication Register

Figure 8: Setup example of an EMCCD. The photons will hit the image capture area. When the image is read out, it is first transferred to the storage area. From there, it goes through a series of cells known as the multiplication register. After the multiplication register, the data is then read out (figure from [11]).

There is a high probability of it being from different input signals due to the chaotic nature of the cascade in electron multiplication. This can lead to additional noise. Figure 9 shows a graph of the probability density functions for 1-5 electrons in the input signal as a function of the output signal. [10]

Compared to normal CCDs, an EMCCD will have a higher sensitivity in low light, due to amplification of detected signal without increasing readout noise. Additionally, a normal CCD will be very slow in low-light conditions due to reducing readout speed in an effort to reduce readout noise. The EMCCD will have less noise and a lower exposure time for the same image as a normal CCD. [11]

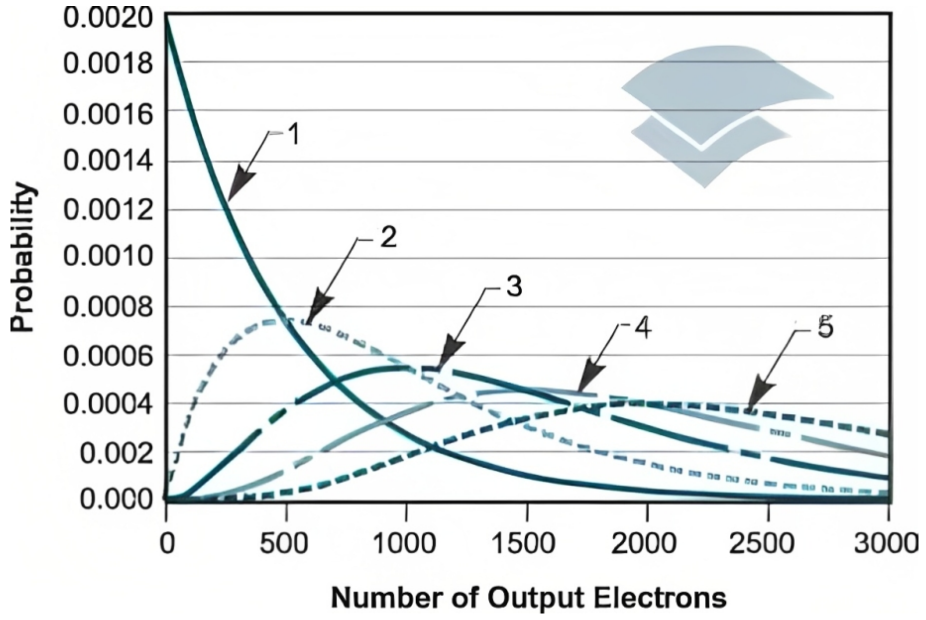


Figure 9: Graph of the probability density functions for 1-5 input electrons as a function of the number of output electrons. The overlapping probability density functions indicate that it is not possible to determine the exact number of input electrons based on the output (figure from [10]).

2.1.4.3 Skipper CCD

For the visual arm of the spectrograph, a Skipper CCD is used. [12] The main difference between a regular CCD and a Skipper CCD is the fact that the Skipper CCD allows multiple reads without affecting the existing charge. The read noise in the Skipper CCD should reduce with multiple reads as

$$\sigma_{skipper} = \frac{\sigma_1}{\sqrt{N}} \tag{3}$$

If the pixel is read enough times, the charge can be measured with sufficient sensitivity to determine the exact number of electrons in it. [13]

The reason it is possible to read the charge more than once is that, in the Skipper CCD, it is possible to move the charge back and forth between the last pixel in the readout register and the sense node as desired, without destroying the charge.

2.1.4.4 General CMOS

The difference between a CCD and a CMOS detector is that, in a CMOS detector, it is possible to address and read out each pixel independently. This also means that the pixels can be processed in parallel, increasing the bandwidth of the readout.

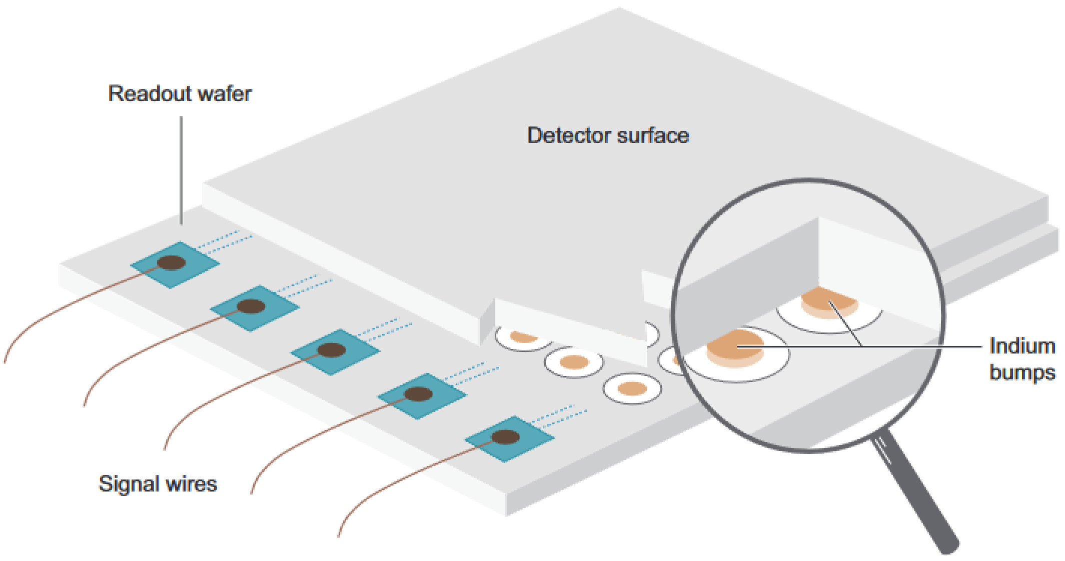


Figure 10: Illustration of setup of the H2RG detector. This shows that a detector surface is fused onto the CMOS circuity with Indium bumps (figure from [15, p. 89]).

However, due to individual systems for each pixel, the resulting image will have less uniformity. Additionally, the physical size of the systems may reduce the area available for capturing photons. [14]

2.1.4.5 H2RG

For the IR arm of the spectrograph, an H2RG is used. While silicon is a common photosensitive material used in detectors, it is not optimal for detecting infrared radiation, due to silicon's sensitivity cutoff at 1.1µm. The H2RG detector uses a HgCdTe (mercury cadmium telluride) photodiode array on top of the readout circuit instead. Like a CMOS detector, each pixel in the H2RG has its own amplifier, and the signal is then transmitted to a small number of output amplifiers. To produce this detector, a grid of diodes is created on a silicon substrate, along with a grid of contacts on the readout wafer. Indium bumps are connected to the silicon structure and the photosensitive HgCdTe is deposited on top. An illustration of the setup can be seen on figure 10. [15]

3 Exposure time calculator

All major telescopes have an exposure time calculator since it is a very useful tool when preparing an observation. An exposure time calculator (ETC) is a tool used by

astronomers to determine the signal-to-noise ratio of a potential observation. This makes it possible to estimate the optimal exposure time and thereby not waste time on the telescope by either doing too long or too short exposures. An unnecessarily long exposure time will take time out of the limited amount of observation time and might even result in the image being overexposed. A too-short exposure time will result in images that are dominated by noise and therefore less useful.
Furthermore, an exposure time calculator can help estimate whether a specific target would be observable by the telescope. If the target is too dim to give a sufficient signal-to-noise ratio, then it might be more valuable, and a better use of the telescope time, to choose another brighter object with a higher signal-to-noise ratio.

An exposure time calculator takes an array of parameters into account when calculating the signal-to-noise ratio, including the telescope efficiency, the telescope size, the sky background, the atmospheric absorption, and the seeing. Using these parameters together with the exposure time and number of exposures for the specific observation, it is possible to predict the signal-to-noise ratio of a given observation.

3.1 The original ETC

The exposure time calculator for NTE is based on an exposure time calculator originally written by Bjarne Thomsen, which has previously been used for X-Shooter. Bjarne Thomsen is a former associate professor at Aarhus University, who is now retired. The calculator was written in the language IDL which makes it more difficult to update or make changes to the exposure time calculator since it is a language that is not as widely known as e.g. Python. Apart from modifying the exposure time calculator to fit NTE, it would therefore also be beneficial to translate the code to Python, to simplify future work.

The general idea of the exposure time calculator made for the NTE is to take a set of spectra, interpolate over the spectra and combine all of them. When combining the spectra a lot of telescope properties are also added. The needed input is:

- Spectrum of the sky
- Spectrum of the target
- Atmospheric extinction
- OH-lines

Both OH-lines and the atmospheric extinction are in different files even though it is not strictly necessary. This is done because extinction and OH flux density do not necessarily depend on the same parameters. Therefore it is made possible to change them individually if needed.

This creates a spectrum close to what will be observed with NTE, and a plot of the signal-to-noise ratio as a function of wavelength.

3.2 Translating to Python

When translating the original code from IDL to Python in order to ease future changes to the exposure time calculator, multiple aspects should be taken into consideration.

Firstly some of the in-built functions from IDL caused some confusion. In IDL functions like 'where' and the logical operator 'NOT', do not align with how the same functions work in Python. An example of this difference is 'NOT 0 = 255' in IDL, whereas 'not 0 = True' in Python. However, in the original exposure time calculator 'NOT' is only used for determining which functions should be used at different wavelengths. It is therefore fairly simple to rewrite the code to fit the way Python interprets 'not'.

Since IDL is a lot faster than Python it has not previously been necessary to reduce the run time of the code. However, translating the original code to Python without making any changes will result in an exposure time calculator that is extremely slow. An obvious way to reduce the run time slightly is by reducing the amount of unnecessary calculations, such as making a calculation inside a for loop which could easily be moved outside the for loop. Another way is to use a profiler on the code to see where the majority of the time is spent. In this case, a profiler clearly shows that the biggest problem is the interpolations. The speed of the interpolations can be improved by using SciPy [16] interpolation functions. This way it is only necessary to create a function for interpolating once, which can then be evaluated multiple times. This improved the speed of the code by an order of magnitude.

This improved the run time enough to make the exposure time calculator usable, but in order for the exposure time calculator to be convenient the run time needs to be cut down even further. This can be done by replacing some of the native Python loops with NumPy [17] functions.

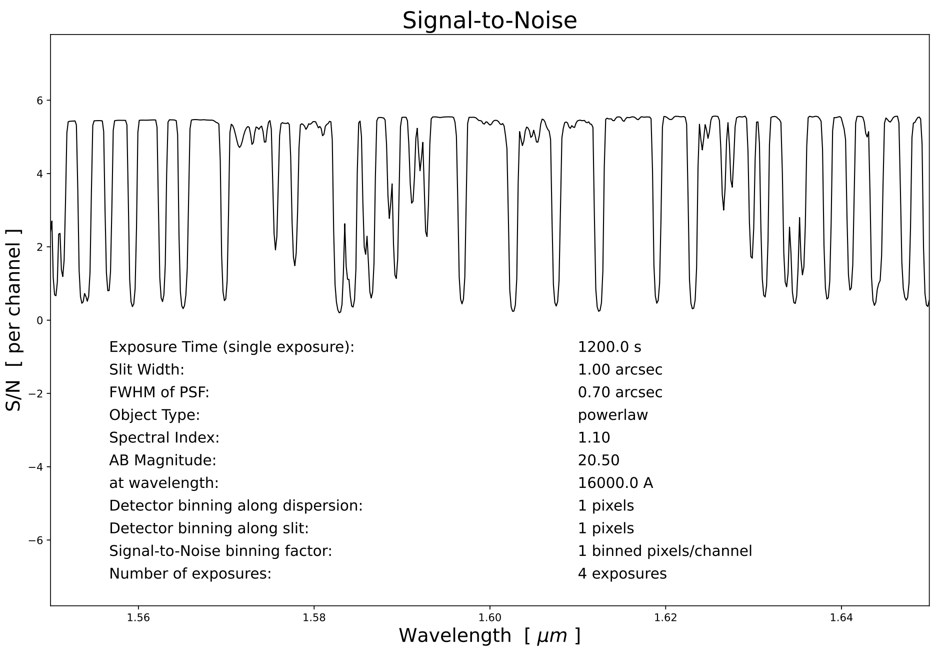
For sparsely sampled data sets, interpolation caused issues since the data set had relatively large wavelength intervals without any data points. This caused the interpolation in these areas to be very far from what would be reasonable. This problem was solved by using linear interpolation instead of cubic spline interpolation, and thereby removing the option for the interpolation to do something unexpected in areas without many data points. Additionally, data with a better resolution is also used if easily available. For this project, most data was acquired through ESOs SkyCalc. For references on SkyCalc see [18, 19, 20].

3.3 Comparison

The initial testing of the exposure time calculator was done by comparing the output of the rewritten code in Python to the output of the original code in IDL. Both outputs were generated using the input values from the original code which were meant for X-shooter [21]. An example of these two outputs is shown in figure 11. More examples are available in appendix A.1. The output from the Python translation looks identical to the output from the original IDL code, but there are small differences that might be caused by slightly different interpolations.

The example in figure 11, as well as the additional examples in appendix A.1, were all done with a wavelength range of 0.1µm = 1000Å and an AB magnitude of 20.50 at the centre of the range. Table 2 shows the median signal-to-noise ratio in three different wavelength ranges using respectively the Python version and the IDL version. For the wavelength range that is covered by both the VIS detector and the IR detector, a median is found for each detector.
The median signal-to-noise ratios found in table 2 show that the Python version and the IDL version derive very similar signal-to-noise ratios. Furthermore, the example in figure 11 also suggests that the differences between the IDL version and the Python version are quite small.

Apart from the signal-to-noise ratio, the exposure time calculator will also simulate spectra, and an example of such a spectrum can be seen in figure 12. The figure shows two spectra created from the same input but by respectively the Python version and the IDL version of the exposure time calculator. These spectra show a general similarity, but since there is a random element in the generation of these spectra, they will be slightly different. Since the larger peaks are generally at the same wavelengths, the spectrum generated by the Python code is considered to be



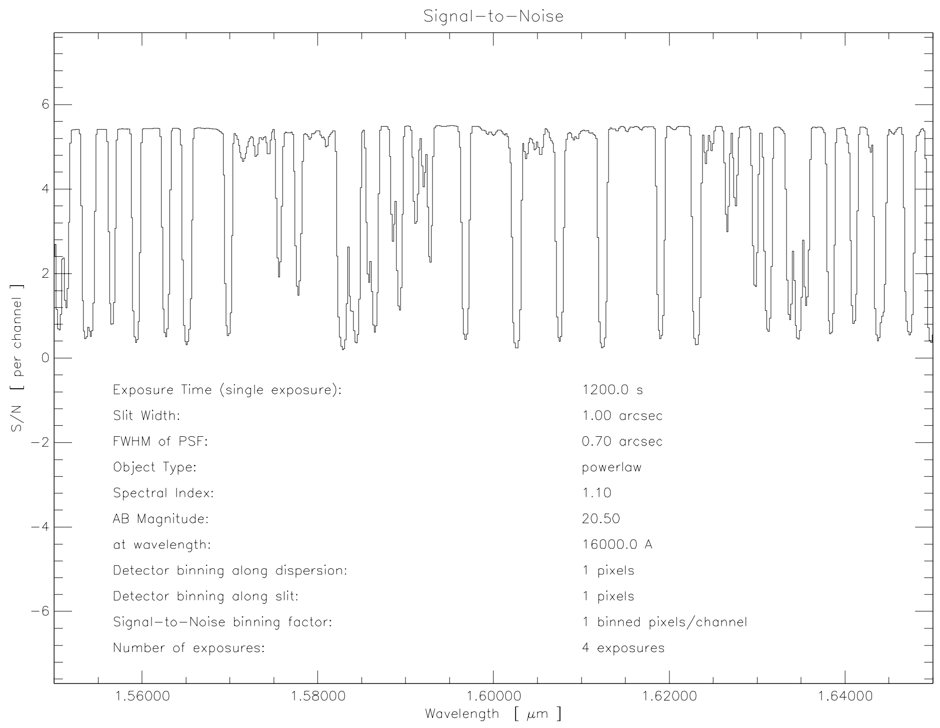


Figure 11: Comparison between output S/N of the Python code (top) vs. the IDL code (bottom). No moon is included in the noise spectrum for this comparison. The Python and the IDL version yield almost identical results.

Wavelengths [Å]	Python	IDL
5500 - 6500	Median VIS S/N = 6.827	Median VIS S/N = 6.827
7000 - 8000	Median VIS S/N = 7.586 Median IR S/N = 6.605	Median VIS S/N = 7.591 Median IR S/N = 6.715
15500 - 16500	Median IR S/N = 5.232	Median IR S/N = 5.180

Table 2: Median values in 3 different wavelength ranges to compare the Python and the IDL outputs.

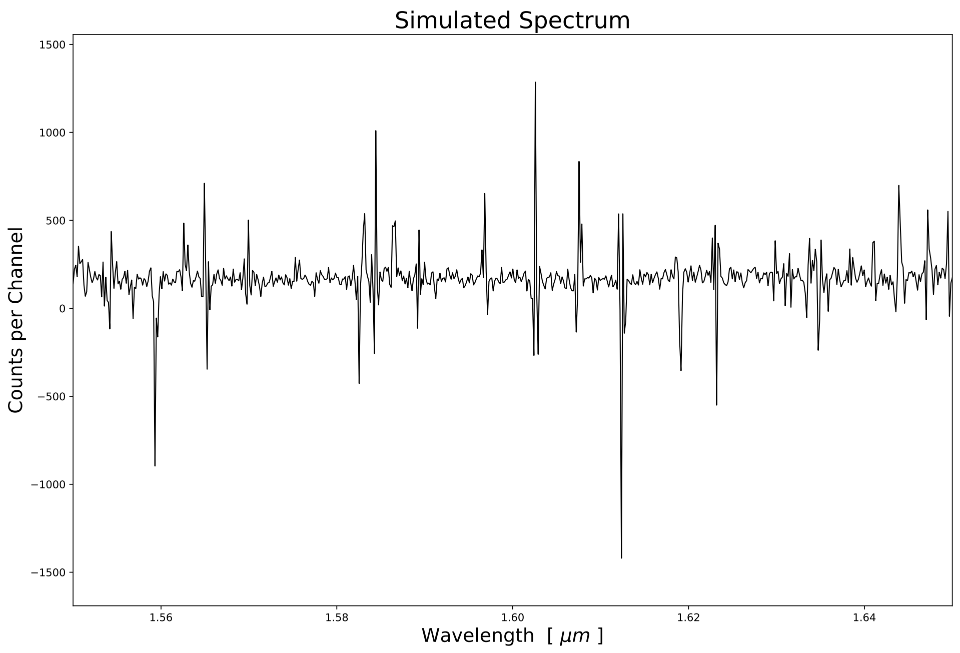
very similar to the spectrum generated by the original IDL code.

3.4 Necessary alterations

In order for the exposure time calculator to be compatible with the NTE instead of X-Shooter, a number of things will have to be changed and added.

Initially, a third spectral arm, the UV arm, has to be included. The NTE contains three arms, which makes it able to cover a wide range of wavelengths in great detail. However the original exposure time calculator only allows for two spectral arms, and the addition of this extra arm is therefore the main alteration to make the exposure time calculator compatible with the NTE.

A smaller alteration is to manage all the data files and variables which have been hard-coded into the original exposure time calculator. Variables, such as wavelength ranges for the arms and their read-out noise, are instead put into a separate configuration file where they can be easily changed if needed. Many of these variables can be obtained from [22]. In addition, data files that are specific to X-Shooter need to be changed to the corresponding data files for NTE. Some of these data files, such as the sky transmission and sky absorption, can be produced using ESO SkyCalc. The total efficiency files depend on the efficiencies of the detectors, coatings, grating, and lenses. The telescope mirrors were expected to be coated with aluminium, the internal surfaces with a UV-enhanced silver coating, the grating efficiency was assumed to be 70%, and the transmissive objects were assumed to have a single layer of magnesium fluoride (MgF2) coating and therefore transitions between air and glass had an assumed 98% efficiency. Coating efficiencies were provided by Joonas Kari Markku Viuho. The Skipper CCD efficiency was taken from [23], the EMCCD efficiency was obtained from a datasheet provided by Teledyne, and the H2RG efficiency was taken from a test report on the detector.



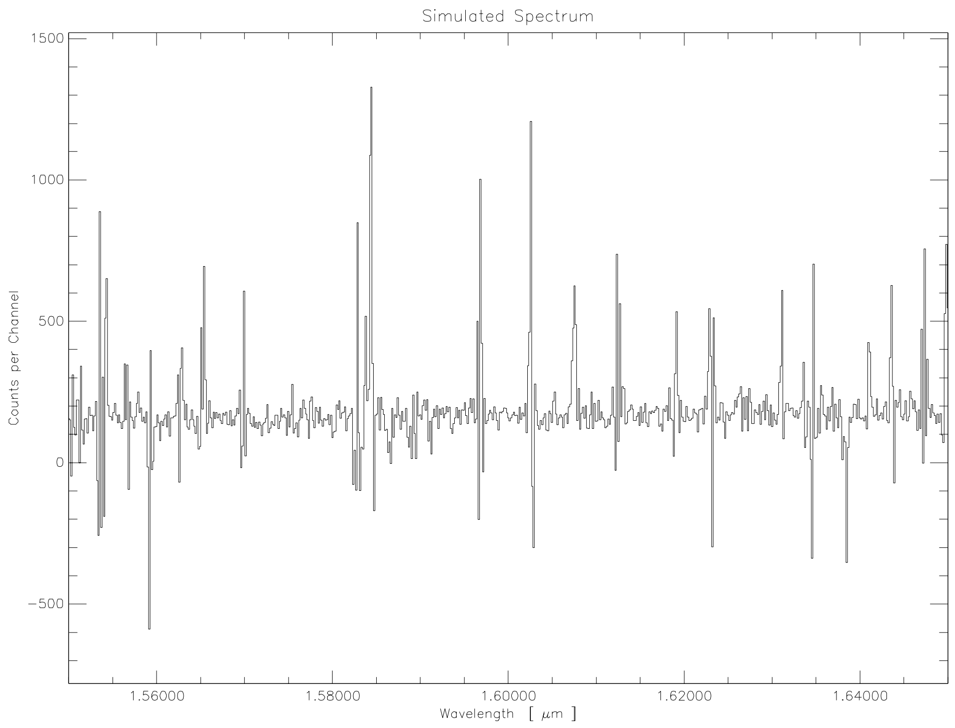


Figure 12: Comparisons between the generated spectrum from the Python version (top) vs. the IDL version (bottom) using the same parameters as in Figure 11. Differences can be explained by a random fluctuation based on the S/N.

3.5 Validation of the results

With the exposure time calculator altered to work for the NTE, it is necessary to check whether the flux of the output spectrum is correct. By calculating the flux at different wavelengths by hand, it will be clear if the output of the exposure time calculator is correct. The equation used for the hand calculations is

$$S = f_0 \cdot 10^{-m/2.5} \cdot A \cdot E \cdot \frac{\lambda}{h \cdot c} \cdot 10^{-0.4 \cdot m_a \cdot k} \tag{4}$$

where S is the observed signal, f_0 is the flux zero point, m is the magnitude of the object in the specific band (in Vega magnitudes), A is the area of the telescope, E is the efficiency of the telescope in the band, m_a is the air mass, λ is the wavelength for observation, and k is the extinction.

To compare the hand calculation with the output of the exposure time calculator, it is necessary that both use an object with the same magnitude. Since the exposure time calculator uses the AB magnitude and eq. 4 uses the Vega magnitude, a conversion between the two magnitudes is needed. These conversions can be found in [24], and table 3 shows these conversions for each of the bands. For all the hand calculations a Vega magnitude of 20 was used, and the exposure time calculator used the corresponding AB magnitude.
The flux zero point, f_0, the extinction, k, and λ are also looked up and provided in table 3, while an air mass of 1.2 is used in both the hand calculation and the exposure time calculator.

The efficiencies are a little more complicated. All the efficiencies are taken from the three efficiency files used by the exposure time calculator. For most of the bands, the wavelength λ is only covered by one of the three efficiency files, but for the B band λ is covered by both the UV and the VIS efficiency files. Therefore the signal in the B band is calculated for both the UV arm and the VIS arm respectively.

Additionally, the in-hand calculations assume that all light comes through the slit. Therefore the seeing is set very low in the ETC, as well as increasing the slit width, just to make sure that all light would get through.

The results from both the hand calculations and the exposure time calculator can be seen in table 4. This table clearly shows how the accuracy of the exposure

	U	B	V	R	I	J	H	K
λ [nm]	360	438	545	641	798	1220	1630	2190
$m_{AB} - m_{Vega}$ [mags]	0.79	-0.09	0.02	0.21	0.45	0.91	1.39	1.85
$f_{\lambda,Vega}$ [e-11 erg cm^{-2} s^{-1} Å^{-1}]	417.5	632.0	363.1	217.7	112.6	31.47	11.38	3.961
k [mags/air mass]	0.55	0.25	0.15	0.09	0.06	0.0	0.0	0.0
Efficiency (E)	0.05	0.34/0.28	0.38	0.36	0.21	0.41	0.44	0.41

Table 3: Values used. wavelength λ [24], mag difference $m_{AB} - m_{Vega}$ [25], zero point $f_{\lambda,Vega}$ [26] and extinction k [27]. Efficiencies from the UV detector are used for the U and B bands, efficiencies from the VIS detector are used for the B, V, R, and I bands, while efficiencies from the IR detector are used for the J, H, and K bands. For the B band, the efficiency of 0.34 is from the UV detector, while the efficiency of 0.28 is from the VIS detector.

	U	B (UV)	B (VIS)	V	R	I	J	H	K
Hand calc	105.1	1713.5	1414.6	1522.6	1111.3	428.7	378.1	197.3	85.1
ETC UV arm	84.5	1559.9							
ETC VIS arm			1288.6	1490.7	1101.8	435.2			
ETC IR arm							381.4	204.1	90.5
ETC/Hand	0.80	0.91	0.91	0.98	0.99	1.02	1.01	1.03	1.06

Table 4: Calculated signals in different bands, compared to the ETC in the relevant arms. All numbers are in units of counts/s/Å.

time calculator varies over the different bands. In the U band, the ETC is about 20% lower than the hand calculation, while in the K band, the ETC is about 6% higher than the hand calculation. Generally, the ETC/Hand ratio increases with the wavelength.

3.6 Description of the code

The main part of the code is split into three separate parts, one for each of the three arms. However, the UV arm and the VIS arm are identical.

Before the code splits into these three separate arms, the 'config.ini' file is read in and units are converted where it is needed. If the template option is used instead of the Planck or power law, this would also be where the template is read in. Lastly it is determined which arms should be used for the specified wavelength range.

This is where the code divides into the three separate arms.

3.6.1 UV arm/VIS arm

Firstly an initialization function is used which reads in the files specific to the arm. For both the UV arm and the VIS arm these files are the sky spectrum and the quantum efficiency matching the arm. Furthermore, independent of the arm, the extinction files are read in for the entire wavelength area covered by the NTE. The reason multiple extinction files are read in was covered in section 3.1. The initialization function also calculates some constants that are needed later, such as the fraction of the flux that passes through the slit, and the disc scale of the object compared to σ of the point spread function (PSF).

After the initialization function, interpolations are done for each of the data files which were just read in. Making all the interpolations at this point and then saving them for later use was one of the alterations that were made to reduce the run time.

Next, a wavelength array for the arm needs to be created. To make this wavelength array logarithmic, the centre wavelength of this logarithmic wavelength range is found.

$$\lambda_0 = \sqrt{\lambda_{min} \cdot \lambda_{max}} \tag{5}$$

Knowing the centre wavelength, the length of the wavelength array can now be found.

$$N = \left\lfloor \lambda_0 \cdot \frac{\log \frac{\lambda_{max}}{\lambda_{min}}}{d_{\lambda_0}} \right\rceil + 2 \tag{6}$$

Here $\lfloor x \rceil$ refers to the rounding of x, and d_{λ_0} is the pixel size in wavelength units.

Then linear array of length N and in the range $[0, 1]$ can be created,

$$a = \frac{\text{arange}(N)}{N-1} \tag{7}$$

and this array can then be used to create the logarithmic wavelength array.

$$\lambda = \lambda_{min} \cdot \left(\frac{\lambda_{max}}{\lambda_{min}} \right)^a \tag{8}$$

This is where the actual calculation of the signal-to-noise ratio begins.

First, the interpolation of the quantum efficiency file is evaluated at wavelengths from the new logarithmic wavelength array. Next, the sky emission is calculated.

The sky emission is generated by convolving the sky spectrum with a Gaussian kernel. In a loop, each wavelength is looked at individually together with its Gaussian σ. The surrounding wavelengths are generated for this specific wavelength based on a step size $d\lambda$. Then the sky flux can be evaluated for each of those points as

$$F_{sky}(\lambda) = d\lambda \cdot T(\lambda)^{m_{air}} \cdot E_{sky}(\lambda) \tag{9}$$

where T is the interpolated transmission, E_{sky} is the interpolated sky spectrum, and m_{air} is the air mass. Here F is in the units of e^{-}/m^{2}/s/arcsec^{2}. The unit of the flux is then converted to e^{-}/pix/exposure by multiplying it with the exposure time, telescope area, pixel length, and slit width.

The Gaussian weighted sum of this transmitted spectrum is calculated as:

$$G_{sky}(\lambda_0) = \frac{\sum\left(F_{sky} \cdot \frac{e^{-0.5\cdot\left(\frac{\lambda-\lambda_0}{\sigma_0}\right)}}{2\pi}\right)}{\sigma_0} \tag{10}$$

where λ is the generated wavelengths around the specific wavelength λ_0 (with a range of $[-12\sigma_0, +12\sigma_0]$) that has a corresponding σ_0. This means that any given wavelength on the original logarithmic scale will have a corresponding G_{sky}.

The final sky emission in e^{-}/pix/exposure is calculated using

$$S_{sky}(\lambda) = (A \cdot t \cdot L_{pix} \cdot W_{slit}) \cdot \eta(\lambda) \cdot D\lambda \cdot G_{sky}(\lambda) \tag{11}$$

Here A is the area of the telescope, t is the exposure time, L_{pix} is the length of a pixel in arcsec, W_{slit} is the width of the slit in arcsec, $D\lambda$ refers to the step size of the output spectrum, and η is the efficiency.

The flux of the observed object can be calculated in almost the same way. Here the unit of the flux is e^{-}/exposure and the equation is

$$S_{obj}(\lambda) = (A \cdot t \cdot s_{eff}) \cdot \eta(\lambda) \cdot D\lambda \cdot G_{obj}(\lambda) \tag{12}$$

where s_{eff} is the fraction of the flux that passes through the slit.

To calculate the signal-to-noise ratio per pixel of a profile fit along the slit, first, a semi-realistic object profile is generated by first defining a function

$$hgauss(x, s) = e^{-2.0 \cdot (\sinh(0.5 \cdot \text{asinh}(x \cdot s))/x)^2} \tag{13}$$

And then the profile is

$$P(\lambda) = hgauss(h_s, x_{pixel}(\lambda)/\sigma_{pix}) \tag{14}$$

where h_s is the disc scale divided by the PSF σ, σ_{pix} is the Gaussian seeing in pixels along the slit, and x is the position of the pixel in the slit.

P is then normalized, and the background variance per pixel is calculated.

$$\text{Var}_{bg}(\lambda) = S_{sky}(\lambda) + I_{dark} + RON^2 + (f_{error} \cdot S_{sky}(\lambda))^2 \tag{15}$$

This can be used to calculate the weight of a pixel.

$$W(\lambda) = \frac{1}{S_{obj}(\lambda) \cdot P(\lambda) \cdot \text{Var}_{bg}(\lambda)} \tag{16}$$

Then the profile is shifted such that the sum of $W \cdot Q$ is zero.

$$Q(\lambda) = P(\lambda) - \frac{\sum(W \cdot P)}{\sum W} \tag{17}$$

And then the S/N in that pixel would be

$$S/N_{pix}(\lambda) = S_{obj}(\lambda) \cdot \sqrt{\sum(W \cdot Q^2)} \tag{18}$$

And then the S/N for multiple exposures

$$S/N_{pix}(\lambda) = \sqrt{n_{exp}} \cdot S/N_{pix_{single_{exp}}}(\lambda) \tag{19}$$

And the output spectrum can also be calculated as

$$S(\lambda) = \left(1 + \frac{\mathcal{N}(0,1)}{S/N_{pix}(\lambda)}\right) \cdot n_{exp} \cdot S_{obj}(\lambda) \tag{20}$$

Where $\mathcal{N}(0,1)$ is a random number from a normal distribution with mean 0 and standard deviation 1.

3.6.2 IR arm

For the IR arm, it is necessary to also include the OH lines and thermal emission, however, no sky. Therefore, the initialization function imports the OH emission spectrum instead of the sky emission in the IR arm.
Like eq. 9, the OH flux can be found at each wavelength point as

$$F_{OH}(\lambda) = T(\lambda)^{m_{air}} \cdot E_{OH}(\lambda) \tag{21}$$

where E_{OH} is the interpolated OH spectrum.

However, the continuum between the OH lines also needs to be calculated and added together with the OH emission to create a complete sky spectrum for the IR arm. This continuum is made by choosing two wavelengths (one in the H band and one in the J band) and their corresponding flux and making a linear interpolation over them. The two points are $\lambda_J = 1.25$ µm with a flux of $F_J = 310.0$ ph/s/m^2/µm/arcsec^2, and $\lambda_H = 1.665$ µm with a flux of $F_H = 590.0$ ph/s/m^2/µm/arcsec^2. [28] The linear interpolation is done using

$$HoJ = \ln\left(\frac{\lambda_H}{\lambda_J}\right) \tag{22}$$

$$E_{cont}(\lambda) = F_J^{\frac{\ln\left(\frac{\lambda_H}{\lambda}\right)}{HoJ}} \cdot F_H^{\frac{\ln\left(\frac{\lambda}{\lambda_J}\right)}{HoJ}} \tag{23}$$

where E_{cont} is the interpolated continuum spectrum. The continuum flux at each of the wavelengths in the wavelength array can then be calculated as

$$F_{cont}(\lambda) = d\lambda \cdot T(\lambda)^{m_{air}} \cdot E_{cont}(\lambda) \tag{24}$$

The sky continuum and the OH emission are added together using the weighted sum from eq. 10. Then for the OH lines, first select all lines within $12\sigma_0$. Then also add the Gaussian weighted sum of the OH lines in the same manner as before.

The other difference between IR and the other arms is the addition of thermal emission in IR. Assuming a temperature of $\tau = 288$K and an emissivity of $\varepsilon = 0.25$ the flux is calculated as

$$F_{thermal}(\lambda) = (1 - (1 - \varepsilon) \cdot \tau) \cdot P_{law}(\lambda, \tau) \tag{25}$$

where P_{law} here denotes the Planck law for black body radiation. Apart from the different sky emissions and the added thermal flux, the rest of the calculations are identical to the UV and VIS arms.

3.7 Final output of the ETC

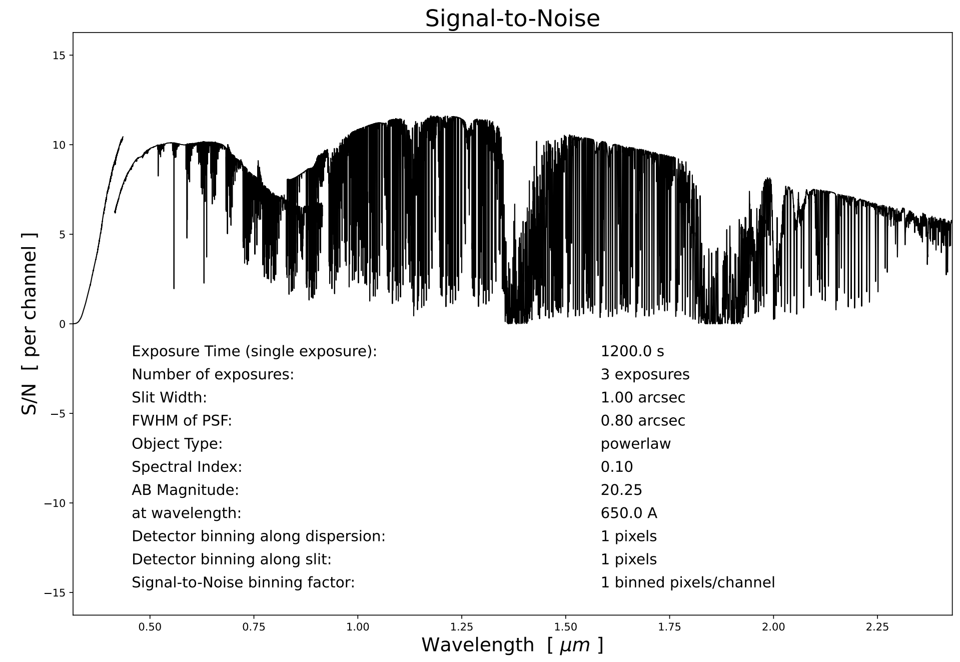
The output from the exposure time calculator is two figures, one of the simulated spectrum and one of the signal-to-noise ratio. On top of the two plots, the min, max, mean, and median of the signal-to-noise ratio for each of the used arms will be printed in the terminal.
Figure 13 shows an example of the two plots, while the output in the terminal will look something like this:

```
Median UVB S/N = 3.623247372345446
Mean UVB S/N = 4.325438922532742
Min UVB S/N = 0.004055276057292641
Max UVB S/N = 10.44167207697164
Median VIS S/N = 9.13503099891106
Mean VIS S/N = 8.48538449322476
Min VIS S/N = 1.330853430598455
Max VIS S/N = 10.178974307685117
Median IR S/N = 7.400861758421662
Mean IR S/N = 6.919310325589565
Min IR S/N = 2.3794887584698142e-05
Max IR S/N = 11.616791240326972
```

Listing 1: Example of the terminal output of the ETC. It is worth noting that the output contains an excessive number of digits.

The parameters used to generate both the two plots and the terminal output are listed on the signal-to-noise plot.
Running the exposure time calculator takes around 7 seconds, and is faster when smaller wavelength ranges are simulated.

The plots of the signal-to-noise and the spectrum in figure 13 are made without taking the moon into consideration. Figure 14 shows a comparison between the signal-to-noise ratios when using the new moon option and when using the full moon option. For this comparison, a relatively faint object is observed in order to make the S/N difference caused by the moon more pronounced. Both the spectrum and the signal-to-noise ratios for all three moon options can be seen in appendix A.2 for further comparison.



Exposure Time (single exposure):	1200.0 s
Number of exposures:	3 exposures
Slit Width:	1.00 arcsec
FWHM of PSF:	0.80 arcsec
Object Type:	powerlaw
Spectral Index:	0.10
AB Magnitude:	20.25
at wavelength:	650.0 A
Detector binning along dispersion:	1 pixels
Detector binning along slit:	1 pixels
Signal-to-Noise binning factor:	1 binned pixels/channel

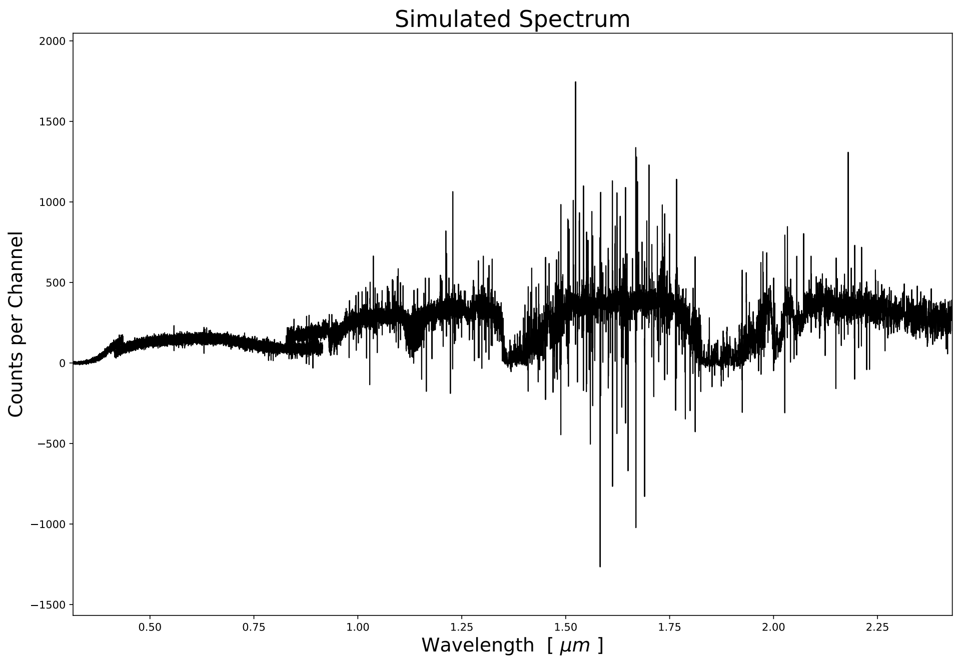
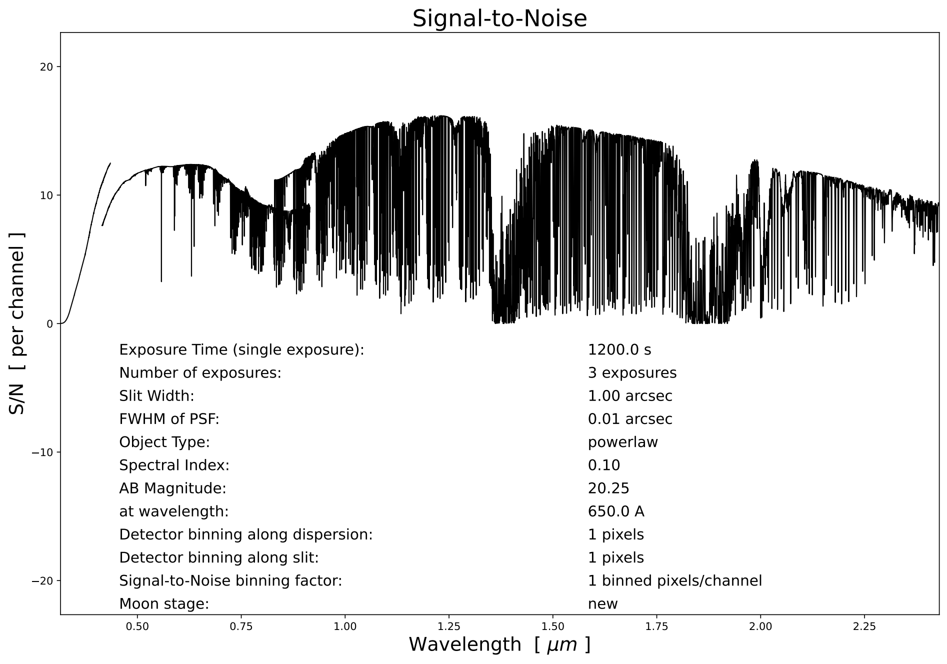
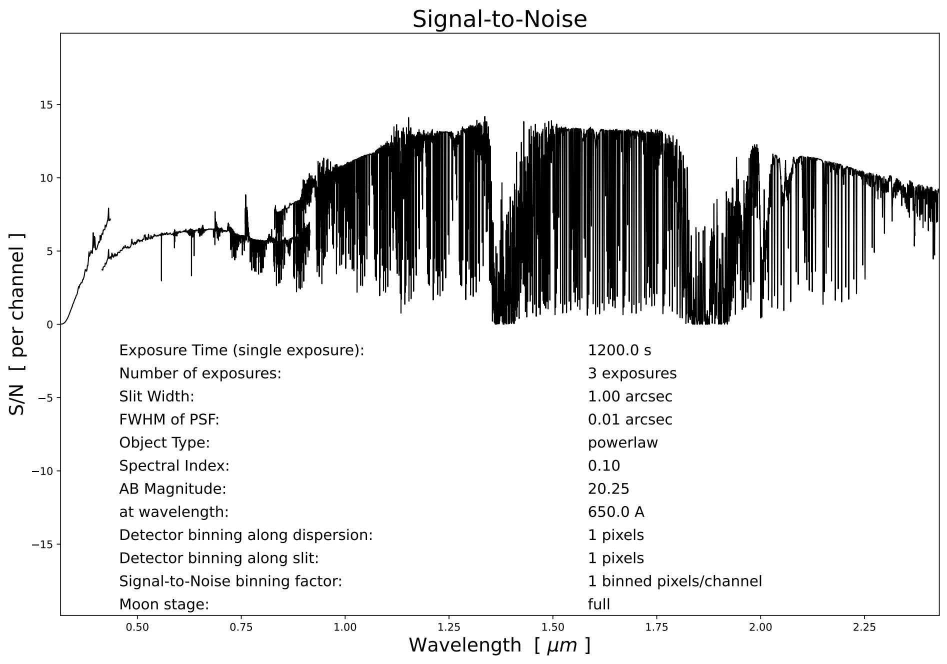


Figure 13: Example of final output images from the ETC. At the top is the signal-to-noise, while at the bottom is the spectrum. No moon spectrum has been considered for these results.



Exposure Time (single exposure):	1200.0 s
Number of exposures:	3 exposures
Slit Width:	1.00 arcsec
FWHM of PSF:	0.01 arcsec
Object Type:	powerlaw
Spectral Index:	0.10
AB Magnitude:	20.25
at wavelength:	650.0 A
Detector binning along dispersion:	1 pixels
Detector binning along slit:	1 pixels
Signal-to-Noise binning factor:	1 binned pixels/channel
Moon stage:	new



Exposure Time (single exposure):	1200.0 s
Number of exposures:	3 exposures
Slit Width:	1.00 arcsec
FWHM of PSF:	0.01 arcsec
Object Type:	powerlaw
Spectral Index:	0.10
AB Magnitude:	20.25
at wavelength:	650.0 A
Detector binning along dispersion:	1 pixels
Detector binning along slit:	1 pixels
Signal-to-Noise binning factor:	1 binned pixels/channel
Moon stage:	full

Figure 14: Example of final output images from the ETC. At the top is the signal-to-noise, while at the bottom is the spectrum. The full moon spectrum has been considered for these results.

4 Using the Exposure Time Calculator

This section covers the practical usage of the exposure time calculator. At this time the code can be obtained from [29] or from the GitHub link in appendix A.4.

All the variables are defined in the 'config.ini' file located in the 'data' folder. These variables should initially be read through and the values adjusted to fit the desired observation. The parameters are divided into groups to make the file more manageable, and the five groups are [OBSERVATION], [SPEC], [UV_DET], [VIS_DET], and [IR_DET]. All the parameters are listed below, with the three detector groups containing identical parameters and therefore only listed once.

[OBSERVATION]

- exposure_time = Exposure time in s.
- nr_of_exposures
- template = {P|B|T}{Index|T in K|FWHM in Å}. P is a power law, B is a Planck law and T is a template.
- template_file = Name of the file containing the spectrum template. Wavelengths in Å and flux in arbitrary units. Only relevant if 'template = T'.
- ab_mag = AB magnitude of the object.
- wavelength_of_ab_mag = Wavelength (in Å) where the AB magnitude is given.
- fwhm_seeing = The FWHM of the Gaussian PSF in arcsec.
- airmass
- moon_stage = {full|half|new|custom}. 'custom' lets the user provide a sky and moon spectrum. The spectra for the moon are taken while looking at the zenith, with the moon positioned at a 45-degree angle from the zenith.
- custom_sky_file = The sky spectrum to use if 'moon_stage = custom'. The moon should be included in this spectrum. Wavelengths in nm and flux in ph/s/m^2/µm/arcsec^2.
- custom_moon_file = The moon only spectrum to use if 'moon_stage = custom'. Wavelengths in nm and flux in ph/s/m^2/µm/arcsec^2.

- csv_output = {1|0} If 1, the output will be saved as a CSV file.

[SPEC]

- wl_lower_limit = Lower limit wavelength to plot in Å.
- wl_upper_limit = Upper limit wavelength to plot in Å.
- slit_width = The width of the slit in arcsec.
- slit_length = The length of the slit in arcsec.
- detector_binning_dispersion_direction
- detector_binning_spatial_direction
- post_detector_binning
- lambda_split_uvb_vis = Wavelength (µm) that separates the UV and VIS flux calculations. Below this split, UV files and calculations are used, while above it, VIS files and calculations are used.
- lambda_split_vis_ir = Wavelength (µm) that separates the VIS and IR flux calculations. Below this split, VIS files and calculations are used, while above it, IR files and calculations are used.
- ff_error = Flat field error as a fraction.
- tel_area = Telescope area in m^2.

[UV_DET]/[VIS_DET]/[IR_DET]

- dark = Dark current in e^-/pix/s.
- ron = Readout noise in e^-/pix.
- pix_length = Pixel length in arcsec.
- pix_width = Pixel width in arcsec.
- arm_min = Minimum wavelength for the arm in µm.
- arm_max = Maximum wavelength for the arm in µm.

The calculator is used by running the file 'NTE_ETC.py' in the terminal. The file has to be run from the directory where the file is located. Running the file from another directory by using the path name will result in an error message.
The exposure time calculator has three outputs:

- A median signal-to-noise value for each band used is returned in the terminal.
- A plot of the signal-to-noise value as a function of wavelength.
- A plot of the simulated spectrum, with flux as a function of wavelength.

The two plots will be saved in the same folder as 'NTE_ETC.py', with the names 'Signal-to-Noise.pdf' and 'Simulated-Spectrum.pdf'.

4.1 Note on using templates

It is possible to use a custom template option, by setting the 'template' variable to 'T', followed by the FWHM of the photometric bandpass used to normalize the spectrum.

The template file has to contain two columns. The first column is wavelength in ångström while the second column is the flux, F_λ, divided by the wavelength, in arbitrary units. The spectrum is then normalized to the magnitude that is defined in the config file.

4.2 Applying the ETC to other instruments

It would be relatively simple to apply this ETC to other instruments. Edit the config file above to closely fit the desired instrument, replace the SkyCalc files, and replace the efficiency files.

Replacing the SkyCalc files makes a relatively small contribution, but factors like the air mass will contribute to the final result.

Replacing the efficiency files is very important, and the used efficiency should be the total efficiency of the arm including the efficiency of the telescope and the detector. The names for the efficiency files are hardcoded into the 'read_uvb_qe', 'read_vis_qe', and 'read_ir_qe' functions, so they should be changed accordingly.

5 Simulation of spectra

For the creation of a data set, multiple tools already exist. The ones that will be used for this specific data set are ZEMAX and PyEchelle [6]. ZEMAX is a program used for designing and simulating optics. It uses ray tracing to simulate light as it travels through the spectrograph.

In this case, ZEMAX will be used to take 5 points on the input light beam and see where they end up on the CCD. The 5 points are positioned such that one is at the centre of the slit and the other four are located at the centre of each edge of the slit. From these points, it can be determined what amount of rotation, translation, scaling, and shear happens through the optical system. These transformations are affine transformations.

These transformations are one of the multiple inputs in PyEchelle. PyEchelle is a tool used to simulate echelle spectra and will be the main tool used to create the data set for NTE.

This simulated data will be a key component in preparing the pipeline for the NTE before the telescope is up and running.

5.1 Types of spectra

The basic spectra that are needed for the testing of a pipeline can all be created using PyEchelle. The spectra created with PyEchelle are 'clean' spectra without any noise. The noise is added in a separate step afterwards. The needed spectra are bias frames, dark frames, flat fields, lamp spectra, sky spectra, etalon, trace, and science spectra. Each of these spectra needs to be created for all three detectors.

The science frames are examples of observed spectra of objects. They will also contain additional sources, e.g. a sky spectrum. These spectra are the ones that should be reduced, using the other files as calibration for this. Examples of science spectra in this project are GRB spectra.

Examples of calibration frames are the bias frames. A bias frame is an image taken with a closed shutter and an exposure time as close to zero as possible. It will still contain a signal which varies from pixel to pixel. This signal will also be present in the science frame, and will therefore need to be subtracted. [30]

For science frames with long exposure times, the science frame will also contain thermal noise. This is corrected for using dark frames. Since the amount of thermal noise is dependent on the exposure time, the dark frame needs to have the same exposure time as the science frame, but will, like the bias frame, be an image taken with a closed shutter. Like the bias frame, the dark frame can now be subtracted from the science frame. The dark frame will also contain the noise from the bias frame, and therefore only one of them should be used. Since the thermal noise is primarily a problem with long exposure times, a bias frame will be used with the UV and VIS detectors, while a dark frame will be used with the IR detector. [30]

Additionally, there will also be made flat fields. These are images of white light so the flux in wavelength is relatively even. These are mostly used to get the location of the slits on the detector. The NTE uses a halogen lamp in combination with a filter to produce a relatively flat spectrum.

To reduce read noise and shot noise, there will usually be made a number of bias frames, dark frames, or flat fields that will be combined into a master bias, master dark, or master flat.

In spectroscopy, it is necessary to know where different wavelengths end up on the CCD. Since NTE covers from IR to UV wavelengths it is necessary to use a lamp that has lines in this entire wavelength interval. For the NTE a HgAr lamp is used to serve this purpose.

In addition to the HgAr lamp, an etalon spectrum can be used to calibrate the wavelengths. In some wavelength intervals, there are very few lines from the HgAr lamp, and in these intervals, it is helpful to have another source for wavelength calibration. The etalon spectra are a sinus curve in flux with an increasing period over the wavelength. This is generated by the NTE calibration unit.

Furthermore, a trace file is used to track the centre of the slit. The file is created using a pinhole.

For ground-based telescopes, the science frames also contain emission and absorption from the atmosphere which needs to be corrected for. Therefore, sky spectra as well as science spectra including the sky are necessary.

To create all these spectra, three different ways to provide the source data to PyEchelle, are used. A constant source is used for creating the field and the trace. Since noise is added separately in a later step, the initial spectrum has a constant flux as a function of wavelength. A constant source is used to produce the bias and dark frames, but they have an exposure time of 0 to produce an empty image.

Another option is using a CSV file as the source. The CSV file will contain a number of wavelengths with the related intensity, and PyEchelle then interpolates over this. This option is used for the etalon spectrum, the sky frame, and the science frame. All these spectra vary with wavelength, so as long as enough wavelengths are provided the interpolation is a good option. The sky source is taken from the ESO SkyCalc, the etalon source was calculated by Anton Norup Sørensen. and the science source was provided by Johan Fynbo.

The final way used to provide a source is a line list. Like with the CSV option, a list of wavelengths and the corresponding intensities are provided but in this case PyEchelle will not interpolate. Instead, all wavelengths that are not provided in the list will have intensity 0. This is exactly what is needed to create the lamp spectrum. The line list for the HgAr lamp is taken from the NIST database, using only the neutral mercury and argon lines.

5.2 Affine transformations

The output from ZEMAX gives the placement of the 5 points after having been through the spectrograph. These 5 points represent points on the slit. These points are not necessarily still placed in a square and might have a different size, rotation, shear, and centre. The transformations needed to move these points back to their original positions are the input needed for PyEchelle. This input is a list of x-translation, y-translation, x-scale, y-scale, rotation, and shear for each wavelength.

For each wavelength, there are 5 points and an example of this can be seen in figure 15. The numbering of the points can be seen in figure 15. The top left image is the output from ZEMAX while the bottom image is the original positions they should be transformed back into. This is done using affine transformations, which can be expressed using a single matrix.

$$\begin{bmatrix} s_x\cos(\theta) & -s_y\sin(\theta+m) & t_x \\ s_x\sin(\theta) & s_y\cos(\theta+m) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{26}$$

where x_i and y_i is the x and y value of point i.

The translation in the x and y direction is the simplest of these transformations.

The centre point, p0, is moved to the origin, (0,0), meaning

$$t_x = x_0 \tag{27}$$

$$t_y = y_0 \tag{28}$$

Next is rotation. The value of the rotation is the angle that will make the line connecting points 1 and 3 parallel to the x-axis and thus have the same y-value. The rotation can therefore be found using trigonometry.

$$\theta = \arctan\left(\frac{x_3}{y_3}\right) \tag{29}$$

The shear can be determined by ensuring that the line connecting points 2 and 4 is parallel to the y-axis after the transformation, resulting in both points having the same x-value. This can be solved using matrices. Initially, there are two matrix equations. Here x_{n_i} and y_{n_i} are the new x and y-values for point i after the transformation, and m is the shear.

$$\begin{bmatrix} x_{n_2} \\ y_{n_2} \end{bmatrix} = \begin{bmatrix} 1 & m \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \tag{30}$$

$$\begin{bmatrix} x_{n_4} \\ y_{n_4} \end{bmatrix} = \begin{bmatrix} 1 & m \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_4 \\ y_4 \end{bmatrix} \tag{31}$$

Because point 2 and point 4 should have the same x value after the transformation, it can be applied that $x_{n_2} = x_{n_4}$.

$$1x_2 + my_2 = 1x_4 + my_4 \tag{32}$$

$$m = \frac{x_2 - x_4}{y_4 - y_2} \tag{33}$$

Lastly, scaling is needed. In this case, it is the distance between opposite points.

$$s_x = x_3 - x_1 \tag{34}$$

$$s_y = y_4 - y_2 \tag{35}$$

Figure 15 shows an example of a full set of transformations done using this approach. The bottom image that shows the result after the transformations is extremely close to the original positions of the points before the distortion by ZEMAX.

All of these transformations are written into an HDF file which can be used in PyEchelle. Additionally, this HDF file also needs to contain the point spread functions (PSFs) for all orders at multiple wavelengths. For the initial testing, a Gaussian PSF and some arbitrary wavelengths were used. Later, PSFs generated by Zemax were provided and used instead.

5.3 PyEchelle

This section describes the functions used for this project.

When using PyEchelle the first step is to define the spectrograph. This is done with the line 'sim = Simulator(ZEMAX('NTE'))'. There are multiple options for this, but ZEMAX is the only one that takes an HDF file as input. ZEMAX is a class with functions to read the HDF file. The input 'NTE' is the name of the HDF file containing the affine transformations. This HDF file, 'NTE.hdf', needs to be placed inside the folder called 'models'.

The 'Simulator' class contains many (optional) functions used to define exactly what PyEchelle needs to do. This includes defining which CCD to use, defining which fiber to use, which order to use, giving a source file, giving an efficiency file, setting an exposure time, selecting whether cuda is used, etc. The functions used for creating the data are listed below.

```python
sim = Simulator(ZEMAX('NTE'))
sim.set_ccd(ccd)
sim.set_fibers(fiber)
sim.set_sources(source)
sim.set_efficiency(efficiency)
sim.set_exposure_time(exposure_time)
sim.set_output(output_dir)
sim.set_cuda(True)
```

Listing 2: Example setup for the PyEchelle Simulator class

After setting all the desired optional functions, PyEchelle can be run with 'sim.run()'. Initially, a matrix is created which will end up being the content of the output file. The matrix has the same dimensions as the selected CCD and is initialized with 0. Next, the slit function is found using info from the HDF file. For each fiber, it is defined in the HDF file as whether the slit is rectangular or circular.

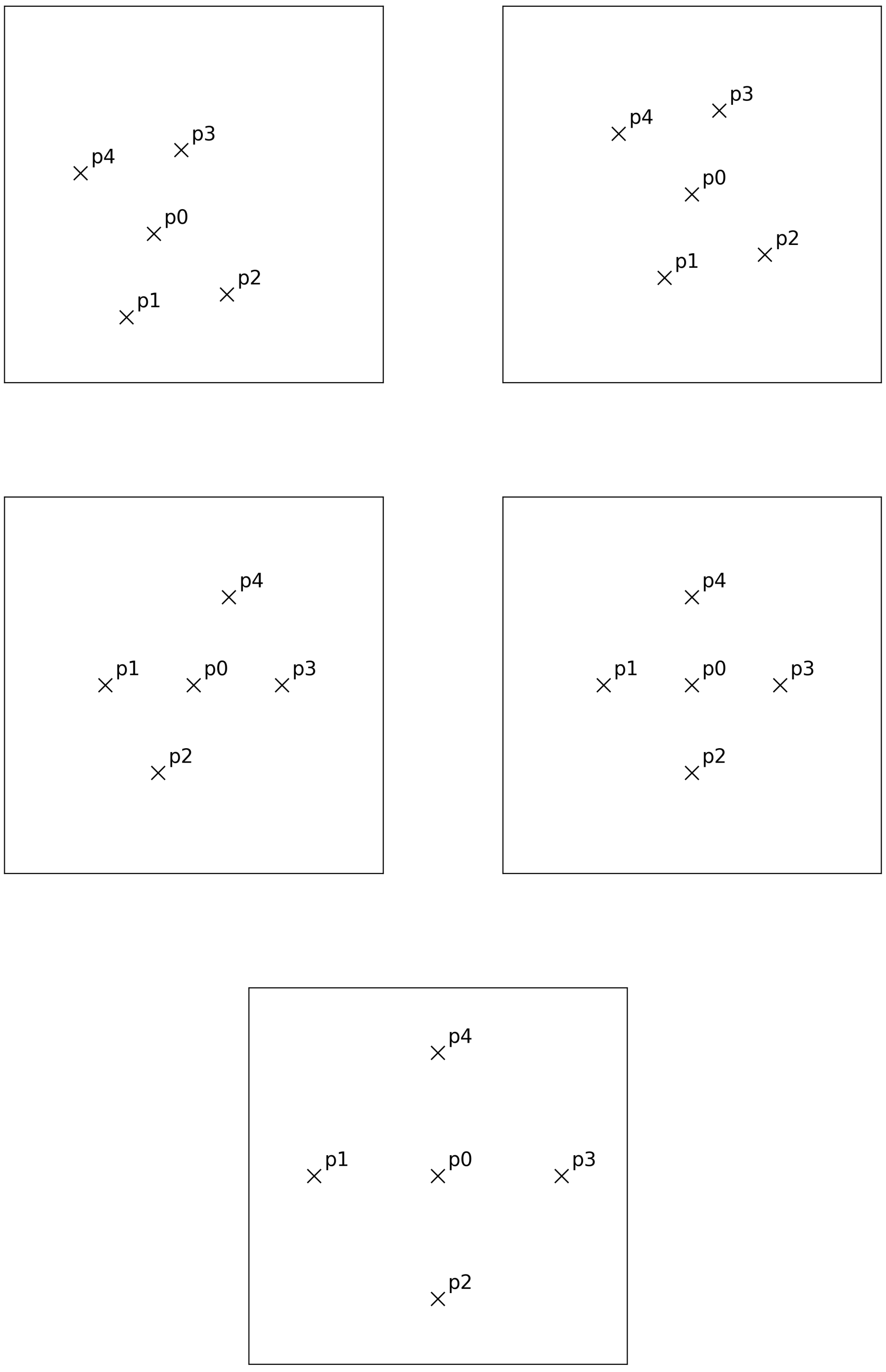


Figure 15: Example of a full set of transformations done by the created code. Top left: distorted input points. Top right: points after translation. Middle left: points after rotation. Middle right: points after shear. Bottom: points after scaling.

Next, the efficiency will usually be added. In the case where an efficiency is specified using 'sim.set_efficiency(efficiency)', both this input efficiency and the grating efficiency from the HDF file are multiplied together and then applied. In the case where the efficiency is not specified, PyEchelle will still add the grating efficiency, but for testing purposes, this was removed. Therefore if the efficiency is not set, no efficiency will be applied.
The grating efficiency is found using the equations

$$\beta = \arcsin\left(-\sin(\alpha) + \frac{n\lambda}{d}\right) \tag{36}$$

$$x = n\pi\left(\frac{\cos(\alpha)}{\cos(\alpha-\phi)}\right)\left(\cos(\phi) - \frac{\sin(\phi)}{\tan((\alpha+\beta)/2)}\right) \tag{37}$$

$$I = k\left(\frac{\sin(x)}{x}\right)^2 \tag{38}$$

where α is the angle of incidence of the incoming light, ϕ is the blaze angle, n is the order, d is the grating size and λ is the wavelength. [31] The grating density, which is the inverse of the grating size, and the blaze angle are found in the HDF file. α is also set to the blaze angle.

The fluxes are produced for one order at a time. The wavelength interval for the order is found in the HDF file by taking the first and last index of the wavelength list. Then a new wavelength array is created over the same wavelength interval but with 100.000 points.

If the radial velocity of the target is not actively defined, as in this case, it is set to 0, and the spectral density is pulled directly from the provided source. In most cases, both a wavelength array and a matching flux array are returned, and in these cases, the previously created wavelength array is replaced with this new one. But in the case of a constant source, only a flux array is returned and the previously created wavelength array will be used going forward.
The effective density can then be calculated by multiplying the spectral density with the efficiency. If the unit of the provided source is already in photons per second per wavelength interval, the flux is equal to the effective density. If the unit of the provided source is instead µW/µm it needs to be converted to ph/s/$\delta\lambda$ first.
The number of photons at each wavelength interval is then found by multiplying the flux with the exposure time, and the total number of photons for the order is found

by summing over this array.

Next, a new wavelength array with 10.000 steps is created over the wavelength interval of the order. This wavelength array is gonna be used with the affine transformations from the HDF file.
PyEchelle gets the values of the separate transformations and a corresponding wavelength array from the HDF file and combines them into transformation matrices like the one shown in equation 26. This combination is done using the equations for each element shown in equation 26. 6 arrays are then created, one for each element in the transformation matrix. Each of these arrays contains the value of a specific element of the transformation matrix for all wavelengths. To get an array of transformation matrices that match the array of wavelengths that was just created, an interpolation is made over each of these 6 arrays. This results in 6 arrays of length 10.000 that combined form 10.000 transformation matrices matching the wavelength array.
Next, 6 new arrays are created to store the gradients, one for each of the previous arrays, where each element in the new array y is related to the old array x as $y_i = x_{i+1} - x_i$. This means that the new arrays are one shorter than the old arrays. This is solved by duplicating the last index.

Then, using the method described in section 5.3.1, two arrays are created that make it possible to choose a random offset using the distribution from the PSFs. This is done for each of the provided PSFs. The wavelengths for each of the PSFs are collected in an array.
For the spectrum distribution the same method from section 5.3.1 is used to create two arrays for choosing a random wavelength with the distribution of the spectrum distribution.

The photons are created one at a time using a for loop over the total number of photons in the order. For each photon, a random index is then chosen using the spectrum distribution generated by 5.3.1. This gives the wavelength of the photon. To find the matching transformation matrix, the wavelength array for the transformations is compared to the wavelength of the photon, and the transformation wavelength just below the photon wavelength is selected. In addition, the difference between the transformation wavelength and the photon wavelength is found. The transformation wavelength selected will then determine the transformation matrix for the photon. An interpolation is made with the wavelength difference to find the specific transformation matrix to use for the photon.

Next, the position where the photon hits the slit is found by choosing two random numbers from a uniform distribution, and the place where the photon hits the CCD is found by applying the transformation matrix to the spot where the photon hits the slit.
Lastly, the PSF is applied. The PSFs are for specific wavelengths, and the one for the nearest wavelength is chosen. Then a random offset for the photon is chosen using the distribution from the PSF (generated as explained in 5.3.1), and the offset is applied to the position of the photon on the CCD.
Now the photon can be placed in the empty CCD matrix at the position that was just found. This index of the matrix is then increased by 1. When all the photons have been placed this way, the CCD matrix is saved as a fits file.

5.3.1 The Alias Method

Choosing random numbers based on arbitrary distributions is not straightforward. The Alias Method provides a solution for generating random numbers from such distributions. It simplifies the process by generating random numbers from a uniform distribution, which is more efficient. Here's how it works: First, roll an n-sided die, where n is the length of the probability distribution array. Next, flip a biased coin. Both of these steps can be accomplished by using random numbers from uniform distributions. In the end, the Alias Method enables the generation of a random number from the desired distribution. [32]

Start with a normalized probability array meaning the sum of the array is 1. For the example in figure 16 the used array is

$$o = [Blue, Pink, Grey, Green] \tag{39}$$

$$p = [1/2, 1/3, 1/12, 1/12] \tag{40}$$

where o is the options and p is probabilities. Then the array is scaled so that the mean is 1. If the initial array is normalized, this can be done by multiplying each element by the length of the array. In this example, where the array has a length of 4, multiplying each element by 4 gives. [32]

$$p = [2, 4/3, 1/3, 1/3] \tag{41}$$

The options are then rearranged so all columns have a probability of 1. This is shown in the bottom row of figure 16. To keep track of the probabilities, two arrays are now needed instead of one. The first array keeps track of the probability in the column of the original colour. The other array keeps track of where the rest of the probability is taken from. Since it will always be possible to make the probabilities of all columns 1 using only the original colour and one other colour, these two arrays are enough. For continuity the probability array is called q and the index array is called j. [32]

Converting p into these two arrays gives

$$q = \left[2, \frac{4}{3}, \frac{1}{3}, \frac{1}{3}\right] \tag{42}$$

$$j = [-, -, -, -] \tag{43}$$

After the first move on the bottom left of figure 16 the arrays become

$$q = \left[\frac{4}{3}, \frac{4}{3}, \frac{1}{3}, \frac{1}{3}\right] \tag{44}$$

$$j = [-, -, -, 0] \tag{45}$$

After the second move shown in the bottom middle of figure 16

$$q = [2/3, 4/3, 1/3, 1/3] \tag{46}$$

$$j = [-, -, 0, 0] \tag{47}$$

And after the last move shown in the bottom right of figure 16

$$q = [2/3, 1, 1/3, 1/3] \tag{48}$$

$$j = [1, -, 0, 0] \tag{49}$$

A random output from this distribution can be found by first rolling a die with the same number of sides as the number of columns to choose one of the four columns. In this case that would be a 4 sided die. The die roll will naturally result in a number from 1 to 4, and the corresponding column will be chosen. If the die roll results in column 1 (corresponding to index 0 in the arrays), there are now only two options for the output. Either the original value of the first column which is $o[0] = blue$, or the value that has been moved into the first column which can be found in $j[0] = 1$,

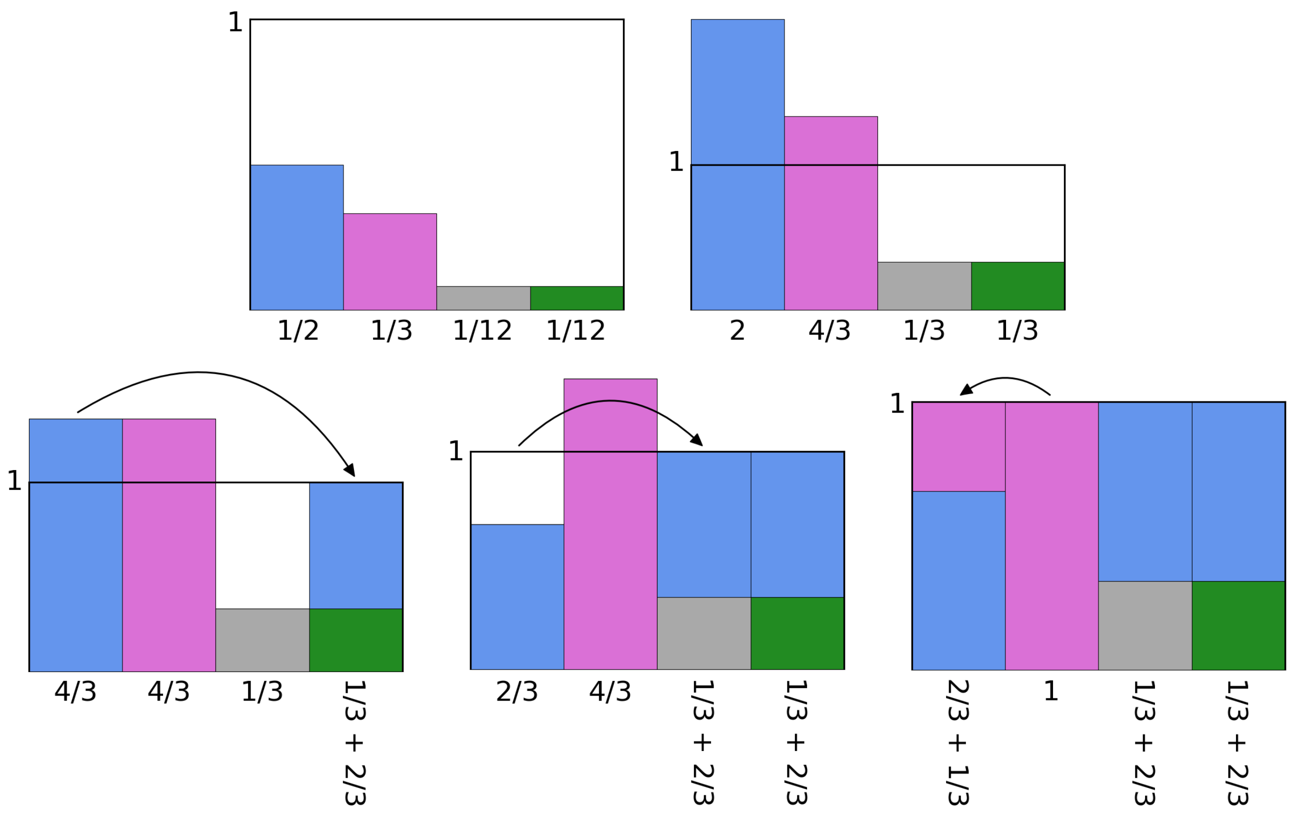


Figure 16: The Alias Method. Here, it is illustrated how the Alias Methods first normalize the probabilities and then rearrange them so that all options have equal probabilities. Figure based on [32].

and therefore the original value from column 1 which is $o[1] = pink$. [32]

To choose between blue and pink, a biased coin is flipped. In the case where column 1 has been chosen from the die roll, the biased coin will have $q[0] = 2/3$ chance of heads (blue) and $1 - 2/3 = 1/3$ chance of tails (pink). [32]

The exact way it is done in PyEchelle is very similar and is illustrated in figure 17. Again, start with a normalized probability array. Initial arrays for the probability and the index are then created. The probability array is called q and the index array is called j. The probability array is the input array multiplied by the length while the j array is an array of the same length filled with 0.

$$q = [2, 4/3, 1/3, 1/3] \tag{50}$$

$$j = [0, 0, 0, 0] \tag{51}$$

Then two arrays are created: one that contains the indices of entries with a probability lower than one, and another that contains the indices of entries with a probability above or equal to 1.

$$s = [2, 3] \tag{52}$$
$$l = [0, 1] \tag{53}$$

Now, the last elements of s and l are used (i_s and i_l). Probability is then moved from $q[i_l]$ to $q[i_s]$ so $q[i_s] = 1$, and j is updated. In this example, probability will be moved from $q[1]$ to $q[3]$ as the first step.

$$q[i_l] = q[i_l] - (1 - q[i_s]) \tag{54}$$
$$j[i_s] = i_l \tag{55}$$

i_s is then removed from the array s while i_l is kept in l if $q[i_l] \geq 1$ or moved to s if $q[i_l] < 1$. This is repeated until s is empty. The entire process for this example is seen in figure 17. This results in

$$q = [1, 2/3, 1/3, 1/3] \tag{56}$$
$$j = [0, 0, 0, 1] \tag{57}$$

The random output is found the same way as described earlier. The die is replaced by 'math.floor(random.random()*len(q))', where 'random.random()' gives a random number from a uniform distribution between [0,1). 'random.random()*len(q)' will, in this case, be 'random.random()*4' and therefore give a random number from a uniform distribution between [0,4). 'math.floor()' rounds down the number to give either 0, 1, 2 or 3 with equal probability. 0, 1, 2, and 3 correspond to the four columns. 'math.floor(random.random()*len(q))' = 3 would choose column 4.

The biased coin can also be replaced with a random number from a uniform distribution between [0,1). For the example with column 4, if the random value is less than $q[3] = 1/3$ then the result is the original value of the column which was $o[3] = green$. If the random number is above $q[3] = 1/3$ then the result is the value that has been moved into the column which can be found from $j[3] = 1$ was $o[1] = pink$.

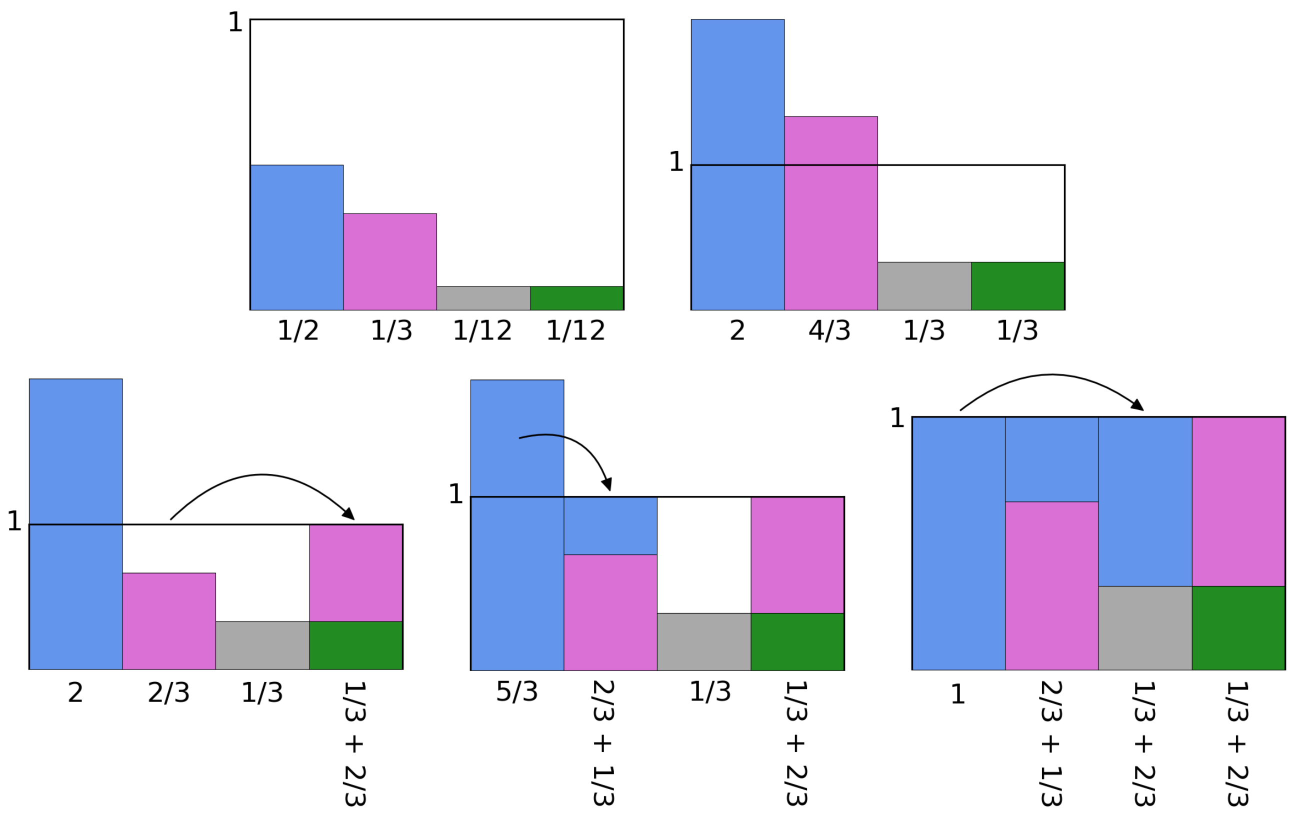


Figure 17: The Alias Method used in Pyechelle. Here, it is illustrated how the slightly altered Alias Methods first normalize the probabilities and then rearrange them so that all options have equal probabilities.

5.4 Bias and noise frames

For the initial testing, Gaussian noise is used, which can directly be added to the files using an option in PyEchelle. This however does not take cold, hot, and dead pixels into account at all. To accomplish this, bias and noise frames made specifically for NTE are needed.
This can be done by getting some pure noise images from the real detectors that will be used for the NTE. At the moment, 10 noise frames have been provided for the IR detector, while none have been provided for the UV and VIS detectors. To get the bias value for each pixel the median is taken over all 10 frames for each of the pixels. To generate the noise, the standard deviation is taken in each pixel over the 10 frames. Then for each pixel, the noise is generated using a normal distribution with this standard deviation.

It should be noted that 10 noise frames are not a lot to make statistics on, but this is all that has been provided so far by the company that is producing the detector.

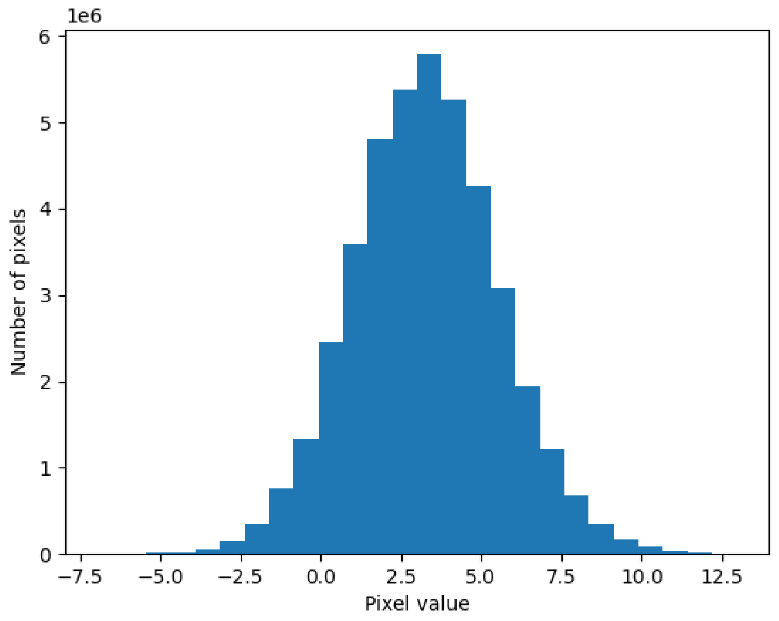


Figure 18: Histogram of pixel values in a single noise frame. This illustrates a close to Gaussian distribution of pixel values in the noise frame.

Furthermore, it is probably not a great assumption that all pixel noise is based on a normal distribution.

To get an idea of the overall distribution of the readout noise (RON) a histogram can be made of the values of all pixels in all 10 frames. Figure 18 shows this histogram. Based on this figure, it appears that a normal distribution would be a reasonable approximation. However, even though the RON for the entire frame seems to follow a normal distribution, the same is not necessarily true for the individual pixels.

As noise frames have only been provided for the IR detector, the VIS and UV detectors instead use a normal distribution based on the RON level provided for the detectors, until the actual noise frames are provided. For the bias level, a placeholder of 1000 is added to all data files.

5.5 Flat fields

Not all pixels on the detectors will have the same sensitivity. This can be accounted for by using a flat field, which is an image where the entire detector is evenly illuminated.

The flat field can then be normalized, and when generating simulated data using PyEchelle, the normalized flat field can be multiplied with the data to simulate pixel-to-pixel response non-uniformity.

Like with the bias frames, a flat field has only been provided for the IR detector. As a result, the flat fields for the UV and VIS detectors consist of placeholder matrices filled with ones.

5.6 Intensity of HgAr lamp

Since the intensity of the HgAr lamp can already be measured at the moment of writing, it is possible to scale the NIST [33] line data for Hg and Ar to match the measurements. Only neutral mercury and argon are used since the measured lamp does not contain lines from ions.

The intensity on the lines for the lamp can be adjusted with two parameters. a is the number all the intensities are multiplied by while b is the relation between Hg and Ar.

$$I_{measured} = a \cdot (I_{Hg,NIST} + b \cdot I_{Ar,NIST}) \tag{58}$$

The measurement of the lamp is made with larger wavelength bins than the spectra created by PyEchelle. It is, therefore, necessary to not just take the intensity of the line in the spectrum, but instead, take the intensity of the entire wavelength interval covered by the measurement. Since the intensities were measured by a photodiode behind a narrow band filter, the wavelength interval is given by the narrow band filter FWHM, and the intensities of all wavelengths in this interval are summed up to make it as simple as possible.

The intensities are measured at four wavelengths, 366 nm, 405 nm, 694 nm and 853 nm. The two shortest wavelengths are primarily mercury lines while the two longest wavelengths are argon lines. Since there are both measured mercury and argon lines one of each is chosen to find a and b. The lines used are 366 nm and 853 nm.

To find a and b eq. 19 is used, together with the measured intensities which can be found in table 5. From the line list produced using NIST, all lines inside each wavelength interval can be found, and the intensities summed up. For 366 nm (362.5 nm to 369.5 nm) $I_{Hg,NIST} = 14500$ and $I_{Ar,NIST} = 0$, and for 853 nm (846.75 nm to

Wavelength [nm]	FWHM [nm]	I_{lamp} [ph/s]	$I_{PyEchelle}$ [ph/s]	$I_{lamp}/I_{PyEchelle}$
366	7.0	3.3e9	3.3e9	1.0000
405	14.0	4.3e9	3.0e9	1.4439
694	11.0	1.2e8	6.6e8	0.18150
853	12.5	6.5e8	6.5e8	0.99997

Table 5: Results from comparing measured flux of the lamp vs. the measured flux from the simulation. I_{lamp} is the measured flux of the lamp. $I_{PyEchelle}$ is the calculated flux simulated by PyEchelle.

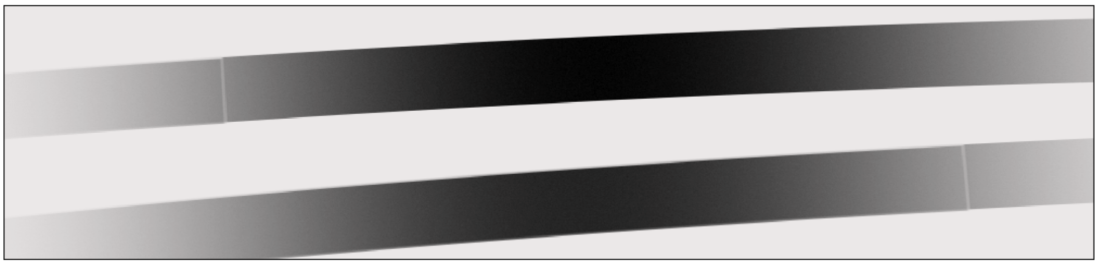


Figure 19: Initial testing on UV arm. It is evident in the top-left and bottom-right portions of the image that there are vertical lines. These lines indicate the breaking points where Pyechelle no longer applies the point spread function (PSF). As a result, the photons in these areas are shifted compared to the regions where the PSFs are applied.

859.25 nm) $I_{Hg,NIST} = 33$ and $I_{Ar,NIST} = 15000$.

$$3.3e9 = a \cdot (14500 + b \cdot 0) \tag{59}$$

$$6.5e8 = a \cdot (33 + b \cdot 15000) \tag{60}$$

$$a = 227586 \tag{61}$$

$$b = 0.188204 \tag{62}$$

Using these values for a and b results in the intensities ($I_{PyEchelle}$) in table 5.

The results are not perfect, but they are within an order of magnitude and this is acceptable for now.

5.7 Challenges

An example of some initial testing on PyEchelle can be seen in figure 19. In this image, two distinct lines are visible. Due to the wavelength range of the orders overlapping, the two breaks occur at the same wavelength.

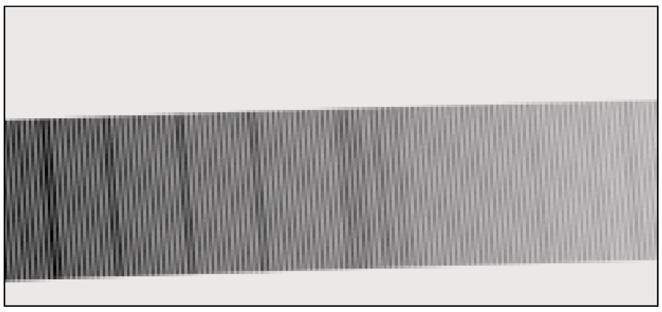


Figure 20: Wavy pattern on sky spectrum on the UV arm. This is a result of low resolution of the input spectrum.

A lot of testing was done to figure out what caused this problem, and it was determined that the issue was related to the application of the PSFs. In initial testing, Gaussian PSFs were used as temporary substitutes for the actual ones, until more realistic PSFs were obtained. However, PyEchelle requires more PSFs for smaller wavelength gaps. If a PSF is not provided sufficiently close in wavelength, then PyEchelle will not use any PSF. Consequently, the problem was resolved by providing a greater number of PSFs at various wavelengths.

While testing the code, another problem was identified in the simulation of the sky spectrum. An image of a sky spectrum in the UV arm is shown in figure 20 and it is clear that the spectrum contains a wavy pattern. This is especially noticeable in the continuum part of the spectrum between the emission lines.

At first, this pattern was assumed to be caused by the PSFs, but even with a new set of PSFs, the problem was still there.
It was tested whether increasing the resolution of the input spectrum would make a difference, and the result can be seen in figure 21. This resolved the problem and a likely cause for this is that the low resolution of the input spectrum was interpreted as a line list by PyEchelle.

5.8 Point source option in PyEchelle

The original PyEchelle code does not have the option to create a spectrum where the target is a point source, which was needed in order to create the spectrum for the science frames. This additional option takes the seeing in arcseconds and the position of the object inside the slit, with the position given as a number in the range 0 to 1 in both directions, as inputs.

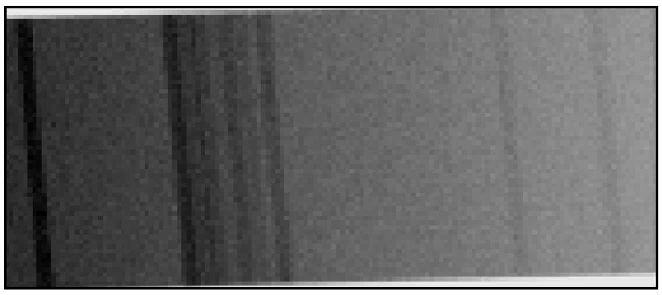


Figure 21: Example of the UV arm sky spectrum after an increase of spectrum resolution.

PyEchelle creates the spectra by tracing the photons individually. The photons are placed on the slit by generating uniform random points in the range 0 to 1, corresponding to the position on the slit.
The option of making the target a point source could then easily be included by changing this uniform distribution to a normal distribution and then removing any photons with a position outside the interval [0,1], to take into account that the whole object might not be inside the slit. The mean value used for this normal distribution is the position input in the point source function, while the standard deviation is found using the seeing. Since the slit is rectangular and the standard deviation depends on the slit width, different standard deviations are used for the two directions. Using that the FWHM of the normal distribution should be equal to the seeing (or in this case, the seeing divided by the slit width), the standard deviation can be found using

$$\sigma = \frac{s}{w}\frac{1}{\sqrt{8\ln(2)}} \tag{63}$$

where σ is the standard deviation, s is the seeing and w is the slit width. The last fraction is the relation between the FWHM and the standard deviation of a normal distribution. This relationship between the seeing and the standard deviation is derived using the equation describing a normal distribution

$$f(x) = \exp\left(-\frac{(x-b)^2}{2\sigma^2}\right) \tag{64}$$

where b is the mean. This Gauss has a max intensity of 1 and the intensity at FWHM is therefore 0.5. At the two points with an intensity of 0.5, it must therefore apply that $(x-b)^2 = (\text{FWHM}/2)^2$.

	IR	VIS	UV
Scale [arcsec/pix]	0.465	0.27	0.345

Table 6: Pixel scales are calculated by Michael I. Andersen.

Seeing ["]	IR FWHM ["]	VIS FWHM ["]	UV FWHM ["]
1.8	2.012	1.830	1.704
2.5	2.721	2.523	2.335
3.2	3.438	3.219	2.989

Table 7: Different seeings and the corresponding FWHM for each arm in the spectrograph.

$$\frac{1}{2} = \exp\left(-\frac{(\mathrm{FWHM}/2)^2}{2\sigma^2}\right) \tag{65}$$

$$2\ln(2)\sigma^2 = (\mathrm{FWHM}/2)^2 \tag{66}$$

$$\mathrm{FWHM} = \sigma\sqrt{8\ln(2)} \tag{67}$$

Figure 22 shows a section of a spectrum where the point source option was used with multiple different positions and seeing as input.

To test whether this new point source option acts as expected, the FWHM of the orders is estimated and compared to the seeing which was used as input. The expectation is that the seeing in arcsec is approximately equal to the FWHM in arcsec.

The FWHM of the point source is found using the information from the HDF file. For each wavelength, a corresponding position and angle can be found in the file, and from this position and angle, the intensity in multiple points along the slit can be found. A Gauss distribution can then be fitted to these intensities and the FWHM can be found. This is done by subtracting half of the max value and finding the two roots. The FWHM is the difference between roots multiplied by arcsec/pix (see table 6) since the roots are in pix and the seeing is in arcsec. This is done at multiple wavelengths and the mean is taken to get a more representative result. The results can be found in table 7.

Initially, there seemed to be a problem with the point source option when increasing the exposure time. When the pixels reached an intensity of 65536 it would stop

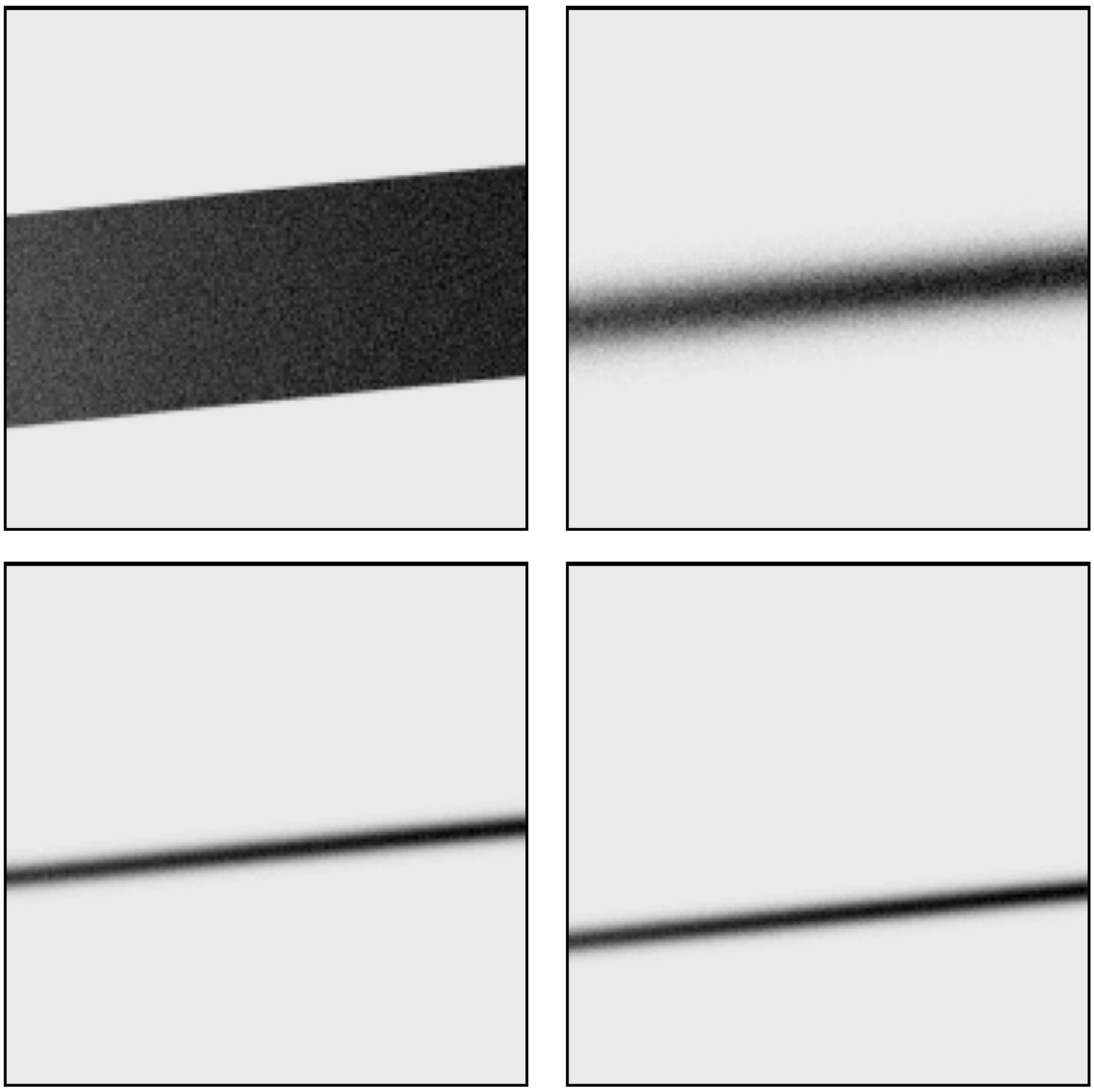


Figure 22: A small area of a science frame for grb25 in the UV filter with exposure time 10. Top left: full slit. Top right: point source with seeing 5 and position [0.5,0.5]. Bottom left: point source with seeing 2 and position [0.5,0.5]. Bottom right: point source with seeing 2 and position [0.5,0.2].

increasing, making the target appear constant across the slit instead of the expected Gaussian distribution. This problem is shown in figure 23. Since the data files will need a much higher pixel value, it was necessary to solve this problem. Fortunately, it turned out to just be a pre-set value in PyEchelle that could easily be changed.

5.9 Cosmic rays

As the main reason for simulating NTE data is to make sure the pipeline is working as expected, it is necessary to include cosmic rays in the simulated spectra. The

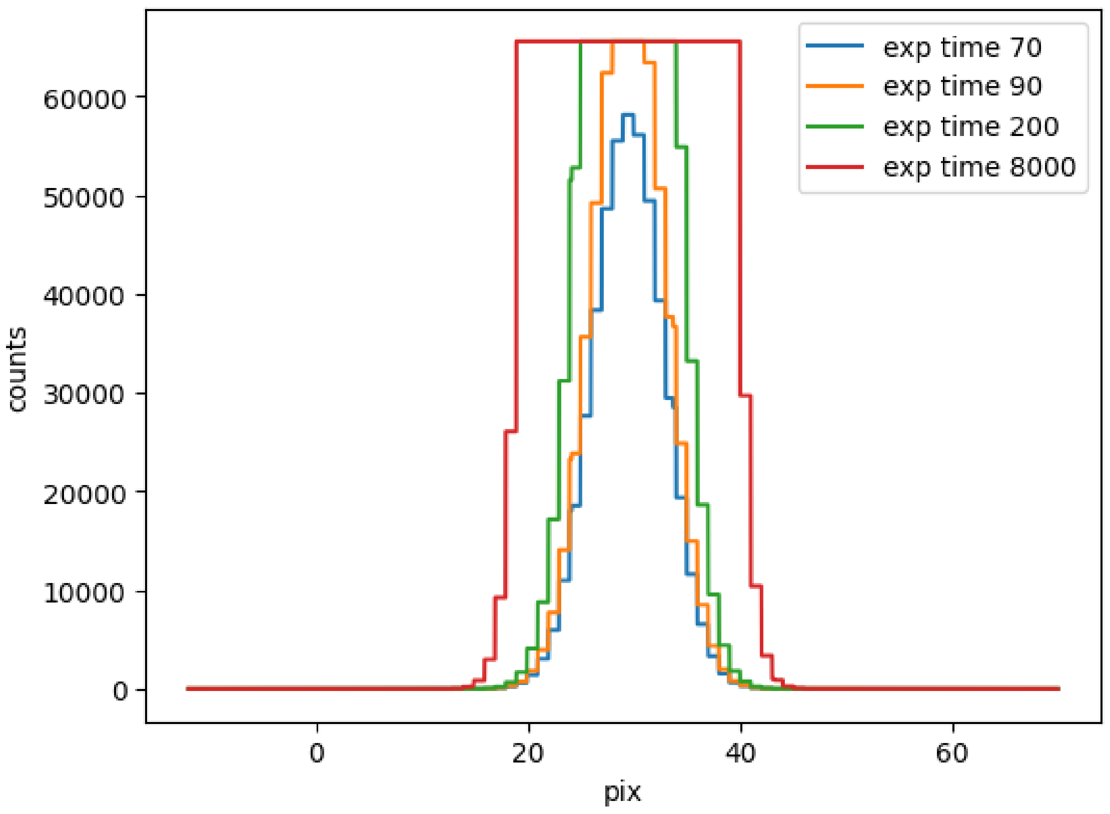


Figure 23: The intensity across the slit for a point source with increasing exposure times. For low exposure times, the shape closely resembles a Gaussian. However, for higher exposure times, the top gets cut off.

real spectra will contain cosmic rays, and it is therefore important that the pipeline is able to remove them.

The purpose of the Python package Pyxel [34] is to be an easy-to-use framework for simulating various detector effects. These detector effects are outside the scope of this project, but Pyxel also has a function for simulating cosmic rays, in addition to the detector effects, which can be used here. Pyxel will make a 3D simulation of the cosmic ray trajectories to figure out what pixels will be illuminated by the cosmic rays.

Since none of the other detector effects is wanted, and some of the detector effects must be included in order for Pyxel to run, an empty image will be used as the input. This is done since cosmic rays will have a very high intensity on the detector compared to the rest of the features, and a simple threshold can therefore be used to separate the cosmic rays from the rest of the image. Everything that is not cosmic rays can then easily be removed from the output image from Pyxel, by

Figure 24: Example of included cosmic rays on an image with a long exposure time for the UV arm.

setting everything below this threshold to 0. This results in an image containing only cosmic rays. Since the maximum possible value of the pixels for each detector is not known, the value of the cosmic rays is set to the max value of the spectrum the cosmic rays are going to be applied to.

A problem with Pyxel is that the cosmic ray function is developed for space-based telescopes, and the cosmic rays generated by Pyxel will therefore not be realistic for a ground-based telescope like NOT. However, since the main purpose of applying cosmic rays to the images is to make sure that the pipeline is able to remove them, it is not highly important that the cosmic rays are realistic. Figure 24 shows an example with a long exposure time. This is not a realistic image of what a spectrum with cosmic rays is going to look like, but it is sufficient for testing purposes.

5.10 Results

5.10.1 Schematic files

To test the pipeline, some specific frames are needed. The first step in creating these frames is creating a schematic.

Figures 25 and 26 show examples of the schematic for the infrared arm. Since the noise and bias data has only been provided for the IR detector, this detector is being used for examples since it is the most realistic.

It is noticeable that the fainter areas have high fluctuations from pixel to pixel. This is because the exposure time is purposely set very low so it will be easier to see the noise in the next section.

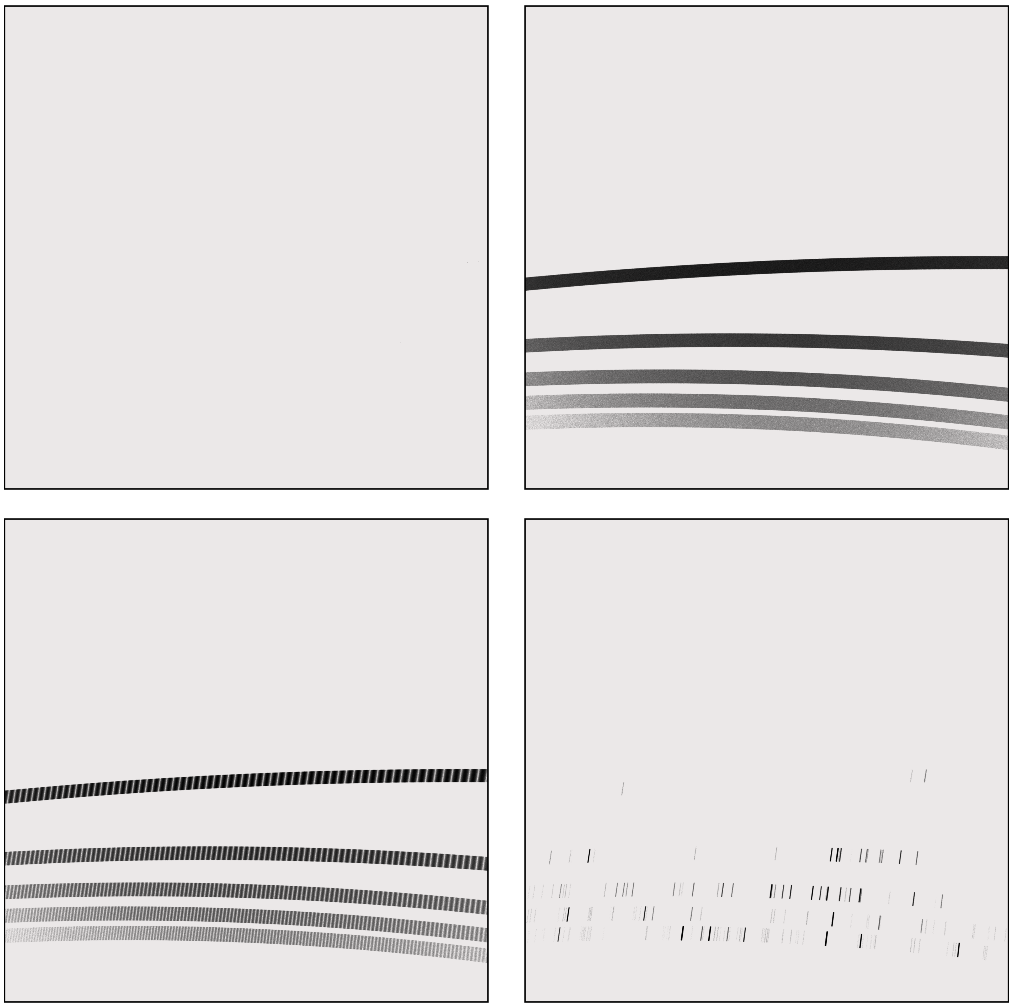


Figure 25: Examples of generated schematics for dark, flat, etalon, and HgAr calibration frames are shown. In this context, 'schematics' refers to images without bias, noise, or pixel sensitivity. Top left: dark. Top right: flat. Bottom left: etalon. Bottom right: HgAr. All plots here are on a log scale.

5.10.2 Completing the files with bias and noise

Based on the schematics shown in figures 25 and 26, the completed files can be made by first adding noise, and then multiplying with the normalized flat field. The

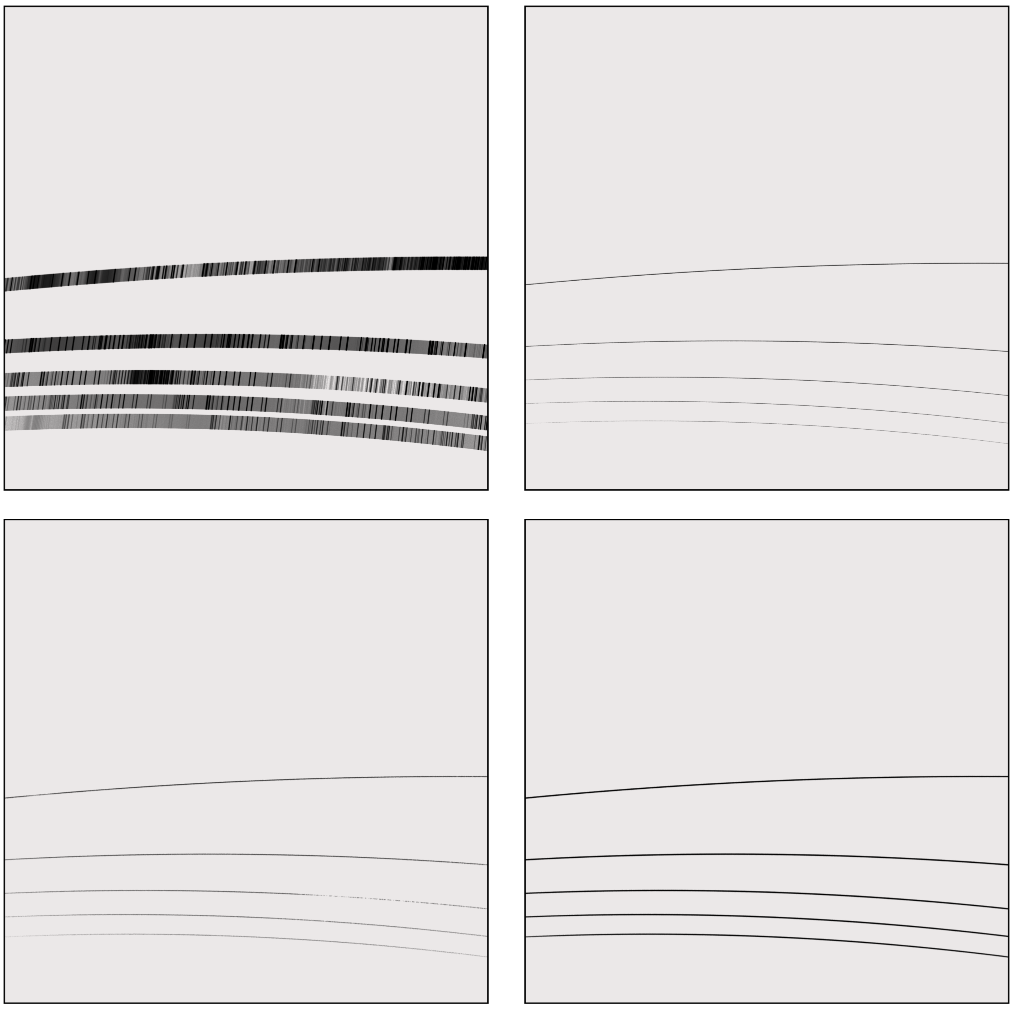


Figure 26: Examples of generated schematics for the sky, trace, GRB, and standard star frames are shown. In this context, schematics refer to images without bias, noise, or pixel sensitivity. Top left: sky. Top right: trace. Bottom left: GRB. Bottom right: standard star. All four plots are on a log scale.

results of this can be seen in figures 27 and 28. For the science frames showing a GRB and a standard star, the sky spectra have also been added.

As an additional example, figure 29 shows what the completed science frame of the standard star looks like on each of the three detectors.

6 Using the spectra simulation package

This section covers the practical usage of the spectra simulation package. At this time the code can be obtained from [35] or from the GitHub link in appendix A.4.

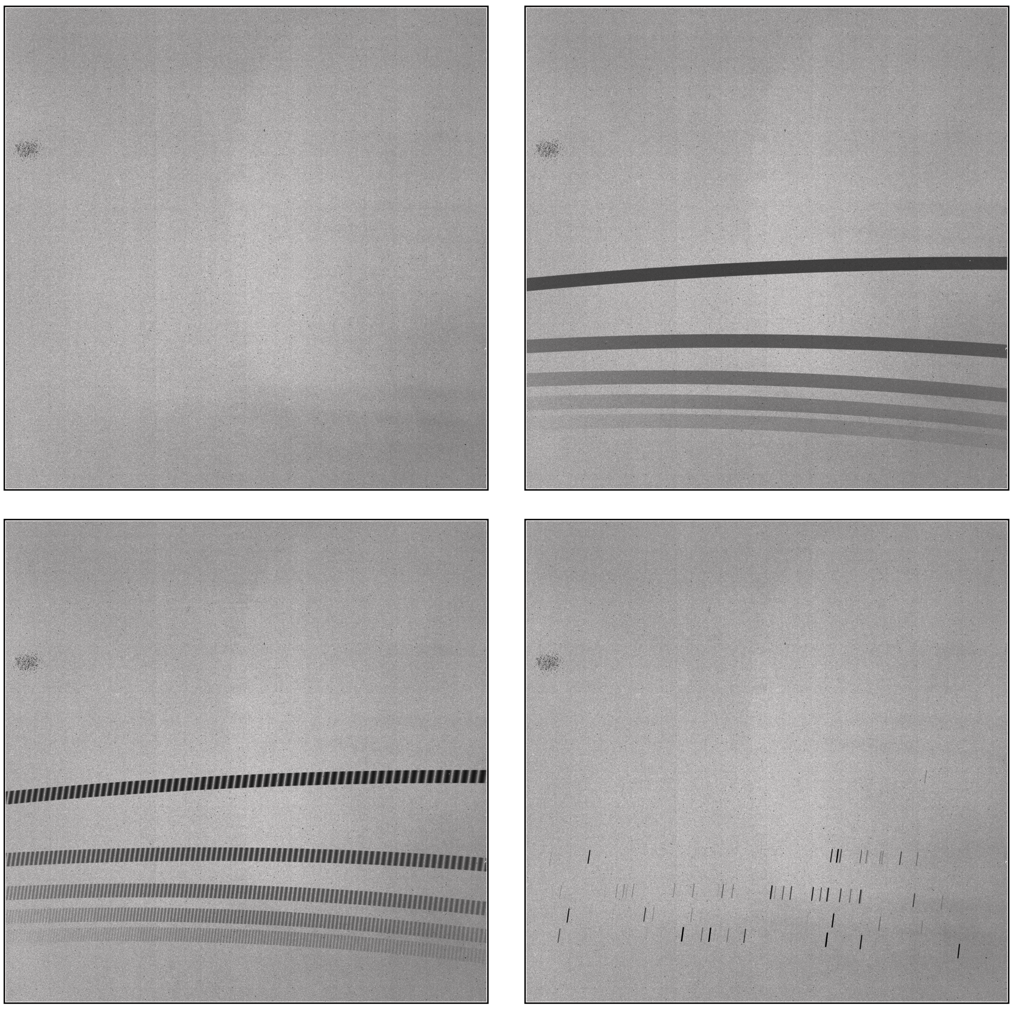


Figure 27: After incorporating bias, noise, and pixel sensitivity, the schematics from Figure 25 will appear as shown. Top left: dark. Top right: flat. Bottom left: etalon. Bottom right: HgAr. All plots here are on a log scale.

Initially, some of the terminology used in this section should be defined.

Schematic: A schematic is the clean data from the target containing no noise and no bias. This is the base image used when generating realistic spectra and also the image that should be returned by the pipeline.

CCD: Because of the way PyEchelle is written, the keyword 'CCD' will in this section refer to the three detectors including the IR detector even though the IR detector is a CMOS and not a CCD. Using the keyword 'detector' instead of 'CCD' would probably have been a better and less confusing choice, but in order to make NTEpyechelle compatible with the documentation written for PyEchelle, the keyword was not changed.

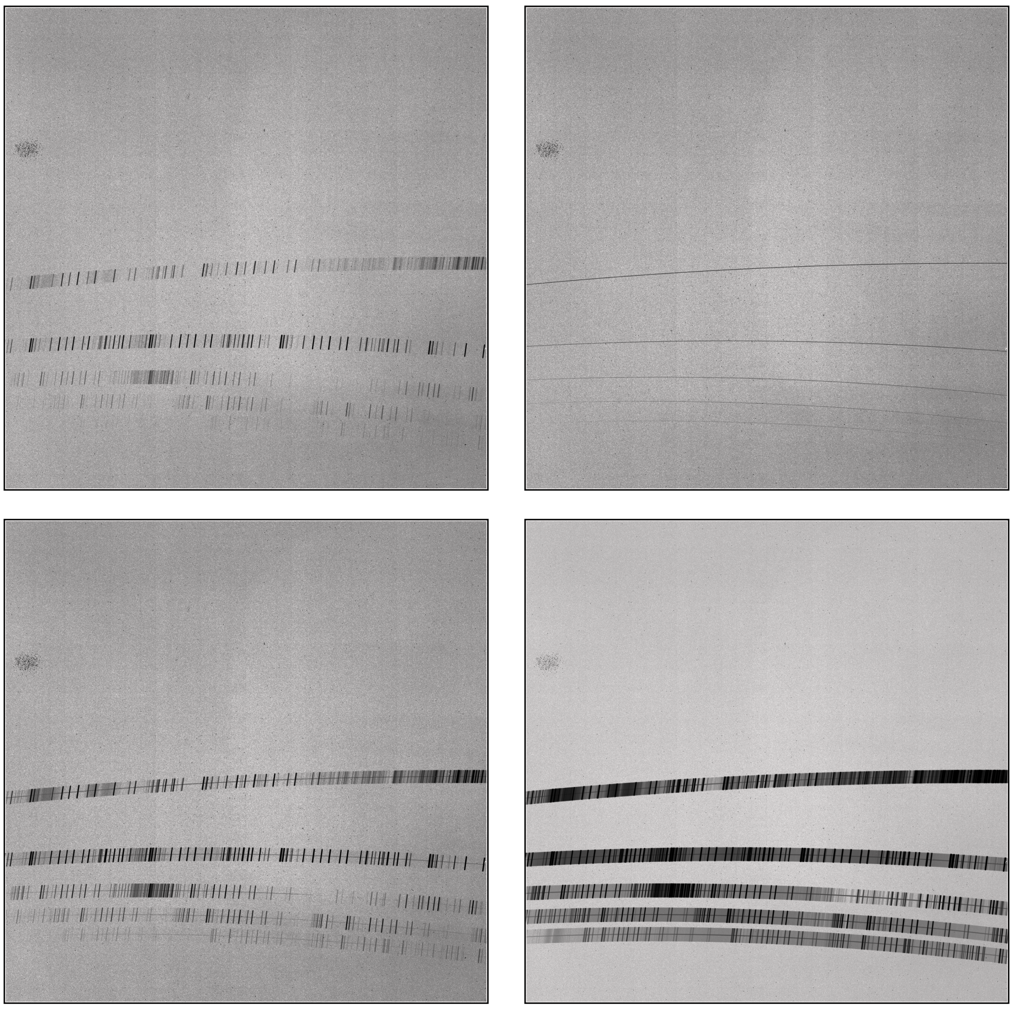


Figure 28: After incorporating bias, noise, and pixel sensitivity, the schematics from Figure 26 will appear as shown. Here the science frames (GRB and the standard star) also have an added sky spectrum. Top left: sky. Top right: trace. Bottom left: GRB. Bottom right: standard star. All plots here are on a log scale.

The NTE contains 3 CCDs where 'CCD 1' is the IR detector, 'CCD 2' is the VIS detector, and 'CCD 3' is the UV detector.

Fiber: PyEchelle also uses the keyword 'fiber' when referring to the slit. The reason fiber and slit are used interchangeably is that PyEchelle was originally made for a different spectrograph where the term fiber was used instead of slit.

NTE contains 9 different fibers with varying sizes and shapes, which are shown in table 8. The first 8 fibers with the slit shape 'r' are rectangular slits with a slit height of 22.8 arcsec and the slit width noted in the table. The last fiber with the slit shape 'p' is a round slit with the diameter noted in the table.

Therefore, if 'fiber=1' is chosen then a rectangular slit with slit width 0.5 arcsec

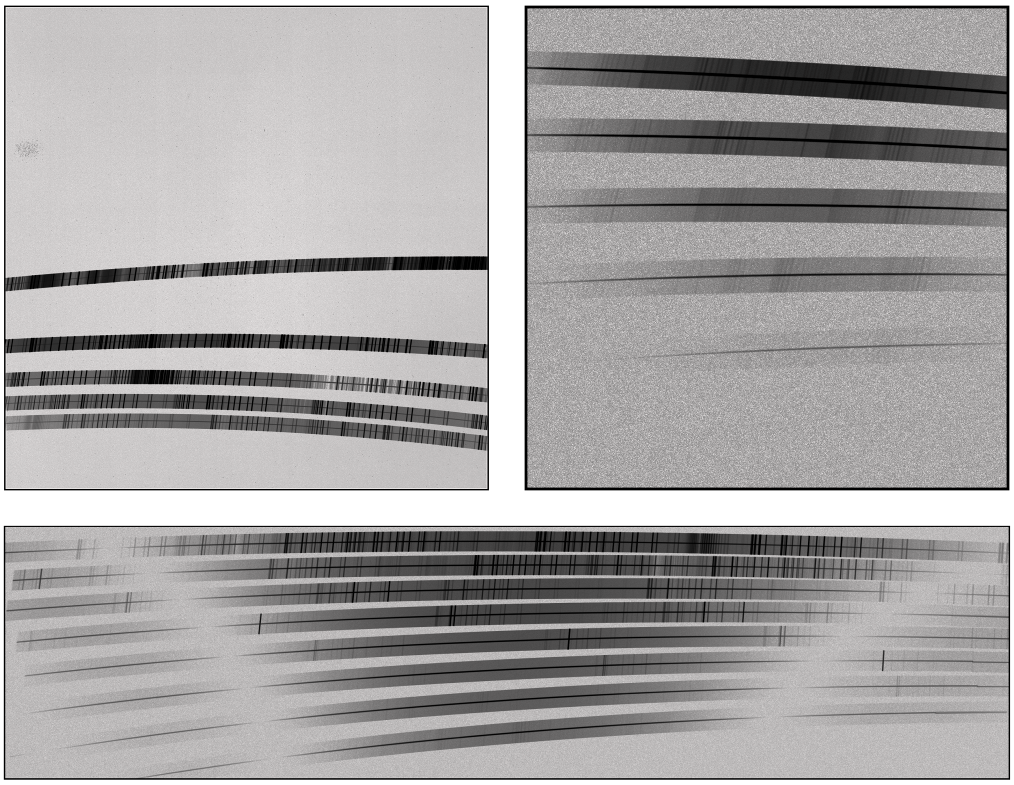


Figure 29: Generated science frames of a standard star with noise for each of the three arms. Top left: standard star on the IR detector. Top right: standard star on the UV detector. Bottom: Standard star on the VIS detector. The star is BD+174708. All figures are on a log scale and a 10 s exposure time.

and slit height 22.8 arcsec will be used.

FITS: FITS stands for Flexible Image Transport System and is a file format that can be used to store many different types of data. For this project, it is mostly used to store images such as schematics.

HDF: HDF stands for Hierarchical Data Format and is, like FITS, a file format that is used to store data. For this project, it is only used to store the data that is the input to PyEchelle. That is the affine transformations needed to calculate

Fiber	1	2	3	4	5	6	7	8	9
Slit shape	r	r	r	r	r	r	r	r	p
Slit size	0.5"	0.8"	1.0"	1.2"	1.5"	1.7"	2.0"	5.0"	0.5"

Table 8: The shape and size of each fiber. Slit shape 'r' is a rectangular slit, while slit shape 'p' is a pinhole slit. For the rectangular fibers, the noted slit size is the slit width, while they all have a slit height of 22.8 arcsec. For the pinhole fiber, the noted slit size is the diameter of the slit.

the positions of the photons when they hit the detector, and also the point-spread functions.

6.1 Set-up of the repository

For this project, it has been necessary to both write and modify packages that have been used for the simulations.

NTEpyechelle

NTEpyechelle is a modified version of PyEchelle, containing minor changes and additions like the point source option. It is used to simulate the spectra of targets. Keep in mind that this is not interchangeable with PyEchelle, since it is a modified version. The changes made to Pyechelle include changing the max value of pixels, including a point source option, and setting the GPU random seed to the current time of day unless otherwise changed. A functionality to use a hard-coded efficiency unless another one was given was also removed.

NTEsim

NTEsim is a new package that was created to make the simulation of data easier. It is used to make schematics and to add bias and noise to schematics.

It contains two main classes:

- MakeFits:
  This class uses pre-generated schematic files to create FITS files that include both bias and noise and it can be called with:

```python
from NTEsim.make_fits import MakeFits
MF = MakeFits()
```

Listing 3: Example of how to call the MakeFits class.

  MakeFits can also be used to add schematics together in case it is needed to e.g. add a sky spectrum on top of an object spectrum.

  To run MakeFits, the following packages need to be installed.

```
numpy
astropy
pyxel (only if cosmic rays are to be added)
```

Listing 4: Packages needed to run the MakeFits functions.

- MakeSchematics:

  This class makes schematics from a source and can be called with:

```
from NTEsim.make_schematics import MakeSchematics
MS = MakeSchematics()
```

Listing 5: Example of how to call the MakeSchematics class.

This class is far more complicated to run than MakeFits. When using this class it is recommended to run the code on a GPU, as it is very slow to run on a CPU. However, if it should run on a CPU, the GPU option needs to be deactivated right after the MakeSchematics class is called by using this command:

```
MS.gpu = False
```

Listing 6: Example on disabling the GPU option.

To make the code work on a CPU, it is necessary to have the following packages installed.

```
python==3.10.9
skycalc_ipy==0.1.3
joblib==1.1.1
scipy==1.10.0
numba==0.56.4
pandas==1.5.3
synphot==1.1.1
numpy==1.23.5
chardet==5.1.0
h5py==3.7.0
```

Listing 7: Packages needed to run the MakeSchematics functions

Different versions of these packages could also work, but this is the combination of versions that have been tested.

For running MakeSchematics on a GPU an additional package will be needed.

```
cudatoolkit==11.3.1
```

Listing 8: Tested CUDAToolKit version

6.2 Complete guide to using a WSL2 system

Since a WSL2 (Windows Subsystem for Linux) system was used to run the code, the following is a guide on how to set up the system to be identical to the one that was used for this project.

WSL2 is a way to run a Linux environment on Windows, and with it, it is possible to run Linux commands and applications on the Windows machine. As many programs or packages might be designed with Windows as a second priority it is a valuable tool if a Windows machine is needed for other purposes. WSL2 has become widespread enough that it has even become normal for some software to be run specifically on WSL2 instead of Windows.

From a totally clean WSL2 system, the first step is to make sure GCC is installed. This can be installed by running

```
sudo apt update
sudo apt install build-essential
```

Listing 9: Initial setup of WSL.

Next, Anaconda must be installed on the system. On WSL2, this is done the easiest by downloading the Linux file from the Anaconda website and then installing it with bash.

This can be done by downloading the 64-Bit installer for Linux, then navigating to the download folder through the terminal and writing:

```
bash name_of_installer.sh
```

Listing 10: Installing from .sh with bash.

Then restart the terminal.

Then all the necessary packages should be installed.

```
conda create -n NTEsimCUDA python=3.10.9
conda activate NTEsimCUDA
conda install pip
conda install cudatoolkit==11.3.1
pip3 install skycalc_ipy==0.1.3
conda install joblib==1.1.1
conda install scipy==1.10.0
conda install numba==0.56.4
conda install pandas==1.5.3
```

```
pip3 install synphot==1.1.1
pip3 install chardet==5.1.0
conda install h5py==3.7.0
```

Listing 11: Commands used to install needed packages.

The environment should now work for making a simulation with MakeSchematics using the GPU.
For this project, this method was used to make a working environment. If using a different method, it is important to make sure that the versions of Numba and Condatoolkit are compatible.

6.3 User guide for NTEsim: Simple

If the goal is to simply get a full set of simulated data based on the already defined presets, then NTEsim is very simple to use. The most important thing is that the 4 folders 'NTEsim', 'NTEpyechelle', 'data', and 'schematics' are all placed in the same folder.

To get a full set of data, simply make a Python script also placed in this folder and run the following code.

```python
from NTEsim.make_fits import MakeFits

MF = MakeFits()
MF.unzip_schematics()
MF.add_bias_and_noise_for_all_preset()
MF.delete_schematics()
```

Listing 12: Example of how to generate all preset frames.

This will make a full set of simulated data, with detector noise, bias, and efficiency already applied. The output data will be placed in a folder called 'Output' which is also placed in the same folder as the rest of the files. If this folder does not already exist it will be created when the code is run.

6.4 User guide for NTEsim: In-depth

For a thorough description of NTEsim, an HTML documentation exists containing a detailed description of all the options. Below is a shorter description of some of the most important features when creating a more specific data set.

6.4.1 Using a different spectrum file

Making the schematic using a different spectrum file can be done using the functions below.

```python
from NTEsim.make_schematics import MakeSchematics
from NTEpyechelle.sources import CSV

MS = MakeSchematics()

source = CSV(filepath="data/scienceframe_data/GRB/
    extended_GRBspec_2.5_withabs.dat", name="", list_like=True,
    wavelength_unit='a', flux_in_photons=False, delimiter=' ')
MS.make_custom(ccd='uv', exposure_time=5, source=source, name='
    custom.fits')
```

Listing 13: Example of generating a schematic based on a data file.

This will use CCD 3, which is the UV CCD, and an exposure time of 5 seconds to create a schematic using the spectrum in the file 'data/scienceframe_data/GRB/extended_GRBspec_2.5_withabs.dat'.

If the 'flux_in_photons' argument of the 'CSV' function is set as 'True', then it will be assumed that the fluxes in the spectrum file are provided in ph/s. If it is set to 'False' it will be assumed that the flux is provided in erg/s/cm^2/cm.
The unit of the wavelengths can be defined using the 'wavelength_unit' argument. This argument has multiple options, some of which are 'a' for ångström, 'nm' for nanometer, and 'micron' for micrometer.

To change what CCD is used, simply change the 'ccd' argument of the 'make_custom' function to either '1' or 'ir' if the IR detector should be used, '2' or 'vis' if the VIS detector should be used, or '3' or 'uv' if the UV detector should be used.

NTEsim will assume that an NVIDEA GPU is available for running the code, as the run time will be very long if a GPU is not used. As described earlier, if the GPU should not be used this can be changed by adding 'MS.gpu = False' right after importing MakeSchematics.

Furthermore, it is also possible to change which slit is used by adding 'MS.fiber'. There are 9 different slits to choose from and the corresponding fiber number is shown in table 8. An example of how to NTEsim uses 'fiber 2' is shown below.

```python
from NTEsim.make_schematics import MakeSchematics
from NTEpyechelle.sources import CSV

MS = MakeSchematics()
MS.fiber = 2
MS.overw = True

source = CSV(filepath="data/scienceframe_data/GRB/
    extended_GRBspec_2.5_withabs.dat", name="", list_like=True,
    wavelength_unit='a', flux_in_photons=False, delimiter=' ')
MS.make_custom(ccd=3, exposure_time=5,  source=source, name='custom
    .fits')
```

Listing 14: Example of how to change the slit.

As seen in table 8, fiber 2 simulates a rectangular slit with a slit height of 22.8 arcsec and a slit width of 0.8 arcsec.

Other options are changing the name of the output file, deciding whether existing schematics should be overwritten, changing which HDF file is used, and changing the efficiency file(s). The option to change the name of the output file is shown in the previous examples, while the other options are described in the HTML documentation.

Lastly, keep in mind that if the spectrum that is being simulated is a continuum, the resolution of the data needs to be quite fine. If the output file contains any weird artefacts increasing the resolution of the input, by using interpolation, could solve the problem.

Simulating a point source

If the target is a point source, this can be changed using the 'point' argument in the 'make_custom' function. The 'point' argument is connected to 3 further arguments, 'posx', 'posy', and 'seeing'. 'posx' and 'posy' are the placement along the slit on the two axes, and both have to be a number in the range 0 to 1. This is also explained in section 5.8. If the position arguments are not used, the position will be 0.5 on both axes. If the seeing argument is not used, the seeing will be 0.8".

```
from NTEsim.make_schematics import MakeSchematics
from NTEpyechelle.sources import CSV

MS = MakeSchematics()

```

```
source = CSV(filepath="data/scienceframe_data/GRB/
    extended_GRBspec_2.5_withabs.dat", name="", list_like=True,
    wavelength_unit='a', flux_in_photons=False, delimiter=' ')
MS.make_custom(ccd=3, exposure_time=5, source=source, name='custom.
    fits', point=True, posx=0.5, posy=0.3, seeing=0.8)
```

Listing 15: Example of how to simulate a point source.

Here a point source is simulated with the position (0.5,0.3) and a seeing of 0.8".

6.4.2 Adding noise and bias

If a lot of simulations have been made and it is necessary to add noise and bias to all the schematics but not to the rest of the files. If all the schematics are marked with a keyword like 'stellar' in the filename, this can easily be done by placing all the files in a zip folder inside the 'schematics' folder. It is possible to add bias and noise to all the files inside this zip folder whose filename contains a specific keyword. An example of this is shown below.

```
from NTEsim.make_fits import MakeFits

MF = MakeFits()
MF.unzip_schematics()
MF.add_bias_and_noise_for_keyword(keyword='stellar', nr=1, add_sky=
    True, sky_factor=0.2)
MF.delete_schematics()
```

Listing 16: Example of how to add bias and noise to multiple schematics based on a keyword.

The “add_sky” keyword determines if a sky spectrum should be added on top, and the “sky_factor” is how strong the sky spectrum should be. For the IR detector sky_factor=1 corresponds to a 0.5s exposure, for the VIS detector it corresponds to a 3s exposure, and for the UV detector, it corresponds to a 233s exposure.

The zip files are used since the schematic files are very large. This is however not necessary, and by removing 'unzip_schematics' and 'delete_schematic', the files can be placed directly inside the 'schematics' folder instead of inside a zip file.

Furthermore, it is required for the filenames of the schematics, to begin with either UV, VIS, or IR, as this is how the code determines which bias and noise to add.

Similar to the function which has just been explained, another function exists which does the opposite. This function will add noise and bias to all files where the filename does NOT contain a specific keyword. An example is shown below where bias and noise are added to all schematics, but a sky spectrum is only added to the files whose filename contains 'grb25'.

```python
from NTEsim.make_fits import MakeFits

MF = MakeFits()
MF.add_bias_and_noise_for_keyword(keyword='grb25', nr=1, add_sky=
    True)
MF.add_bias_and_noise_for_not_keyword(keyword='grb25', nr=1)
```

Listing 17: Example of how to add bias and noise to all frames while adding the sky to only the GRB25.

For both of these functions, it is possible to use multiple keywords at once. An example of how this is done is shown below.

```python
from NTEsim.make_fits import MakeFits

MF = MakeFits()
MF.unzip_schematics()
MF.add_bias_and_noise_for_keyword(keyword=['stellar', 'grb25'], nr
    =1, add_sky=True)
MF.add_bias_and_noise_for_not_keyword(keyword=['stellar', 'grb25'],
     nr=1, add_sky=True)
MF.delete_schematics()
```

Listing 18: Example on how to use an array as keyword.

6.4.3 Adding schematics together

In cases where it is necessary to add multiple files together, e.g. if sky spectrum has to be added to another file, this can be done as shown below.

```python
from NTEsim.make_fits import MakeFits

MF = MakeFits()
files_to_add = ['schematic1.fits', 'schematic2.fits',
                'schematic3.fits']
factors = [1, 1, 1]
output_name = 'combined.fits'
MF.add_schematics_and_add_noise(list_of_names=files_to_add,
    list_of_scales=factors, output_name=output_name)
```

Listing 19: Example of adding schematics.

The 'factors' input will multiply the data from the schematic with the factor. This is mostly in case the files that are added together do not have the same exposure time.

6.4.4 Simulating cosmic rays

In order to use the cosmic ray function, an additional package is required. The installation process of Pyxel depends on which machine is being used, and a guide can be found on the website:
https://esa.gitlab.io/pyxel/doc/stable/tutorials/install.html

For this project, Pyxel version 1.7 was used. The option to include cosmic rays is a part of the MakeFits class, and an example of how it is used is shown below.

```python
from NTEsim.make_fits import MakeFits

MF = MakeFits()
MF.cosmic_rays = True
```

Listing 20: Example of how to enable the cosmic ray simulation.

After this proceed to make the fits files as explained above.

When including cosmic rays, the filename must contain the exposure time followed by 'sec', for the right amount of cosmic rays to be produced. An example of this is 'ir_0.5sec_sky.fits'.

As explained earlier, this is not an ideal implementation of cosmic rays, since the value of the cosmic ray hits is simply set to the maximum value of the image and it is designed for a spaced-based telescope. It is meant for pipeline testing only.

6.5 Future updates

Some files may need to be replaced in the future when new data is acquired. In some cases, this is as simple as just replacing the file, while in other cases multiple things need to be done in order for NTEsim to use the new files. Therefore the following is a guide to how to replace data files.

6.5.1 Flats

Depending on which CCD is chosen, all the files in either 'data/flat_ir', 'data/flat_vis', or 'data/flat_uv' are used to create the flat. At the moment only data from the IR detector has been provided, and the two other detectors therefore have placeholder files for the flats. At the moment the VIS and UV detector instead makes the flat from a normal distribution with a mean of 1 and a standard deviation of 0.01.

In the future, it will therefore be necessary to include data for both the VIS and the UV arm. Apart from adding the new data files to 'data/flat_vis' and 'data/flat_uv' some changes also need to be made to the code. However, this should simply be a matter of replacing the code used in the VIS and UV detectors with the code used for the IR detector.

6.5.2 Noise

When creating the combined noise, all the files in either 'data/noise_ir', 'data/noise_vis', or 'data/noise_uv' are used. However, like with the flats, data has only been provided for the IR detector.

As with the flats, apart from adding the data files for UV and VIS, the code section used for the VIS and UV detector needs to be replaced with the code used for the IR detector.

6.5.3 New ZEMAX data

If a new set of data is simulated from ZEMAX, then a new HDF model file for PyEchelle must be made. All the ZEMAX files should be added to the 'HDF_generation' folder, while the PSFs must be put into the subfolders. The code for reading these files is written in a way where all the files must be formatted in the same manner. Therefore it is necessary to check whether the new ZEMAX files are formatted as the existing ZEMAX files.

After replacing the files, run the 'generateHDF.py' file from within the 'HDF_generation' folder, and a new HDF model will be added in the NTEpyechelle folder. If the old HDF model should not be replaced, the filename of the HDF model can be changed using the 'outfile' variable in 'generateHDF.py'.

6.5.4 Changing the HDF file

Changing which HDF file is used for generating a schematic is very simple, and an example is shown below.

```python
from NTEsim.make_schematics import MakeSchematics
import h5py
```

```

MS = MakeSchematics()
MS.h5file = h5py.File('NTEpyechelle/models/NTE_2.hdf', 'r')
```
Listing 21: Example of how to change the HDF file used for schematic generation

6.5.5 New efficiency files

When new efficiency data is provided, the files in the 'data/efficiency' folder must be updated. If the new files are in the same units as the existing ones, then it should be as simple as replacing the existing files with the new ones and running the 'make_effs.py' file from within the 'data/efficiency' folder.

6.6 Applying NTEsim to a different instrument

There are two major things that must be done before this can be used on other instruments.

Firstly, there is the ZEMAX simulation. It is required to create the raytracing model with ZEMAX. In the folder 'HDF_generation,' there is an example macro called 'NTE_VISNIR_footprint_JV_23_02_17.zpl' that generates the necessary ZEMAX files. After generating the ZEMAX files, numerous changes must be made to the 'generateHDF.py' file. Unfortunately, these changes must be made directly in the code. Firstly, the code uses a keyword to find the ZEMAX files that should match the instrument. Then update the detector specifications, spectrograph grating specifications, and slit specifications. Additionally, update the variables 'aperturenames', 'slitnames', 'detector_area', and 'slit_nr_param'. The final changes relate to the PSFs. Starting from line 341 in 'generateHDF.py', there is a loop over the detectors. Here, it is needed to modify the 'CCD_name' and 'input_dir' to suit the new requirements. If the instrument also has three arms (UV, VIS, and NIR), the number of required file changes is significantly reduced. Alternatively, a new script can be created to generate the HDF while noting the variables that must be included from 'generateHDF.py'.

Secondly, it is necessary to generate the total efficiency of the telescope for each detector.

Once these two tasks are completed, it should be possible to run the 'make_schematics' part of the NTE package. An example of how to use it with the new files is shown below.

```python
from NTEsim.make_schematics import MakeSchematics
from NTEpyechelle.efficiency import CSVEfficiency
import h5py

MS = MakeSchematics()
MS.h5file = h5py.File('path_to_hdf_file', 'r')
ccd1_eff = CSVEfficiency('ccdname', 'path_to_eff_csv')
ccd2_eff = CSVEfficiency('ccdname', 'path_to_eff_csv')
ccd3_eff = CSVEfficiency('ccdname', 'path_to_eff_csv')
MS.efficiency = [ccd1_eff, ccd2_eff, ccd3_eff]
```

Listing 22: Example of how to use 'MakeSchematics' for a different intrument

When this is done it should be possible to run the 'make_schematics' part of the NTEsim package as normal.

To adapt the 'make_fits' part of NTEsim is relatively easy. There are three different variables that need to be changed for each detector: bias, noise, and flat field. The code is initially written for UV, VIS, and IR detectors, but it is not a strict requirement. The bias, noise, and flat field should all be images that correspond to the detectors. In other words, there should be a bias, noise, and flat field value for each individual pixel. It's important to note that the noise value represents the standard deviation of Gaussian noise in the pixel. An example of how to change the files is shown below.

```python
from NTEsim.make_fits import MakeFits

MF = MakeFits()
self.ir_bias = DATA_FOR_IR_BIAS
self.ir_std = DATA_FOR_IR_NOISE
self.ir_flat = DATA_FOR_IR_FLAT
self.vis_bias = DATA_FOR_VIS_BIAS
self.vis_std = DATA_FOR_VIS_NOISE
self.vis_flat = DATA_FOR_VIS_FLAT
self.uv_bias = DATA_FOR_UV_BIAS
self.uv_std = DATA_FOR_UV_NOISE
self.uv_flat = DATA_FOR_UV_FLAT
```

Listing 23: Example of how to change bias, noise, and flatfielding files for MakeFits. All data files should be 2d NumPy arrays.

Doing this should be all that is needed to run most of the 'make_fits'. However, if cosmic rays are needed, a Pyxel setup for the specific detector needs to be created, where cosmic rays are the only meaningful contribution. Then, in the 'add_bias_and_noise' method of the MakeFits class in 'make_fits', the YAML file used with Pyxel should be changed for each detector.

7 Initial results of the pipeline

Tom Reynolds worked on reducing the data using a pipeline package called PypeIt. From his work, initially reduced spectra and S/N data were acquired based on the simulated data.

Based on this, it is possible to compare the output from the ETC to the output of PipeIt to see if they appear as expected. Tom only worked on the visual detector, so this is the arm that will be compared.

In Figure 30, the different extracted spectra are shown. From the figure, it is evident that the PipeIt output is very close to the raw data. It is not perfect, but with more work, it shows great potential. Additionally, it is clear that the magnitude normalization has made the spectrum steeper, even though a magnitude that should not lead to significant changes was chosen. Surprisingly, the ETC output resembles the raw data more closely than the magnitude-normalized data used for the ETC. In Figure 39, it is illustrated without dividing the ETC output with the telescope efficiency.

PipeIt will also provide a noise level, allowing for a comparison of the signal-to-noise output between the ETC and PipeIt. Figure 31 shows the signal-to-noise values from both the ETC and PipeIt. It is apparent from the figure that the overall downward trend is similar, but the slope is more pronounced for the ETC.

8 Conclusion

Based on an existing exposure time calculator for X-Shooter, originally created by Bjarne Thomsen, a modified version has been developed for the NOT Transient Explorer, which is currently under development. The exposure time calculator will be an important tool for users of the NTE to optimize their observations and obtain high-quality data from the NTE.
The tests conducted on the calculator showed a decent agreement with calculations carried out by hand for the expected signals. For the UV and VIS arms, the calcu-

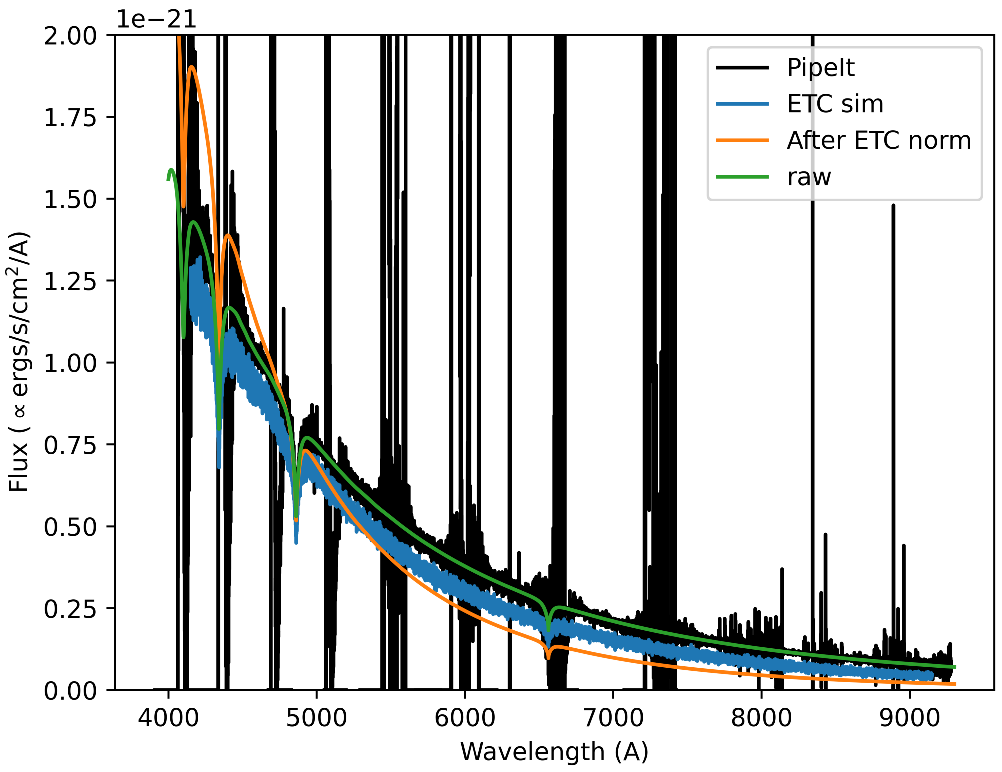


Figure 30: Flux as a function of wavelength for GD71. Black: Extracted spectrum from PipeIt after it was calibrated with a standard star. Blue: The output spectrum from the ETC divided by the telescope efficiency. Orange: Spectrum input into the ETC. It is the GD71 spectrum normalized to an AB magnitude of 12.9 at 5000A. Green: Raw spectrum from GD71. The PipeIt and raw data have both been scaled with 2.4e-8. A plot without dividing the ETC output with the efficiency can be found in appendix A.3.

lator generally gave a lower signal value than expected. On the other hand, the IR arm gave a higher signal value than expected when compared to the simple calculations.

Generally, the results of the ETC live up to expectations, and it is difficult to perform further testing without real data from the NTE.

Furthermore, a set of simulated data were created resembling the expected output from the NTE. This was done using existing tools like PyEchelle and ZEMAX. The simulated data is needed to test the pipeline which will be used to process the data from NTE and also help optimize the instrument while it is still in the development phase.

To make the data as realistic as possible, noise, cosmic rays, and an option for

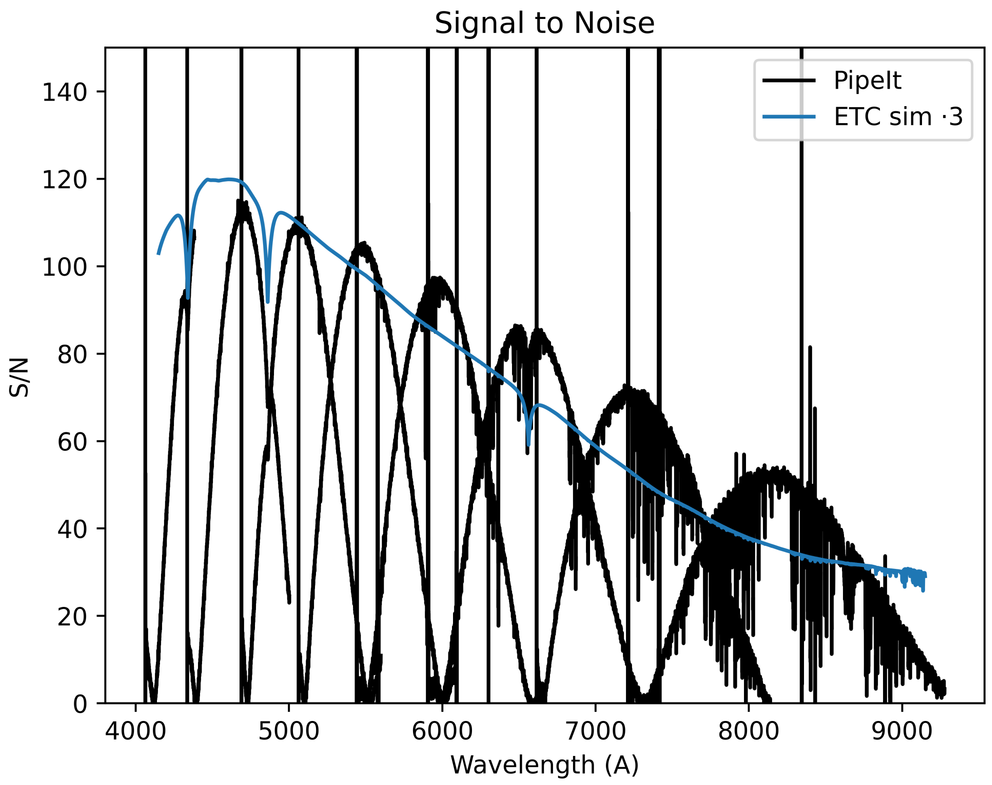


Figure 31: Signal-to-noise from PipeIt and the ETC for GD71. Black: Signal-to-noise for the arms of the VIS detector after PipeIt reduction. Blue: Output signal-to-noise (multiplied by 3) from the ETC with an AB magnitude of 12.9 at 5000A.

making a point source were included. Since the main purpose of creating the data is for testing the pipeline, it is important to include these aspects to ensure that the pipeline can handle all the different features that will inevitably be included in an observation.

The output from the ETC has also been compared to the output of an early version of the pipeline to reduce the data for the NTE. Using the same spectrum for the ETC as well as the simulated data reduced by the pipeline showed a decent correlation and looks promising.

Validating the exposure time calculator against real data from the NTE would be a crucial next step. Although the calculator relies heavily on interpolated spectra and therefore is expected to produce a reasonable signal-to-noise ratio, it is important to acknowledge the inherent assumptions. This includes the spectra obtained from ESO SkyCalc, which are a good baseline, but they are not specific to the NTE location or the variable sky condition. As a result, it is anticipated that the exposure time calculator will not perfectly match the observations. Nevertheless, through observations with NTE and minor adjustments, it would be possible to improve its performance.

For the simulated data it would be valuable to incorporate realistic simulations for detector effects, as well as including realistic cosmic rays. Additionally, the pipeline should be optimized using these simulated data.

9 References

[1] Vernet, J. et al. “X-shooter, the new wide band intermediate resolution spectrograph at the ESO Very Large Telescope”. In: *Astronomy & Astrophysics* (Oct. 2011).

[2] *NTE: About the project.* Last accessed 14.02.2023. URL: `https://nte.nbi.ku.dk/about/`.

[3] Fynbo, J. P. U.; Covino, S.; Heintz, K. E. *NTE Science Cases.*

[4] *Science with NTE.* Last accessed 14.02.2023. URL: `https://nte.nbi.ku.dk/science-with-nte/`.

[5] *ZEMAX.* Last accessed 22.05.2023. URL: `https://www.zemax.com/`.

[6] Stürmer, J. et al. *Echelle++, a Fast Generic Spectrum Simulator.* 2019. DOI: `10.1088/1538-3873/aaec2e`.

[7] Boffin, H. et al. “ESO’s Exposure Time Calculator 2.0”. In: *SPIE* (2020). DOI: `10.1117/12.2562236`.

[8] Eversberg, T.; Vollmann, K. *Spectroscopic Instrumentation - Fundamentals and Guidelines for Astronomers.* First Edition. Springer Praxis Books. Springer, 2015. ISBN: 978-3-662-44534-1. DOI: `10.1007/978-3-662-44535-8`.

[9] Spring, K. R.; Fellers, T. J.; Davidson, M. W. *Introduction to Charge-Coupled-Devices (CCDs).* Last accessed 05.04.2023. URL: `https://www.microscopyu.com/digital-imaging/introduction-to-charge-coupled-devices-ccds`.

[10] Coates, C.; Mullan, A. *What is an EMCCD Camera?* Last accessed 19.01.2023. 2021. URL: `https://andor.oxinst.com/learning/view/article/electron-multiplying-ccd-cameras`.

[11] *EMCCDs: The Basics.* Last accessed 05.04.2023. URL: `https://www.princetoninstruments.com/learn/camera-fundamentals/emccds-the-basics`.

[12] Janesick, J. et al. “New advancements in charge-coupled device technology: subelectron noise and 4096 x 4096 pixel CCDs”. In: *SPIE* 1242 (July 1990).

[13] Tiffenberg, J.; Sofo-Haro, M.; Drlica-Wagner, A., et al. “Single-Electron and Single-Photon Sensitivity with a Silicon Skipper CCD.” In: *Physical review letters* 119(13) (Sept. 2017).

[14] *CCD vs. CMOS.* Last accessed 05.04.2023. URL: https://www.teledynedalsa.com/en/learn/knowledge-center/ccd-vs-cmos/.

[15] Rieke, G. “Infrared Detector Arrays for Astronomy”. In: *Annual Reviews* (Sept. 2007).

[16] Virtanen, P. et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[17] Harris, C. R. et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[18] *ESO SkyCalc.* Last accessed 20.05.2023. URL: https://www.eso.org/observing/etc/bin/gen/form?INS.MODE=swspectr+INS.NAME=SKYCALC.

[19] Noll, S. et al. “An atmospheric radiation model for Cerro Paranal”. In: *Astronomy & Astrophysics* (July 2012).

[20] Jones, A. et al. “An advanced scattered moonlight model for Cerro Paranal”. In: *Astronomy & Astrophysics* (Dec. 2013).

[21] *X-SHOOTER.* Last accessed 20.05.2023. URL: https://www.eso.org/sci/facilities/paranal/instruments/xshooter.html.

[22] Andersen, M. I. et al. *Developing the NOT Transient Explorer (NTE) for the NOT.*

[23] Drlica-Wagner, A. et al. “Characterization of skipper CCDs for cosmological applications”. In: (2020).

[24] *Useful Astronomical Data.* Last accessed 21.03.2023. URL: https://www.astronomy.ohio-state.edu/martini.10/usefuldata.html.

[25] Blanton, M. R.; Roweis, S. “K-Corrections and Filter Transformations in the Ultraviolet, Optical, and Near-Infrared”. In: *The Astronomical Journal* 133.2 (Feb. 2007), pp. 734–754. DOI: 10.1086/510127. arXiv: astro-ph/0606170 [astro-ph].

[26] Bessell, M. S.; Castelli, F.; Plez, B. “Model atmospheres broad-band colors, bolometric corrections and temperature calibrations for O - M stars”. In: *Astronomy and Astrophysics* 333 (May 1998), pp. 231–250. URL: https://ui.adsabs.harvard.edu/abs/1998A&A...333..231B.

[27] Littlefair, S. *PHY217 Observational Astronomy: Photometry II, calibrating photometry (L04)*. Last accessed 01.02.2023. URL: `https://slittlefair.staff.shef.ac.uk/teaching/phy217/lectures/principles/l04/`.

[28] Maihara, T. et al. "OBSERVATIONS OF THE OH AIRGLOW EMISSION". In: *The Astronomical Society of the Pacific* (1993). DOI: `10.1086/133259`.

[29] Nymann-Lynggaard, M.; Henneberg, C. V. *Exposure Time Calculator for NTE*. Version v1.0.2. May 2023. DOI: `10.5281/zenodo.7953775`. URL: `https://doi.org/10.5281/zenodo.7953775`.

[30] Léna, P. et al. *Observational Astrophysics*. Third Edition. Astronomy and Astrophysics Library. Springer, 2012. ISBN: 978-3-642-21814-9. DOI: `10.1007/978-3-642-21815-6`.

[31] Casini, R.; Nelson, P. G. "On the Intensity Distribution Function of Blazed Reflective Diffraction Gratings". In: *Journal of the Optical Society of America. A, Optics, image science, and vision* 31.10 (Oct. 2014), pp. 2179–2184. DOI: `10.1364/JOSAA.31.002179`.

[32] Schwarz, K. *Darts, Dice, and Coins: Sampling from a Discrete Distribution*. Last accessed 21.03.2023. URL: `https://www.keithschwarz.com/darts-dice-coins/`.

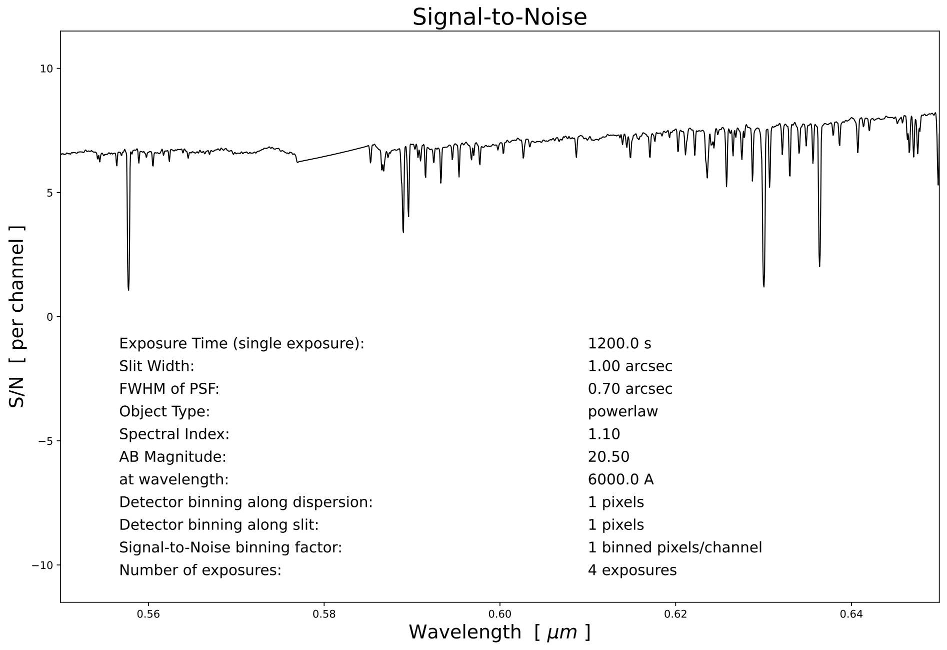
[33] A. Kramida et al. NIST Atomic Spectra Database (ver. 5.10), [Online]. Available: `https://physics.nist.gov/asd` [2023, May 21]. National Institute of Standards and Technology, Gaithersburg, MD. 2022.

[34] Prod'homme, T. et al. "Pyxel: the collaborative detection simulation framework". In: *SPIE* (2020). DOI: `10.1117/12.2561731`.

[35] Nymann-Lynggaard, M.; Henneberg, C. V. *Simulated Data for NTE*. Version v1.0.2. May 2023. DOI: `10.5281/zenodo.7953776`. URL: `https://doi.org/10.5281/zenodo.7953776`.

A Appendix

A.1 S/N comparisons



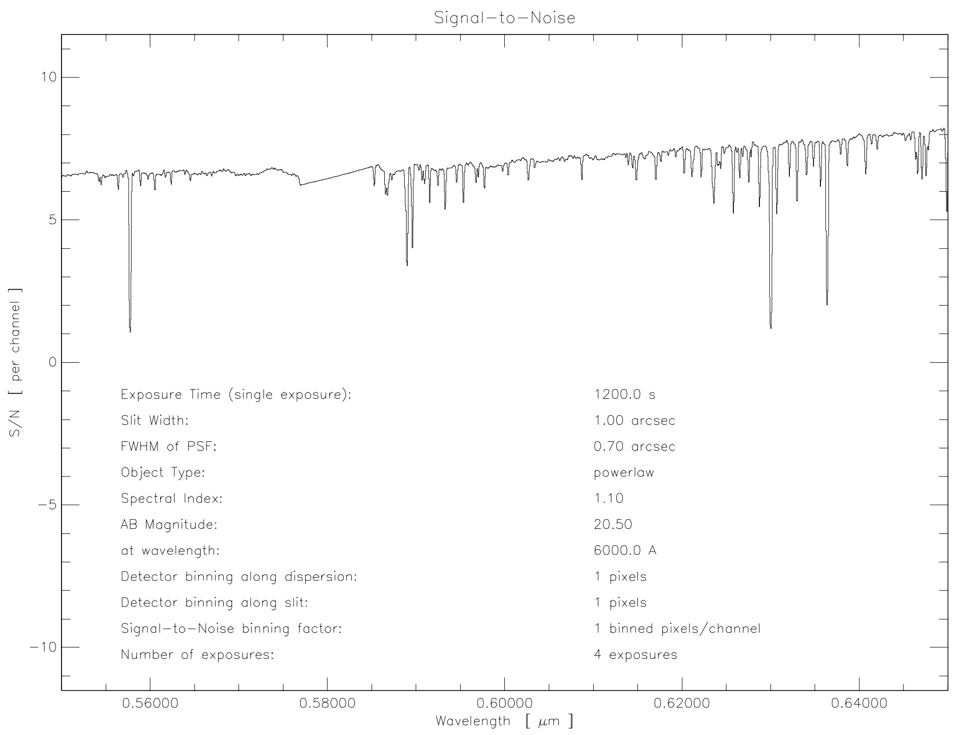
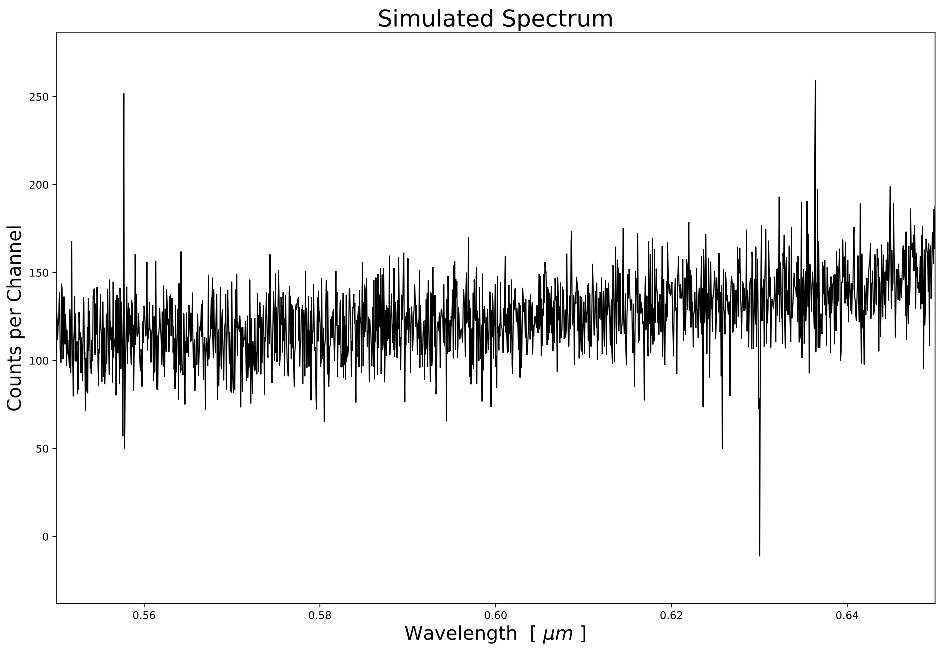


Figure 32: Comparison between output S/N of the Python code (top) vs. the IDL code (bottom). No moon is included in the noise spectrum for this comparison. The Python and the IDL version yield almost identical results. The wavelength range is 5500-6500.



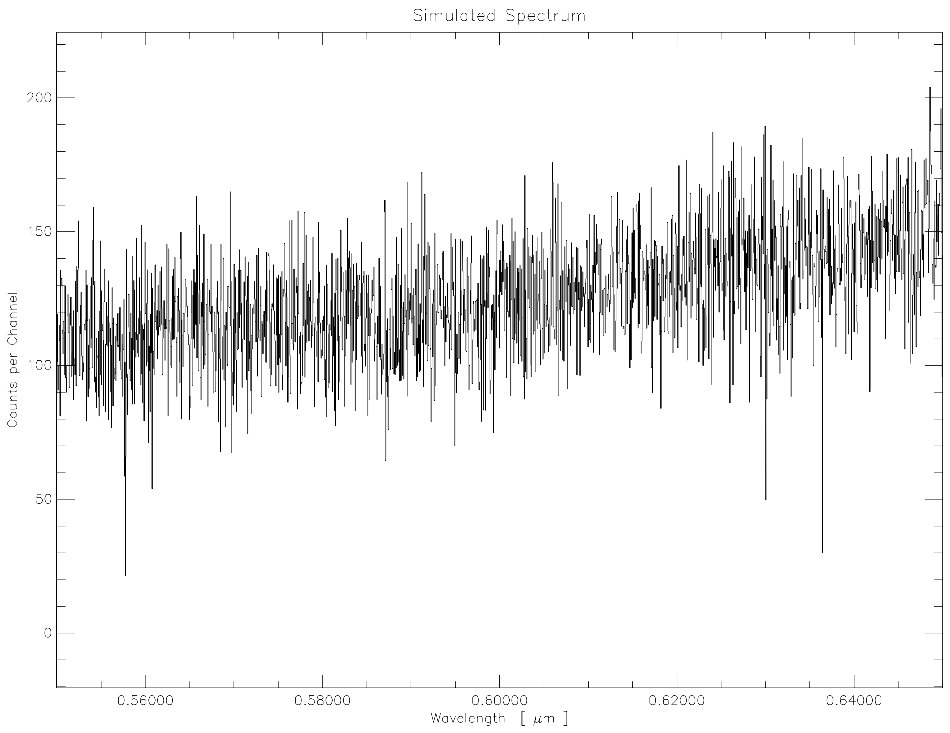
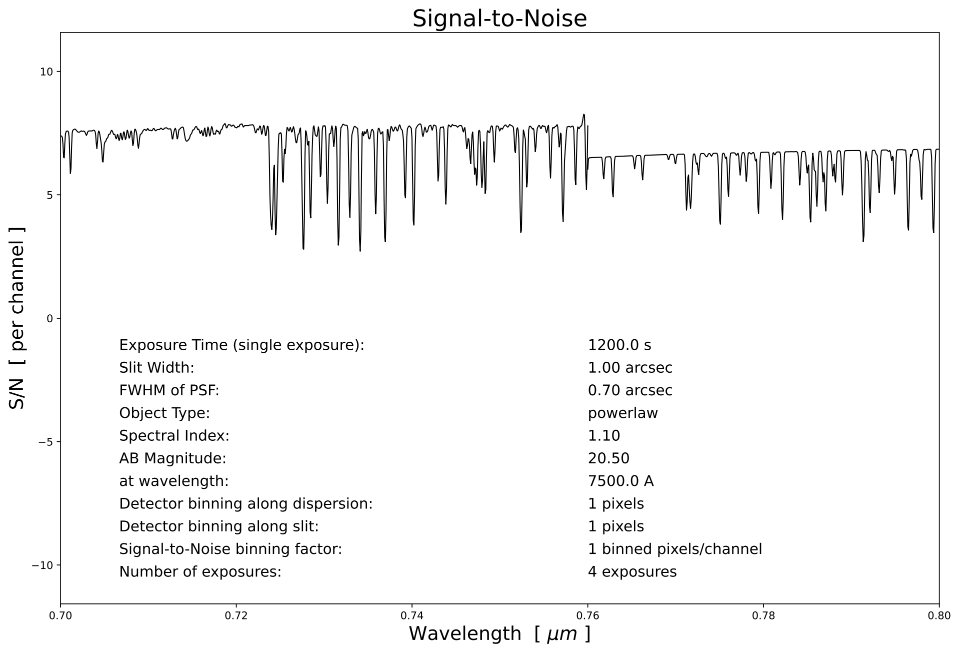


Figure 33: Comparisons between the generated spectrum from the Python version (top) vs. the IDL version (bottom) using the same parameters as in Figure 32. The wavelength range is 5500-6500.



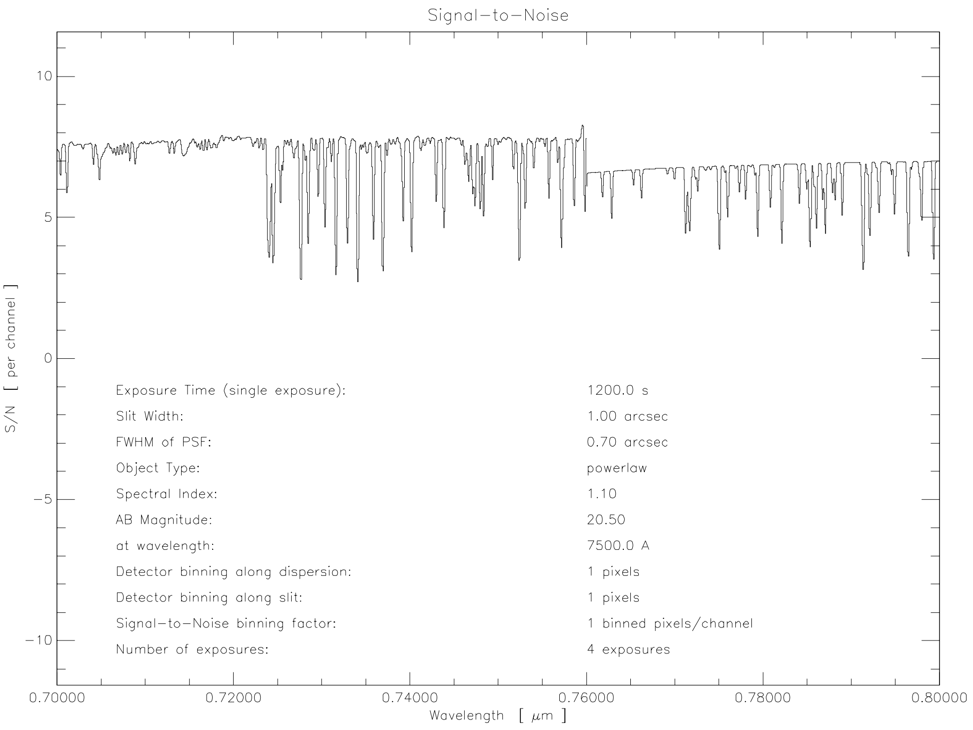
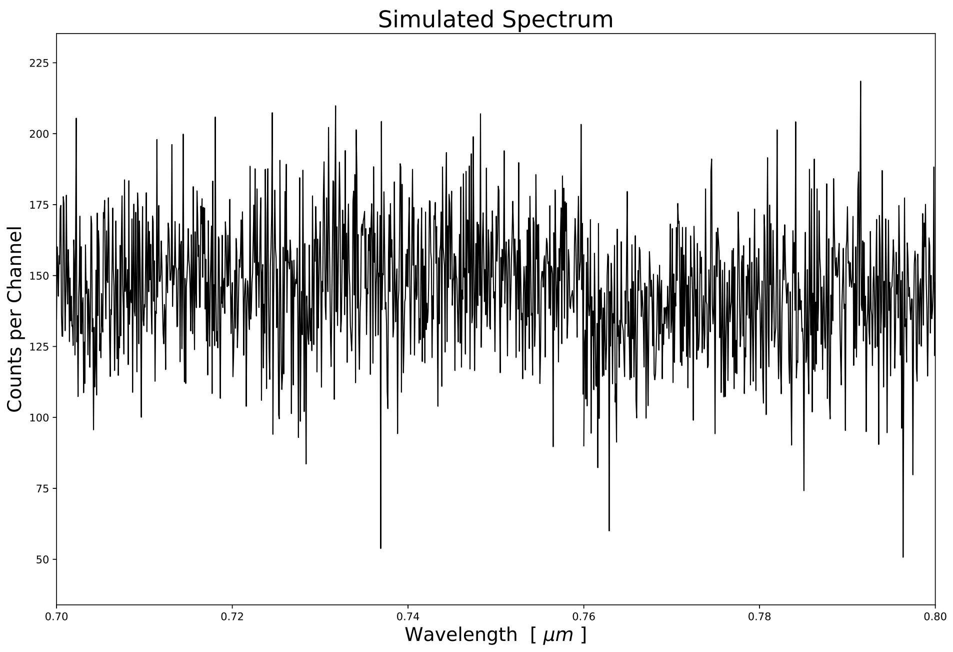


Figure 34: Comparison between output S/N of the Python code (top) vs. the IDL code (bottom). No moon is included in the noise spectrum for this comparison. The Python and the IDL version yield almost identical results. The wavelength range is 7000-8000.



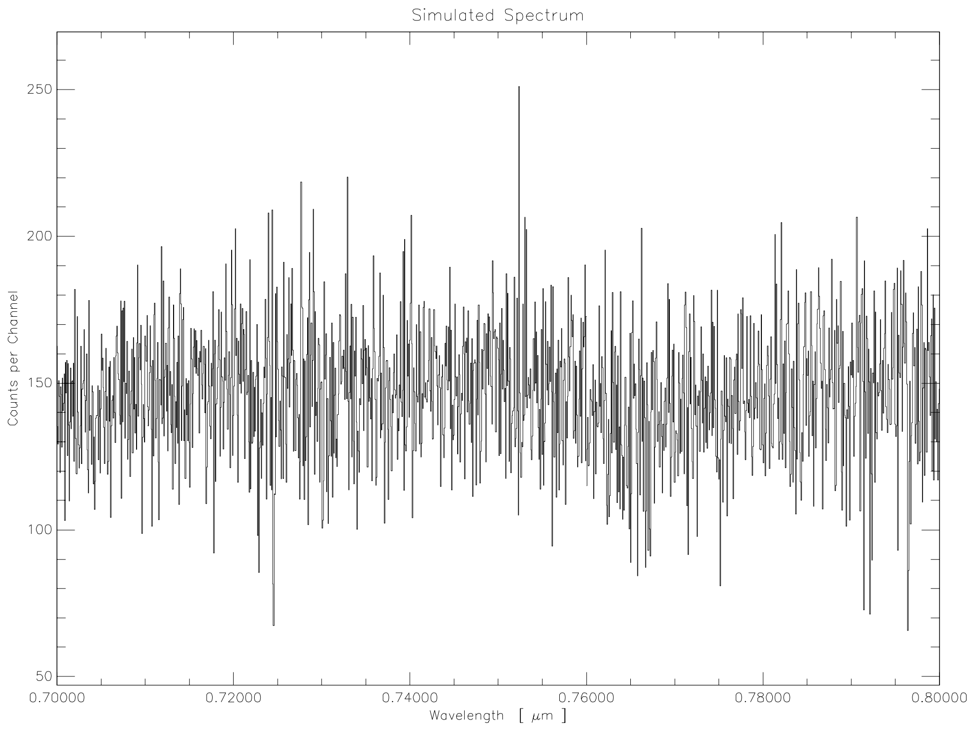
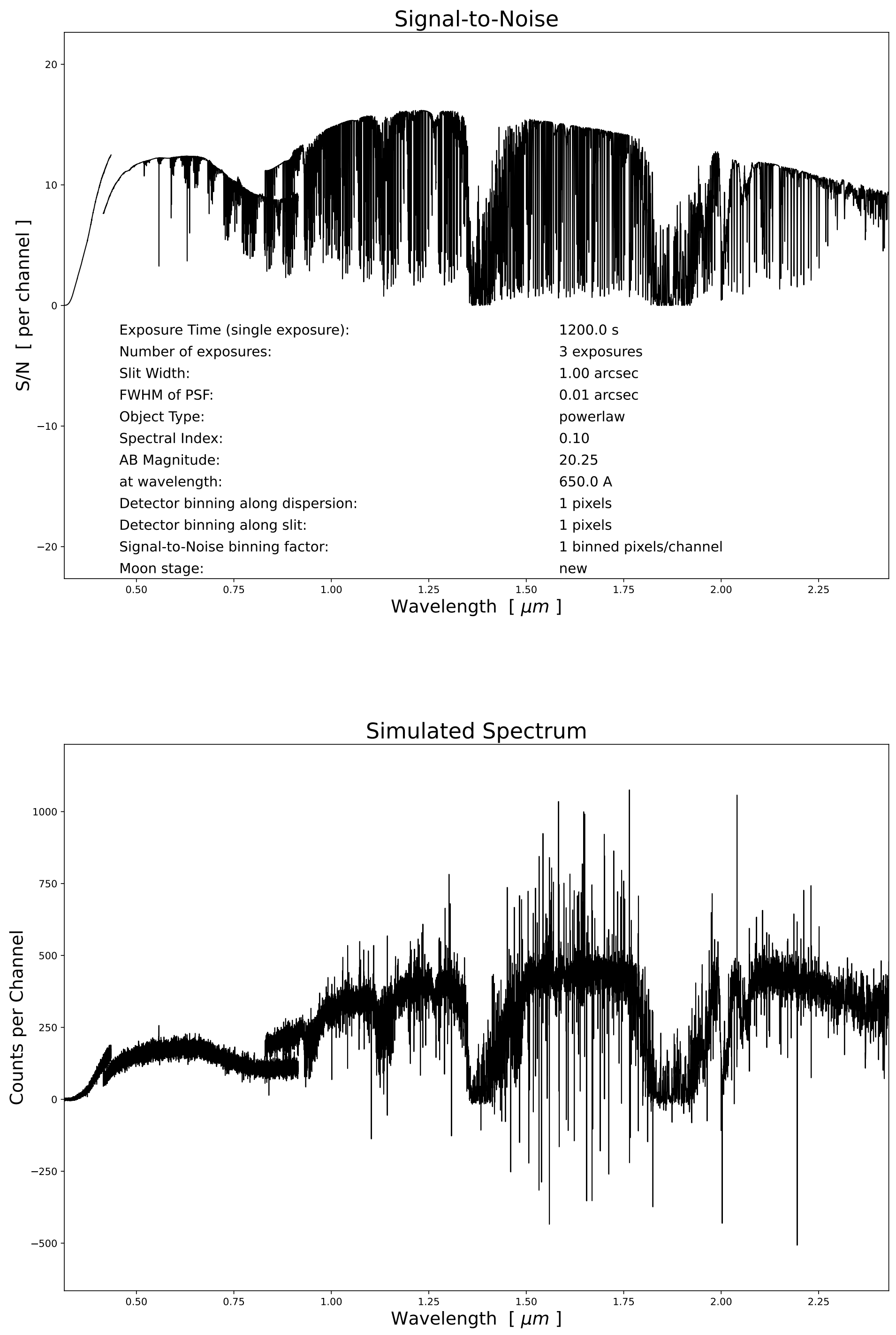


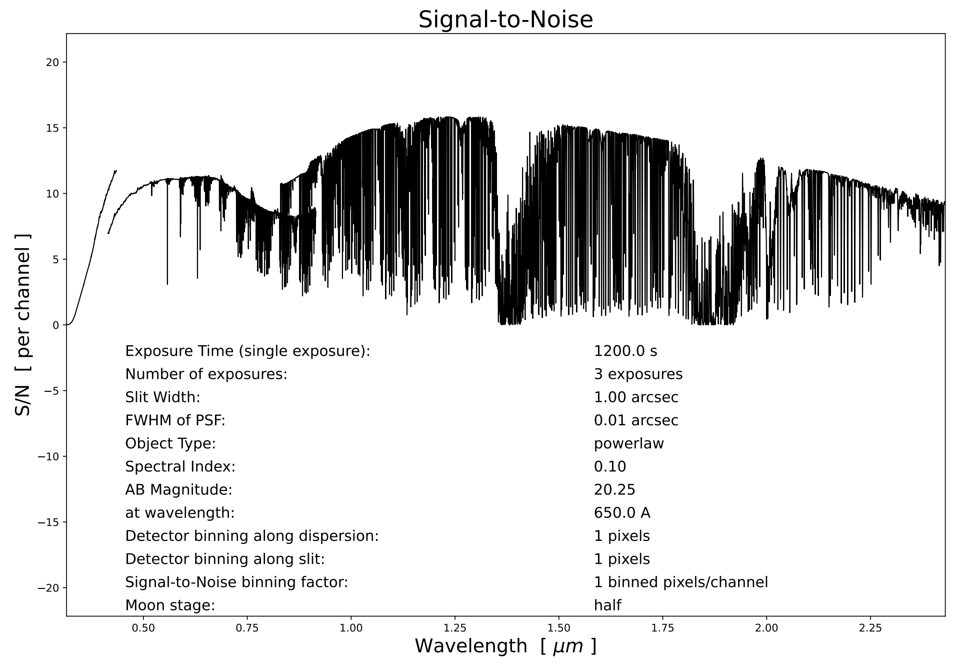
Figure 35: Comparisons between the generated spectrum from the Python version (top) vs. the IDL version (bottom) using the same parameters as in Figure 34. The wavelength range is 7000-8000.

A.2 Moon phase comparisons



Exposure Time (single exposure):	1200.0 s
Number of exposures:	3 exposures
Slit Width:	1.00 arcsec
FWHM of PSF:	0.01 arcsec
Object Type:	powerlaw
Spectral Index:	0.10
AB Magnitude:	20.25
at wavelength:	650.0 A
Detector binning along dispersion:	1 pixels
Detector binning along slit:	1 pixels
Signal-to-Noise binning factor:	1 binned pixels/channel
Moon stage:	new

Figure 36: Example of the output figures from the ETC. Here with the new moon option as an example.



Exposure Time (single exposure): 1200.0 s
Number of exposures: 3 exposures
Slit Width: 1.00 arcsec
FWHM of PSF: 0.01 arcsec
Object Type: powerlaw
Spectral Index: 0.10
AB Magnitude: 20.25
at wavelength: 650.0 A
Detector binning along dispersion: 1 pixels
Detector binning along slit: 1 pixels
Signal-to-Noise binning factor: 1 binned pixels/channel
Moon stage: half

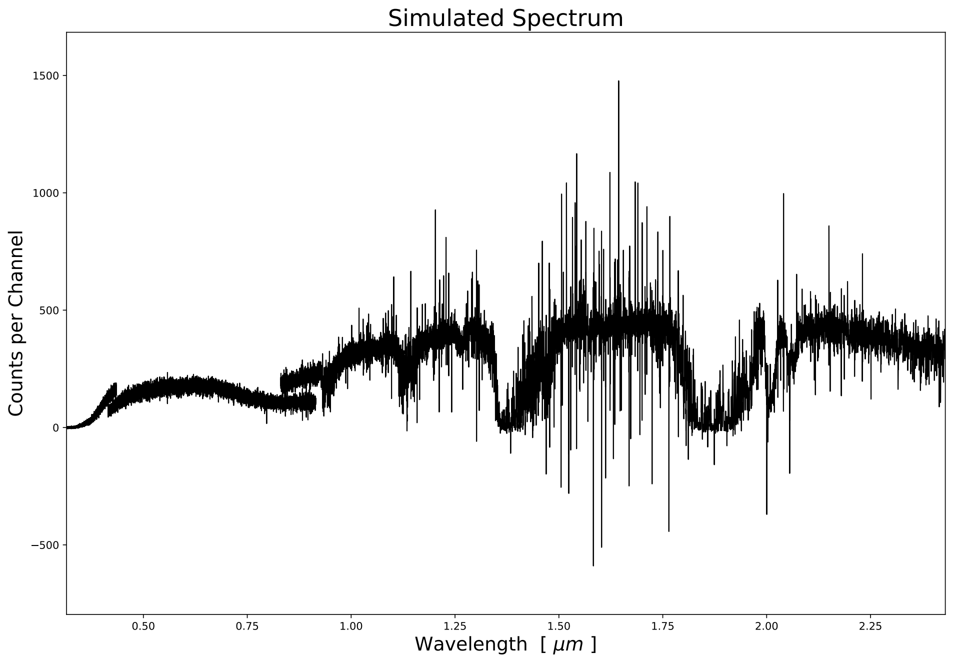
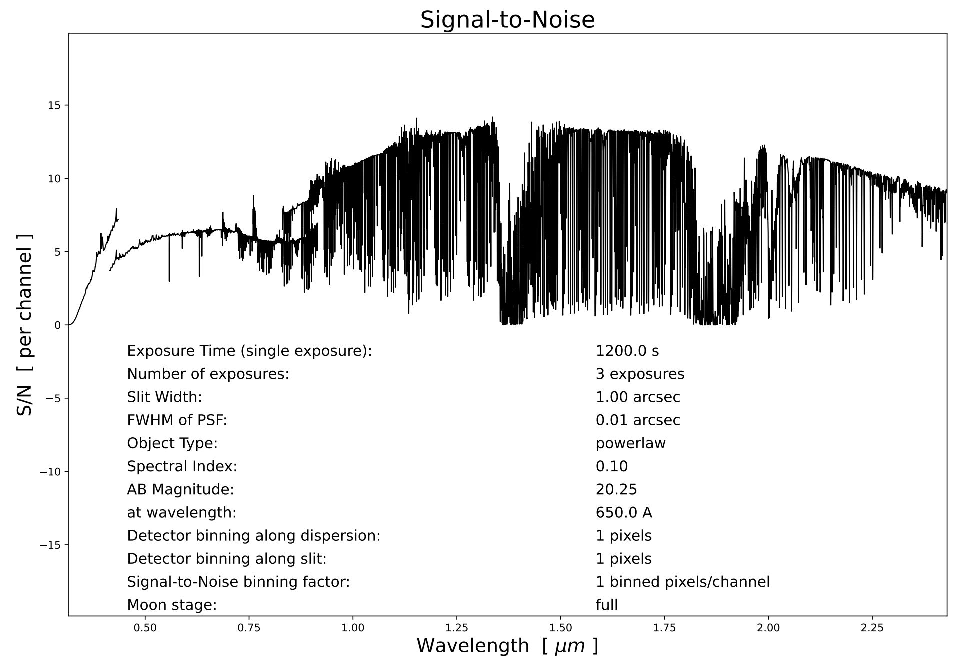


Figure 37: Example of the output figures from the ETC. Here with the half-moon option as an example.



Exposure Time (single exposure):	1200.0 s
Number of exposures:	3 exposures
Slit Width:	1.00 arcsec
FWHM of PSF:	0.01 arcsec
Object Type:	powerlaw
Spectral Index:	0.10
AB Magnitude:	20.25
at wavelength:	650.0 A
Detector binning along dispersion:	1 pixels
Detector binning along slit:	1 pixels
Signal-to-Noise binning factor:	1 binned pixels/channel
Moon stage:	full

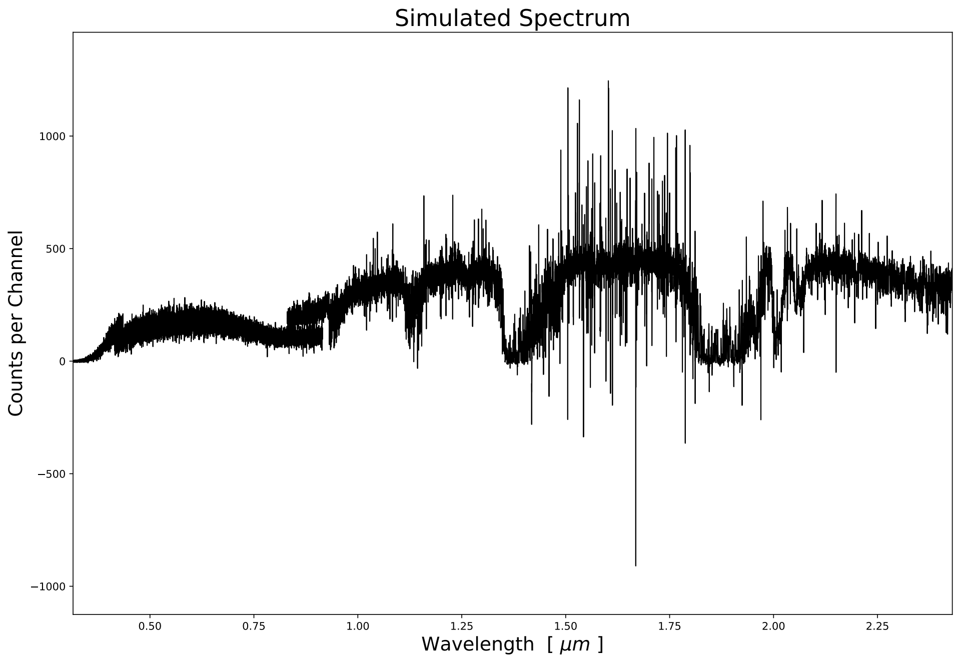


Figure 38: Example of the output figures from the ETC. Here with the full moon option as an example.

A.3 PipeIt

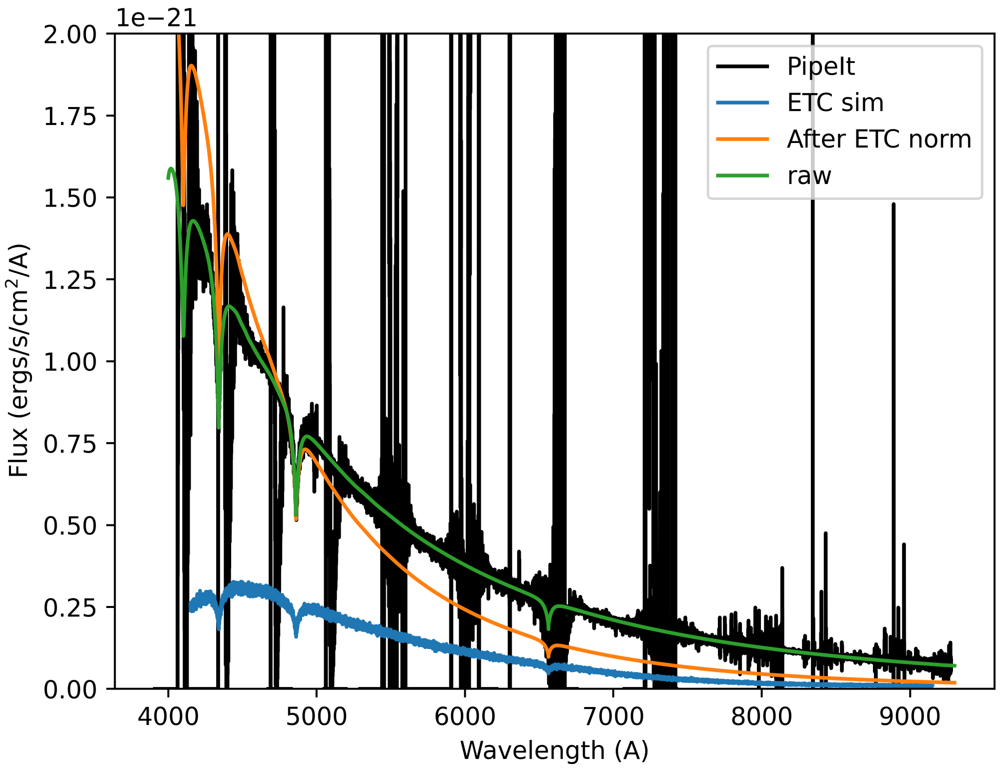


Figure 39: Flux as a function of wavelength for GD71. Black: Extracted spectrum from PipeIt after it was calibrated with a standard star. Blue: The output spectrum from the ETC. Orange: Spectrum input into the ETC. It is the GD71 spectrum normalized to an AB magnitude of 12.9 at 5000A. Green: Raw spectrum from GD71. The PipeIt and raw data have both been scaled with 2.4e-8.

A.4 GitHub with the code

ETC: `https://github.com/ceci8154/NTE_ETC.git`

[29]

SimData: `https://github.com/ceci8154/NTE_SimulatedData.git`

[35]