engine Documentation

Release

Natasha Batalha;Kevin Stevenson

Jan 30, 2019

CONTENTS

Tools to help the community with planning exoplanet observations.

Note: The online PandExo module has moved to STScI's ExoCTK.

PandExo is both an online tool and a python package for generating instrument simulations of JWST's NIRSpec, NIRCam, NIRISS and NIRCam and HST WFC3. It uses throughput calculations from STScI's Exposure Time Calculator, Pandeia: Pandeia + Exoplanets = PandExo. This documentation contains information on how to download, install and analyze PandExo output.

Citation: PandExo: A Community Tool for Transiting Exoplanet Science with JWST and HST

Contents:

CHAPTER

ONE

GETTING STARTED

Tools to help the community with planning exoplanet observations.

PandExo is both an online tool and a python package for generating instrument simulations of JWST's NIRSpec, NIRCam, NIRISS and NIRCam and HST WFC3. It uses throughput calculations from STScI's Exposure Time Calculator, Pandeia: Pandeia + Exoplanets = PandExo. This documentation contains information on how to download, install and analyze PandExo output.

1.1 Should I install PandExo or use the online interface?

Install if...

- I will be using PandExo for more than 10 runs
- I want to submit bash runs
- I want to use the online GUI but don't want to be subject to any slow downs because of high user frequency
- I want to be able to use plotting functions to analyze my output

Install EVEN if...

- I am scared of Python

Do not install if...

- I will only be using PandExo fewer than 10 times
- I have a student who can install it for me

1.2 Requires

- Python >2.7, that's it

Warning: Before reading further, if you do not wish to install PandExo,there is an online version of the code here at STScI's ExoCTK.

CHAPTER

TWO

PRE-INSTALLATION DATA DOWNLOAD

PandExo requires: JWST instrument info and stellar SEDs. Users must set up these two environment variables before proceeding.

2.1 JWST Reference Data

For JWST, the link to access the required reference data is located here. It is important to make sure that the version number of *PandExo* matches the version number of your reference data (currently V1.3).

Then, create environment variable:

```
echo 'export pandeia_refdata="$USRDIR/pandeia_data"' >>~/.bash_profile
```

2.2 Stellar SEDs

Likewise, the user may wish to install specific data for use with the PySynPhot package Pandeia uses. We will only be using the Phoenix stellar atlas, which can be downloaded here.

More reference files can be downloaded via ftp:

- ftp archive.stsci.edu
- username "anonymous"
- password is your e-mail address
- cd pub/hst/pysynphot
- download desired files.

Note that the tar.gz files downloaded from the STScI anonymous FTP site will untar into the directory structure "grp/hst/cdbs", with the actual data files in an assortment of directories under "cdbs". pysynphot (and pandeia) expect that the "PYSYN_CDBS" environment variable will point to the "cdbs" directory. As such, you can either move the files out of "grp/hst/" to wherever you would like to store them, or point "PYSYN_CDBS" to "/path/to/data/files/grp/hst/cdbs" in order to allow pysynphot and pandeia to properly detect the reference files.

Finally, create your environment variable:

```
echo 'export PYSYN_CDBS="$USRDIR/path/to/data/files/grp/hst/cdbs"' >>~/.bash_profile
```

CHAPTER

THREE

INSTALLATION WITH PIP OR GIT

Install STScI specific packages

```
conda config --add channels http://ssb.stsci.edu/astroconda
conda install pyfftw
```

Now, install *pandexo*.

```
pip install pandexo.engine
```

OR Download PandExo's repository via Github. The Github also has helpful notebooks for getting started!

```
git clone --recursive https://github.com/natashabatalha/pandexo
cd pandexo
python setup.py install
```

CHAPTER

FOUR

FINAL TEST FOR SUCCESS

There is a *run_test.py* in the *github*. Test that you're code is working:

```
python run_test.py
Starting TEST run
Running Single Case for: NIRSpec G140H
Optimization Reqested: Computing Duty Cycle
Finished Duty Cycle Calc
Starting Out of Transit Simulation
End out of Transit
Starting In Transit Simulation
End In Transit
SUCCESS
```

CHAPTER

FIVE

TROUBLESHOOTING-COMMON ERRORS

5.1 PyFFTW

PyFFTW is needed to run PandExo. In order to run PyFFTW you need to also isntall fftw. To do so, it is necessary to do so through Homebrew, if you do not have conda.

```
brew install fftw
pip install pyfftw
```

5.2 Multiprocessing

Python 2.7 users might need to install multiprocessing

```
pip install multiprocessing
```

5.3 RecursionError: maximum recursion depth exceeded while calling a Python object

There is a known bug with Python 3.6 and Sphinx <1.6. Before updating or installing pandexo do the following:

PIP USERS:

```
pip install sphinx==1.5.6
```

CONDA USERS:

```
conda install sphinx=1.5.6
```

5.4 TypeError: super() argument 1 must be type

This is the same error above with Sphinx, but for Python 2.7 users. The fix is the same:

PIP USERS:

```
pip install sphinx==1.5.6
```

CONDA USERS:

```
conda install sphinx=1.5.6
```

CHAPTER

SIX

THE IMPORTANCE OF UPGRADING PANDEXO

It is crucial that your verison of PandExo remain up to date. Especially through commissioning and leading up to launch, there may be crucial changes to the code or the reference data. Updating PandExo requires three crucial steps.

6.1 Verify Reference Data is Current

The link to the reference data is located on Pandeia's PyPI page. Before doing a large batch of calculations, make sure that you have this version.

6.2 Verify pandeia.engine is Current

```
pip install pandeia.engine --upgrade
```

6.3 Verify pandexo.engine is Current

```
pip install pandexo.engine --upgrade
```

CHAPTER

SEVEN

WINDOWS INSTALLING GUIDE FOR PANDEXO

Created by J. N. van Haastere (Physics and Astronomy Bachelorstudent) Univesity of Amsterdam and VU University Amsterdam

7.1 Step 1

You may want to uninstall older python versions to prevent troubles along the way.

7.2 Step 2

Download most recent Anaconda version (tested with Python 3.6.4 64 bit)

7.3 Step 3

Install Anaconda as main python and add to path environment

Note: If succesfully installed you can open CMD and type >>*python* and >>*quit()* to check version

7.4 Step 4

Install Build Tools for Visual Studio 2017.

7.5 Step 5

Follow PandExo Installation guide in getting files

- get pandeia_data (pandeia_data-1.2.tar.gz)
- get pysynphot_data (synphot5.tar.gz)
- get PandExo github folder (pandexo-master.zip)

7.6 Step 6

Open pandexo.sh in your favorite text editor (eg. Notepad++) and read through installation steps. Below we will follow these steps but with Windows commands.

7.7 Step 7: Setting Reference Data

Using an achiver (eg. Winrar, 7Zip) extract Pandeia and Pysynphot data in two seperate folders.

```
C:/Users/USERNAME/pandeia_data
C:/Users/USERNAME/pysynphot_data
```

Check the variable names

```
echo 'export VARIABLE_NAME = ".."'
```

In case nothing changes these should be *PYSYN_CDBS* & *pandeia_refdata*

Set these in Windows environment variables / registry using SETX. Open CMD or Anaconda Prompt and run:

```
SETX PYSYN_CDBS 'C:/Users/USERNAME/pysynphot_data'
SETX pandeia_refdata 'C:/Users/USERNAME/pysynphot_data'
```

You can check this step by opening a new CMD and run > SET

7.8 Step 7: Installing needed Packages

Add conda channel, in CMD:

```
conda config --add channels http://ssb.stsci.edu/astroconda
```

Check needed packages, at the moment:

```
numpy synphot joblib scipy astropy==2.0.2 pyfftw pysynphot photutils sphinx=1.5.6␣
↪bokeh=0.12.6
```

All but *Pyfftw* can be installed using:

```
conda install PACKAGE
```

or

```
pip install PACKAGE
```

If they fail check in the error if you are missing a certain .dll or look online.

For *Pyfftw* get a prebuiled wheel from here.

Get a matching cp36-cp36 for your python version. For Python 3.6.4 64-bit I used:

- *pyFFTW-0.10.5.dev0+d45d7fe-cp36-cp36m-win_amd64.whl*

Navigate to download folder and install

```
cd 'C:\Users\USERNAME\Downloads'
pip install pyFFTW-0.10.5.dev0+d45d7fe-cp36-cp36m-win_amd64.whl
```

With your fingers crossed, install PandExo Engine

```
pip install pandexo.engine
```

Probabily fail, look what I/you missed and try again. Troubleshooting here.

7.9 Step 8: Run Test

Navigate to your pandexo-master and run the run_test. Check in the installation guide on the exact expected output.

```
cd '...'
python run_test.py
```

Congrats on being persevering/stubborn enough to get it on to work on Windows! There are some test Jupyter files to try out in *pandexo-master/notebooks*

CHAPTER

EIGHT

JWST TUTORIAL

Here you will learn how to:

- set planet properties
- set stellar properties
- run default instrument modes
- adjust instrument modes
- run pandexo

```
import warnings
warnings.filterwarnings('ignore')
import pandexo.engine.justdoit as jdi # THIS IS THE HOLY GRAIL OF PANDEXO
import numpy as np
import os
#pip install pandexo.engine --upgrade
```

8.1 Setting up a run

To start, load in a blank exoplanet dictionary with empty keys. You will fill these out for yourself in the next step.

```
exo_dict = jdi.load_exo_dict()
print(exo_dict.keys())
#print(exo_dict['star']['w_unit'])
```

8.1.1 Edit exoplanet observation inputs

Editting each keys are annoying. But, do this carefully or it could result in nonsense runs

```
exo_dict['observation']['sat_level'] = 80    #saturation level in percent of full well
exo_dict['observation']['sat_unit'] = '%'
exo_dict['observation']['noccultations'] = 2 #number of transits
exo_dict['observation']['R'] = None           #fixed binning. I usually suggest ZERO␣
↪binning.. you can always bin later
                                             #without having to redo the calcualtion
exo_dict['observation']['baseline_unit'] = 'total'  #Defines how you specify out of␣
↪transit observing time
                                                    #'frac' : fraction of time in␣
↪transit versus out = in/out
                                                    #'total' : total observing time␣
↪(seconds)
```

```
exo_dict['observation']['baseline'] = 4.0*60.0*60.0 #in accordance with what was␣
↪specified above (total observing time)

exo_dict['observation']['noise_floor'] = 0    #this can be a fixed level or it can be␣
↪a filepath
                                              #to a wavelength dependent noise floor␣
↪solution (units are ppm)
```

8.1.2 Edit exoplanet host star inputs

Note... If you select 'phoenix' you **do not** have to provide a starpath, w_unit or f_unit, but you **do** have to provide a temp, metal and logg. If you select 'user' you **do not** need to provide a temp, metal and logg, but you **do** need to provide units and starpath.

```
exo_dict['star']['type'] = 'phoenix'          #phoenix or user (if you have your own)
exo_dict['star']['mag'] = 8.0                 #magnitude of the system
exo_dict['star']['ref_wave'] = 1.25           #For J mag = 1.25, H = 1.6, K =2.22.. etc␣
↪(all in micron)
exo_dict['star']['temp'] = 5500               #in K
exo_dict['star']['metal'] = 0.0               # as log Fe/H
exo_dict['star']['logg'] = 4.0                #log surface gravity cgs
```

8.1.3 Edit exoplanet inputs using one of three options

1. user specified
2. constant value
3. select from grid

1) Edit exoplanet planet inputs if using your own model

```
exo_dict['planet']['type'] ='user'                          #tells pandexo you are␣
↪uploading your own spectrum
exo_dict['planet']['exopath'] = 'wasp12b.txt'
exo_dict['planet']['w_unit'] = 'cm'                         #other options include "um",
↪"nm" ,"Angs", "sec" (for phase curves)
exo_dict['planet']['f_unit'] = 'rp^2/r*^2'                  #other options are 'fp/f*'
exo_dict['planet']['transit_duration'] = 2.0*60.0*60.0      #transit duration
exo_dict['planet']['td_unit'] = 's'                         #Any unit of time in␣
↪accordance with astropy.units can be added
```

2) Users can also add in a constant temperature or a constant transit depth

```
exo_dict['planet']['type'] = 'constant'                     #tells pandexo you want a␣
↪fixed transit depth
exo_dict['planet']['transit_duration'] = 2.0*60.0*60.0      #transit duration
exo_dict['planet']['td_unit'] = 's'
exo_dict['planet']['radius'] = 1
exo_dict['planet']['r_unit'] = 'R_jup'              #Any unit of distance in accordance␣
↪with astropy.units can be added here
exo_dict['star']['radius'] = 1
```

```
exo_dict['star']['r_unit'] = 'R_sun'                #Same deal with astropy.units here
exo_dict['planet']['f_unit'] = 'rp^2/r*^2'          #this is what you would do for␣
↪primary transit

#ORRRRR....
#if you wanted to instead to secondary transit at constant temperature
exo_dict['planet']['f_unit'] = 'fp/f*'
exo_dict['planet']['temp'] = 1000
```

3) Select from grid

NOTE: Currently only the fortney grid for hot Jupiters from Fortney+2010 is supported. Holler though, if you want another grid supported

```
exo_dict['planet']['type'] = 'grid'                 #tells pandexo you want to pull␣
↪from the grid
exo_dict['planet']['temp'] = 1000                   #grid: 500, 750, 1000, 1250, 1500,␣
↪1750, 2000, 2250, 2500
exo_dict['planet']['chem'] = 'noTiO'                #options: 'noTiO' and 'eqchem',␣
↪noTiO is chemical eq. without TiO
exo_dict['planet']['cloud'] = 'ray10'                 #options: nothing: '0',
#                                                     Weak, medium, strong scattering:␣
↪ray10,ray100, ray1000
#                                                     Weak, medium, strong cloud: flat1,
↪flat10, flat100
exo_dict['planet']['mass'] = 1
exo_dict['planet']['m_unit'] = 'M_jup'              #Any unit of mass in accordance␣
↪with astropy.units can be added here
exo_dict['planet']['radius'] = 1
exo_dict['planet']['r_unit'] = 'R_jup'              #Any unit of distance in accordance␣
↪with astropy.units can be added here
exo_dict['star']['radius'] = 1
exo_dict['star']['r_unit'] = 'R_sun'                #Same deal with astropy.units here
exo_dict['planet']['transit_duration'] = 2.0*60.0*60.0   #transit duration
exo_dict['planet']['td_unit'] = 's'
```

8.2 Load in instrument dictionary (OPTIONAL)

Step 2 is optional because PandExo has the functionality to automatically load in instrument dictionaries. Skip this if you plan on observing with one of the following and want to use the subarray with the smallest frame time and the readout mode with 1 frame/1 group (standard): - NIRCam F444W - NIRSpec Prism - NIRSpec G395M - NIRSpec G395H - NIRSpec G235H - NIRSpec G235M - NIRCam F322W - NIRSpec G140M - NIRSpec G140H - MIRI LRS - NIRISS SOSS

```
#jdi.print_instruments()
result = jdi.run_pandexo(exo_dict,['NIRCam F322W2'])
```

```
inst_dict = jdi.load_mode_dict('NIRSpec G140H')

#loading in instrument dictionaries allow you to personalize some of
#the fields that are predefined in the templates. The templates have
#the subbarays with the lowest frame times and the readmodes with 1 frame per group.
#if that is not what you want. change these fields
```

```
#Try printing this out to get a feel for how it is structured:

print(inst_dict['configuration'])
```

```
#Another way to display this is to print out the keys
inst_dict.keys()
```

8.2.1 Don't know what instrument options there are?

```
print("SUBARRAYS")
print(jdi.subarrays('nirspec'))

print("FILTERS")
print(jdi.filters('nircam'))

print("DISPERSERS")
print(jdi.dispersers('nirspec'))
```

```
#you can try personalizing some of these fields

inst_dict["configuration"]["detector"]["ngroup"] = 'optimize'    #running "optimize"␣
↪will select the maximum
                                                                 #possible groups␣
↪before saturation.
                                                                 #You can also write␣
↪in any integer between 2-65536

inst_dict["configuration"]["detector"]["subarray"] = 'substrip256'    #change the␣
↪subbaray
```

8.2.2 Adjusting the Background Level

You may want to think about adjusting the background level of your observation, based on the position of your target. PandExo two options and three levels for the position:

- `ecliptic` or `minzodi`
- `low`, `medium`, `high`

```
inst_dict['background'] = 'ecliptic'
inst_dict['background_level'] = 'high'
```

8.2.3 Running NIRISS SOSS Order 2

PandExo only will extract a single order at a time. By default, it is set to extract Order 1. Below you can see how to extract the second order.

NOTE! Users should be careful with this calculation. Saturation will be limited by the **first** order. Therefore, I suggest running one calculation with `ngroup='optmize'` for Order 1. This will give you an idea of a good number of groups to use. Then, you can use that in this order 2 calculation.

```
inst_dict = jdi.load_mode_dict('NIRISS SOSS')
inst_dict['strategy']['order'] = 2
inst_dict['configuration']['detector']['subarray'] = 'substrip256'
ngroup_from_order1_run = 2
inst_dict["configuration"]["detector"]["ngroup"] = ngroup_from_order1_run
```

8.3 Running PandExo

You have **four options** for running PandExo. All of them are accessed through attribute **jdi.run_pandexo**. See examples below.

```
jdi.run_pandexo(exo, inst, param_space = 0, param_range = 0,save_file = True,
output_path=os.getcwd(), output_file = '')
```

8.3.1 Option 1- Run single instrument mode, single planet

If you forget which instruments are available run **jdi.print_isntruments()** and pick one

```
jdi.print_instruments()
```

```
result = jdi.run_pandexo(exo_dict,['NIRCam F322W2'])
```

8.3.2 Option 2- Run single instrument mode (with user dict), single planet

This is the same thing as option 1 but instead of feeding it a list of keys, you can feed it a instrument dictionary (this is for users who wanted to simulate something NOT pre defined within pandexo)

```
inst_dict = jdi.load_mode_dict('NIRSpec G140H')
#personalize subarray
inst_dict["configuration"]["detector"]["subarray"] = 'sub2048'
result = jdi.run_pandexo(exo_dict, inst_dict)
```

8.3.3 Option 3- Run several modes, single planet

Use several modes from **print_isntruments()** options.

```
#choose select
result = jdi.run_pandexo(exo_dict,['NIRSpec G140M','NIRSpec G235M','NIRSpec G395M'],
               output_file='three_nirspec_modes.p')
#run all
#result = jdi.run_pandexo(exo_dict, ['RUN ALL'], save_file = False)
```

8.3.4 Option 4- Run single mode, several planet cases

Use a single modes from **print_isntruments()** options. But explore parameter space with respect to **any** parameter in the exo dict. The example below shows how to loop over several planet models

You can loop through anything in the exoplanet dictionary. It will be planet, star or observation followed by whatever you want to loop through in that set.

i.e. planet+exopath, star+temp, star+metal, star+logg, observation+sat_level.. etc

```
#looping over different exoplanet models
jdi.run_pandexo(exo_dict, ['NIRCam F444W'], param_space = 'planet+exopath',
                param_range = os.listdir('/path/to/location/of/models'),
               output_path = '/path/to/output/simulations')

#looping over different stellar temperatures
jdi.run_pandexo(exo_dict, ['NIRCam F444W'], param_space = 'star+temp',
                param_range = np.linspace(5000,8000,2),
               output_path = '/path/to/output/simulations')

#looping over different saturation levels
jdi.run_pandexo(exo_dict, ['NIRCam F444W'], param_space = 'observation+sat_level',
                param_range = np.linspace(.5,1,5),
               output_path = '/path/to/output/simulations')
```

8.4 Running PandExo GUI

The same interface that is available online is also available for use on your machine. Using the GUI is very simple and good alternative if editing the input dictionaries is confusing.

```
start_pandexo
```

Then open up your favorite internet browser and go to: http://localhost:1111

8.5 Analyzing Output

There are pre computed functions for analyzing most common outputs. You can also explore the dictionary structure yourself.

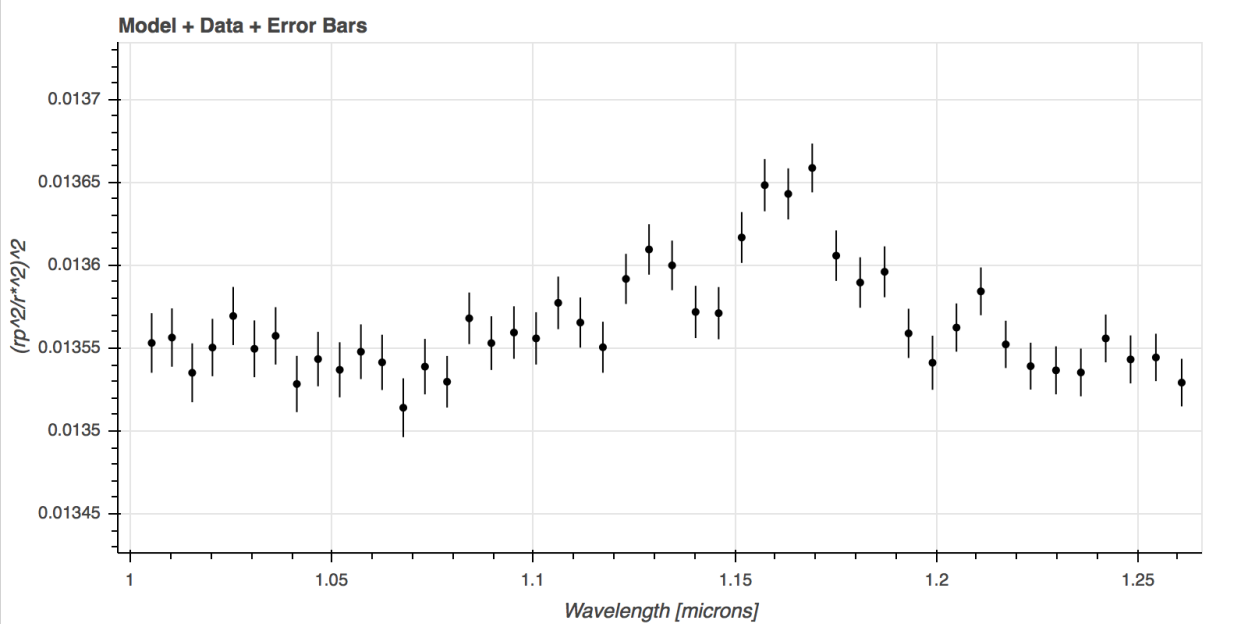
```
import pandexo.engine.justplotit as jpi
import pickle as pk
```

8.5.1 Plot 1D Data with Errorbars

Multiple plotting options exist within *jwst_1d_spec*

1. Plot a single run

```
#load in output from run
out = pk.load(open('singlerun.p','r'))
#for a single run
x,y, e = jpi.jwst_1d_spec(out, R=100, num_tran=10, model=False, x_range=[.8,1.28])
```

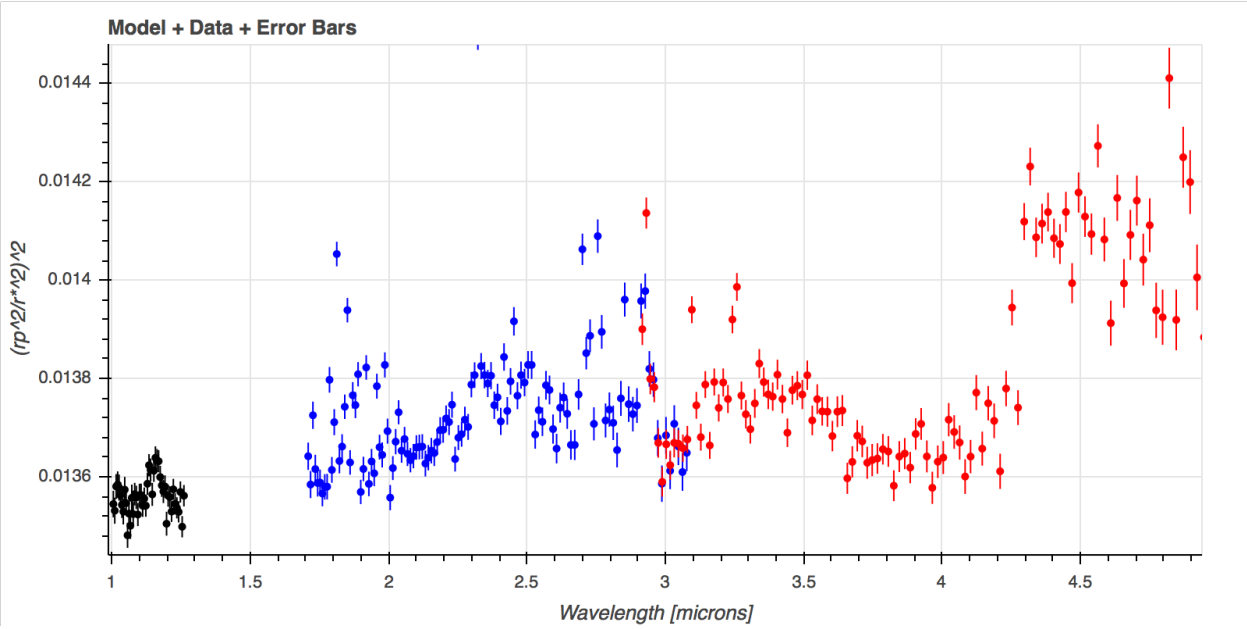


2. Plot several runs from parameters space run

```
#load in output from multiple runs
multi = pk.load(open('three_nirspec_modes.p','r'))

#get into list format
list_multi = [multi[0]['NIRSpec G140M'], multi[1]['NIRSpec G235M'], multi[2]['NIRSpec␣
↪G395M']]

x,y,e = jpi.jwst_1d_spec(list_multi, R=100, model=False, x_range=[1,5])
```

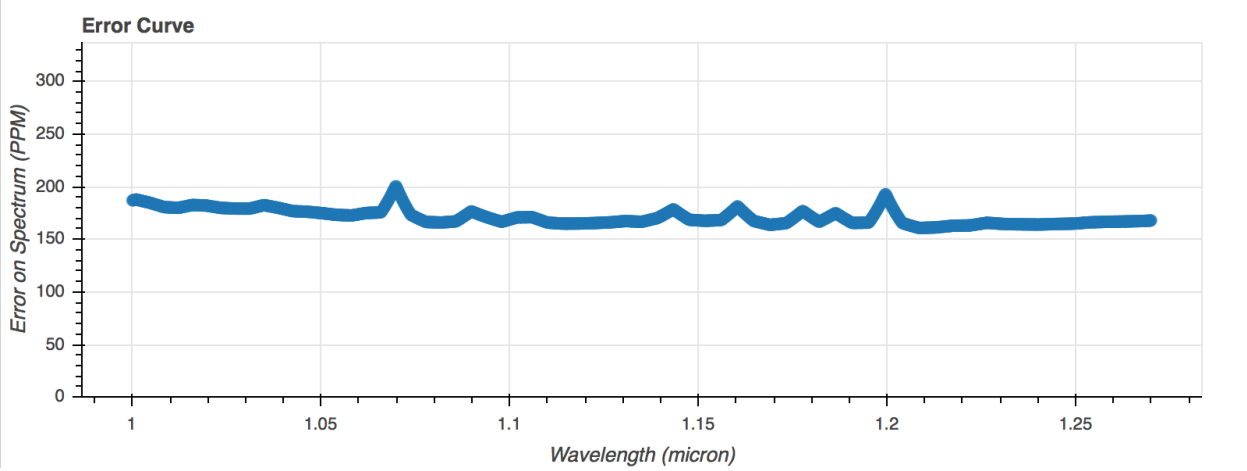


8.5.2 Plot Noise & More

Several functions exist to plot various outputs.

See also **jwst_1d_bkg**, **jwst_1d_snr**, **jwst_1d_flux**,

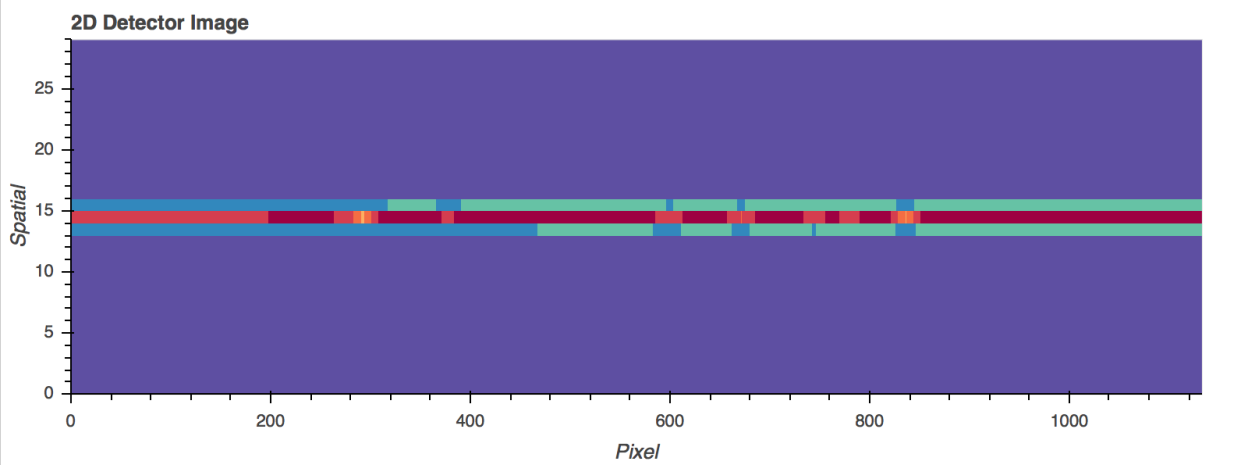
```
x,y = jpi.jwst_noise(out)
```



8.5.3 Plot 2D Detector Profile

See also **jwst_2d_sat** to plot saturation profile

```
data = jpi.jwst_2d_det(out)
```



CHAPTER

NINE

POSSIBLE INSTRUMENT INPUT PARAMS

Below are all the possible instrument input parameters, which match the JWST APT (downloadable here).

9.1 NIRSpec

9.1.1 Grisms/Filter combinations

9.1.2 Subarrays

9.2 NIRISS

9.3 NIRCam

9.4 MIRI

CHAPTER

TEN

PY DICT STRUCTURE OF JWST OUTPUT

The PandExo output is organized into Python Dictionaries. See example notebooks for an explanation of how to manipulate these and plot the most common outputs. Below is a breakdown of everything contained in the PandExo output dictionary.

10.1 FinalSpectrum (contains 4 keys)

- **spectrum_w_rand**: Planet spectrum with random noise added in units of *(Rp/Rs)2* or *(Fp/Fs)*
- **spectrum**: Planet spectrum with no random noise
- **error_w_floor**: Error with user defined noise floor. If no floor was specified, there is no floor.
- **wave**: wavelength (microns)

```
print dict['FinalSpectrum']['spectrum_w_rand']
```

10.2 OriginalInput (contains 2 keys)

- **model_wave**: original wavelength input by user
- **model_spec**: original spectrum input by user

```
print dict['OriginalInput']['model_wave']
```

10.3 Warning (contains 6 keys)

- **Num Groups Reset?**: Before PandExo simulates in and out of transit observations, it computes a single integration with 2 groups in order to figure out how many additional groups it can add before hitting saturation. If it computes a number less than 2, it resets the number of groups to 2.
- **Group Number Too Low?**: Prints out warning if number of groups is less than 5 and the saturation level less than 60%.
- **Group Number Too High?**: Prints out warning if number of groups per integration exceeds 65536
- **Saturated?**: This is an output directly taken from Pandeia's "hard saturation" flag. If there are any saturated pixels, it will alert you here. You can also see the saturation profile.
- **Non linear?**: This is an output directly taken from Pandeia's "soft saturation" flag.

- **% full well high?**: If you've set the saturation level over 80%, it will warn you.

```
print dict['Warning']['Num Groups Reset?']
```

10.4 PandeiaOutTrans (contains 9 keys)

This is the raw output of Pandeia's simulation of the out of transit observation. For a complete breakdown of these outputs go to STScI's Pandeia Documentation.

- **sub_reports**
- **information**
- **warnings**
- **transform**
- **2d**
- **scalar**
- **1d**
- **input**
- **3d**

```
print dict['PandeiaOutTrans']['information']
```

10.5 RawData (contains 6 keys)

- **var_in**: The variance of only the in transit data
- **wave**: wavelength vector
- **flux_in**: Flux of the in transit data in units of e-/s
- **flux_out**: Flux of the out of transit data in units of e-/s
- **error_no_floor**: Error without any noise floor
- **var_out**: The variance of only the out of transit data

```
print dict['RawData']['var_in']
```

10.6 Timing (contains 10 keys)

- **Seconds per Frame**
- **Number of Transits**
- **Observing Efficiency (%)** = (num groups - 1)/(num groups + 1)
- **Num Integrations Out of Transit**
- **On Source Time**
- **Exposure Time Per Integration (secs)**

- **Reset time Plus TA time (hrs)**: Target acquisition time is assumed to be 30 minutes
- **Num Integrations In Transit**
- **Num Groups per Integration**
- **Num Integrations per Occultation**

```
print dict['Timing']['Seconds per Frame']
```

10.7 Input (contains 10 keys)

- **Target Mag**
- **Readmode**
- **Disperser**
- **Filter**
- **Instrument**
- **Mode**
- **Saturation Level (electons)**
- **Aperture**
- **Subarray**
- **Primary/Secondary**

```
print dict['Input']['Target Mag']
```

10.8 Timing, Warning, and Input Divs (all contain html script)

Here are the html scripts for the three tables that are rendered on the website output page.

CHAPTER

ELEVEN

HST TUTORIAL

This file demonstrates how to use TExoNS to predict the: 1. Transmission/emission spectrum S/N ratio 2. Observation start window for any system observed with WFC3/IR.

```
import pandexo.engine.justdoit as jdi
```

11.1 Editting Input Dictionaries

11.1.1 Step 1) Load in a blank exoplanet dictionary

```
exo_dict = jdi.load_exo_dict()
```

Edit stellar and planet inputs

```
#WASP-43
exo_dict['star']['mag']      = 9.397                # H magnitude of the system
#WASP-43b
exo_dict['planet']['type']    = 'user'               # user specified inputs
exo_dict['planet']['exopath'] = 'WASP43b-Eclipse_Spectrum.txt' # filename for model␣
↪spectrum
exo_dict['planet']['w_unit']  = 'um'                 # wavelength unit
exo_dict['planet']['f_unit']  = 'fp/f*'              # flux ratio unit (can also put
↪"rp^2/r*^2")
exo_dict['planet']['depth']   = 4.0e-3               # flux ratio
exo_dict['planet']['i']       = 82.6                 # Orbital inclination in degrees
exo_dict['planet']['ars']     = 5.13                 # Semi-major axis / stellar␣
↪radius
exo_dict['planet']['period']  = 0.8135               # Orbital period in days
exo_dict['planet']['transit_duration']= 4170.0/60/60/24#(optional if given above␣
↪info) transit duration in days
exo_dict['planet']['w']       = 90                   #(optional) longitude of␣
↪periastron. Default is 90
exo_dict['planet']['ecc']     = 0                    #(optional) eccentricity.␣
↪Default is 0
```

11.1.2 Step 2) Load in instrument dictionary

- WFC3 G141

- WFC3 G102

```
inst_dict = jdi.load_mode_dict('WFC3 G141')
```

Edit HST/WFC3 detector and observation inputs

```
exo_dict['observation']['noccultations']                = 5             # Number of␣
↪transits/eclipses
inst_dict['configuration']['detector']['subarray']      = 'GRISM256'    # GRISM256 or␣
↪GRISM512
inst_dict['configuration']['detector']['nsamp']         = 10            # WFC3 N_SAMP,␣
↪1..15
inst_dict['configuration']['detector']['samp_seq']      = 'SPARS5'      # WFC3 SAMP_SEQ,
↪ SPARS5 or SPARS10
inst_dict['strategy']['norbits']                        = 4             # Number of HST␣
↪orbits
inst_dict['strategy']['nchan']                        = 15          # Number of␣
↪spectrophotometric channels
inst_dict['strategy']['scanDirection']                = 'Forward'     # Spatial scan␣
↪direction, Forward or Round Trip
inst_dict['strategy']['schedulability']               = 30          # 30 for small/
↪medium program, 100 for large program
inst_dict['strategy']['windowSize']                   = 20          # (optional)␣
↪Observation start window size in minutes. Default is 20 minutes.
```

11.2 Run PandExo Command Line

`jdi.run_pandexo(exo, inst, param_space = 0, param_range = 0,save_file = True, output_path=os.getcwd(), output_file = '')`

See wiki Attributes for more thorough explanation fo inputs

```
foo = jdi.run_pandexo(exo_dict, inst_dict, output_file='wasp43b.p')
Running Single Case w/ User Instrument Dict
****WARNING: Observing plan may incur mid-orbit buffer dumps.  Check with APT.
```

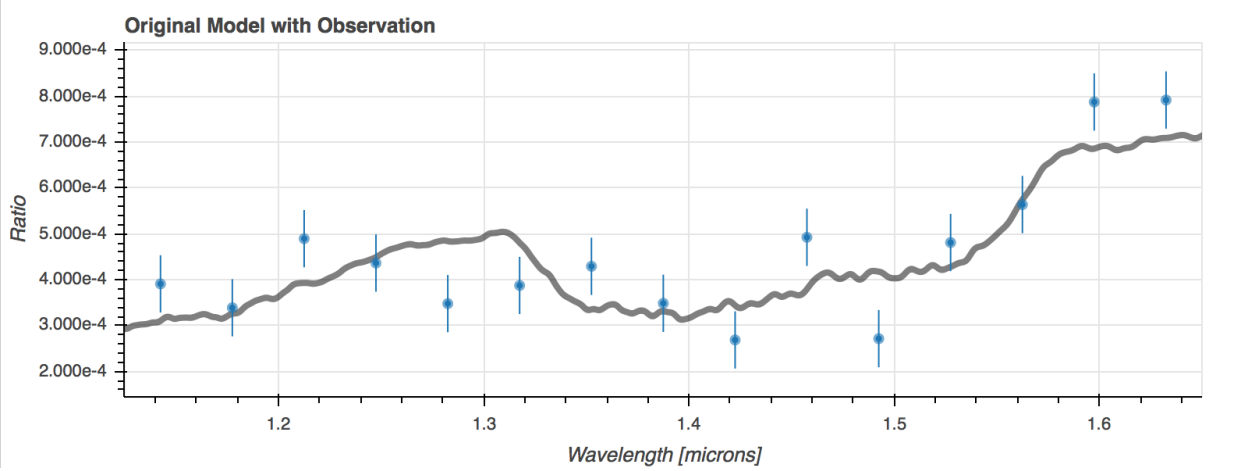
```
inst_dict['configuration']['detector']['nsamp'] = None
inst_dict['configuration']['detector']['samp_seq'] = None
bar = jdi.run_pandexo(exo_dict, inst_dict, output_file='wasp43b.p')
Running Single Case w/ User Instrument Dict
```

```
exo_dict['observation']['scanDirection'] = 'Round Trip'
hst = jdi.run_pandexo(exo_dict, inst_dict, output_file='wasp43b.p')
Running Single Case w/ User Instrument Dict
```

11.3 Plot Results

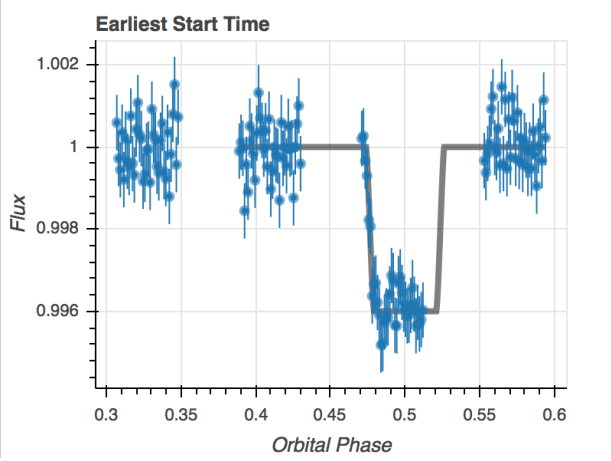
11.3.1 Plot simulated spectrum using specified file

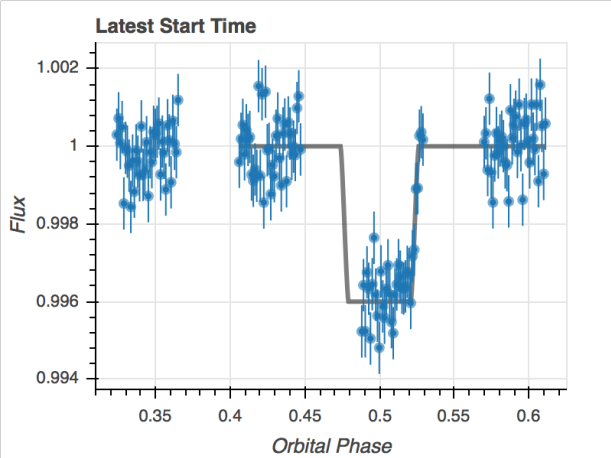
```
import pandexo.engine.justplotit as jpi
#using foo from above
#other keys include model=True/False
datawave, dataspec, dataerror, modelwave, modelspec = jpi.hst_spec(foo)
```



11.3.2 Compute earliest and latest start times for given start window size

```
#using foo from above
obsphase1, obstr1, obsphase2, obstr2,rms = jpi.hst_time(foo)
```





Print important info for observation

```
foo['wfc3_TExoNS']['info']
{'Estimated duty cycle (outside of Earth occultation)': 24.991166666666668,
 'Maximum pixel fluence (electrons)': 30266.370139081948,
 'Number of HST orbits': 4,
 'Number of Transits': 5,
 'Number of channels': 15,
 'Recommended scan rate (arcsec/s)': 0.07599999999999998,
 'Scan height (pixels)': 13.454512396694216,
 'Start observations between orbital phases': '0.307075052926-0.324148057092',
 'Transit depth uncertainty(ppm)': 62.433045276228441,
 'WFC3 parameters: NSAMP': 10,
 'WFC3 parameters: SAMP_SEQ': 'SPARS5'}
```

CHAPTER

TWELVE

THE CODE

12.1 engine.justdoit

engine.justdoit.**dispersers**(*inst*)
function to show available dispersers

Parameters **inst** (*str*) – string of either niriss, nirspec, miri or nircam

Returns lsit with available dispersers

Return type list

engine.justdoit.**filters**(*inst*)
Function to show availalbe filters

Parameters **inst** (*str*) – string of either niriss, nirspec, miri or nircam

Returns list with availalbe filters

Return type list

engine.justdoit.**get_thruput**(*inst*, *niriss=1*, *nirspec='f100lp'*)
Returns complete instrument photon to electron conversion efficiency Pulls complete instrument photon to electron conversion efficiency (PCE) based on instrument key input

Parameters

- **inst** (*str*) – One of the instrument keys in *print_instruments*
- **niriss** (*int*) – (Optional) defines which niriss order you want (1 or 2)
- **nirspec** (*str*) – (Optional) for NIRISS G140M/H there are two available filters (f100lp and f070lp) if you are selecting G140M or G140H, this allows you to pick which one

Returns Dictionary with wave solution and PCE

Return type dict

Example

```
>>> thru_dict = get_thruput('NIRISS SOSS_Or1')
```

engine.justdoit.**grid_options**(*grid='fortney'*)
Function to show available grid options

PandExo now supports various grid options. Currently, the only one that is availalbe is the Fortney grid for giant planets. We will be implementing others, as they become available. It will become increasingly difficult for

users to see what options are availalbe to them. This function should guide users to choose the grid that best fits their needs.

Parameters **grid** (*str*) – (Optional) string which 'fortney'

engine.justdoit.**load_exo_dict**()
Loads in empty exoplanet dictionary for pandexo input

Loads in empty exoplanet dictionary so that the user can manually edit different planet parameters. Must be done for every bash run. User must edit each keys within the dictionary.

Returns Empty dictionary to be filled out by the user before running PandExo

Return type dict

Example

```
>>> exo_dict = load_exo_dict()
>>> exo_dict['planet']['transit_duration'] = 2*60*60 #2 hours
```

engine.justdoit.**load_mode_dict**(*inst*)
Function to pull in correct instrument dictionary

This is the instrument counterpart to load_exo_dict. It loads in a template instrument dictionary for a specific instrument mode.

Parameters **inst** (*str*) – One of the allowable instrument keys. To see which instrument modes are available use *print_instruments()*

Returns Filled out template of instrument dictionary, which can be editted before running to PandExo (not required).

Return type dict

Example

```
>>> inst_dict = load_mode_dict('MIRI LRS')
>>> inst_dict['configuration']['instrument']['aperture'] = 'lrsslit'
```

engine.justdoit.**print_instruments**()
Prints a list of the possible instrument templates to load

engine.justdoit.**run_inst_space**(*inst*, *exo*)
Changes inst dictionary and submits run

This function is used to reset the instrument dictionary.

Parameters

- **exo** (*dict*) – Exoplanet dictionary which can be loaded in and editted through *load_exo_dict*
- **inst** (*str*) – Key which indicates with instrument

Returns Dictionary with output of pandexo. Key is the value of the parameter that was looped through.

Return type dict

engine.justdoit.**run_pandexo**(*exo*, *inst*, *param_space=0*, *param_range=0*, *save_file=True*, *output_path='/Users/natashabatalha/Desktop/JWST/pandexo/docs'*, *output_file=''*)

Submits multiple runs of pandexo in parallel.

Functionality: program contains functionality for running single or multiple runs of PandExo

Parameters

- **exo** (*dict*) – exoplanet input dictionary
- **inst** (*dict or str or list of str*) – instrument input dictionary OR LIST of keys (for allowable keys see *print_instruments()*
- **param_space** (*str or 0*) – (Optional) Default is 0 = no exoplanet parameter space. To run through a parameter specify which one need to specify two keys from exo dict with + in between. i.e. observation+fraction star+temp planet+exopath
- **param_range** (*list of str or list of float*) – (Optional) Default = 0 An array or list over which to run the parameters space. i.e. array of temperatures if running through stellar temp or array of files if running through planet models. Must specify param_space if using this.
- **save_file** (*bool*) – (Optional) Default = True saves file, False does not
- **output_path** (*str*) – (Optional) Defaults to current working directory
- **output_file** (*str*) – (Optional) Default is "singlerun.p" for single runs, "param_space.p" for exo parameter runs or "instrument_run.p" for instrument parameter space runs.

Returns For single run output will just be a single PandExo output dictionary https://github.com/natashabatalha/PandExo/wiki/PandExo-Output For multiple runs the output will be organized into a list with each a dictionary named by whatever you are looping through i.e. [{'First temp': PandExoDict}, {'Second temp': PandExoDict}, etc..]

Return type dict

Example

For single run:

```
>>> a = run_pandexo(exo_dict, ['MIRI LRS'])
```

For multiple instruments:

```
>>> a = run_pandexo(exo_dict, ['MIRI LRS','NIRSpec G395H'])
```

Loop through a exoplanet parameter (stellar magnitude):

```
>>> a = run_pandexo(exo_dict, ['NIRSpec G395M'],
        param_space ='star+mag',param_range = np.linspace(6,10,5))
```

engine.justdoit.**run_param_space**(*i*, *exo*, *inst*, *param_space*)

Changes exo dictionary and submits run

This function is used to reset the exo dictionary based on what parameter is being looped over and submits run to *wrapper* so that all the jobs are run in parallel

Parameters

- **i** (*str or float*) – Can be either a str or float based on what you are looping through (str for filenames, float for stellar temps, float for magnitudes, etc)
- **exo** (*dict*) – Exoplanet dictionary which can be loaded in and editted through *load_exo_dict*
- **inst** (*str*) – Key which indicates with instrument
- **param_space** (*str*) – Set of keys within exo_dict to indicate which parameter to loop through. Should be in the format of "first level of dict"+"second level of dict". For example, for stellar temp *param_space* would be "star+temp"

Returns Dictionary with output of pandexo. Key is the value of the parameter that was looped through.

Return type dict

engine.justdoit.**subarrays**(*inst*)
function to show availalble subarrays and their times (in secons)

Parameters **inst** (*str*) – string of either niriss, nirspec, miri or nircam

Returns dictionary with name of subarray as keys and time in seconds as entry

Return type dict

12.2 engine.justplotit

engine.justplotit.**bin_wave_to_R**(*w*, *R*)
Creates new wavelength axis at specified resolution

Parameters

- **w** (*list of float or numpy array of float*) – Wavelength axis to be re-binned
- **R** (*float or int*) – Resolution to bin axis to

Returns New wavelength axis at specified resolution

Return type list of float

Examples

```
>>> newwave = bin_wave_to_R(np.linspace(1,2,1000), 10)
>>> print(len(newwave))
11
```

engine.justplotit.**hst_spec**(*result_dict*, *plot=True*, *output_file='hstspec.html'*, *model=True*)
Plot 1d spec with error bars for hst

Parameters

- **result_dict** (*dict*) – Dictionary from pandexo output.
- **plot** (*bool*) – (Optional) True renders plot, False does not. Default=True
- **model** (*bool*) – (Optional) Plot model under data. Default=True
- **output_file** (*str*) – (Optional) Default = 'hstspec.html'

Returns

- **x** (*numpy array*) – micron
- **y** (*numpy array*) – 1D spec fp/f* or rp^2/r*^2
- **e** (*numpy array*) – 1D rms noise
- **modelx** (*numpy array*) – micron
- **modely** (*numpy array*) – 1D spec fp/f* or rp^2/r*^2

See also:

hst_time()

engine.justplotit.**hst_time**(*result_dict*, *plot=True*, *output_file='hsttime.html'*, *model=True*)

Plot earliest and latest start times for hst observation

Parameters

- **result_dict** (*dict*) – Dictionary from pandexo output.
- **plot** (*bool*) – (Optional) True renders plot, False does not. Default=True
- **model** (*bool*) – (Optional) Plot model under data. Default=True
- **output_file** (*str*) – (Optional) Default = 'hsttime.html'

Returns

- **obsphase1** (*numpy array*) – earliest start time
- **obstr1** (*numpy array*) – white light curve
- **obsphase2** (*numpy array*) – latest start time
- **obstr2** (*numpy array*) – white light curve
- **rms** (*numpy array*) – 1D rms noise

See also:

hst_spec()

engine.justplotit.**jwst_1d_bkg**(*result_dict*, *plot=True*, *output_file='bkg.html'*)

Plot background

Parameters

- **result_dict** (*dict*) – Dictionary from pandexo output. If parameter space was run in run_pandexo make sure to restructure the input as a list of dictionaries without they key words that run_pandexo assigns.
- **plot** (*bool*) – (Optional) True renders plot, Flase does not. Default=True
- **output_file** (*str*) – (Optional) Default = bkt.html

Returns

- **x** (*numpy array*) – micron
- **y** (*numpy array*) – 1D bakground e/s

See also:

jwst_1d_spec(), *jwst_noise()*, *jwst_1d_flux()*, *jwst_1d_snr()*, *jwst_2d_det()*, *jwst_2d_sat()*

engine.justplotit.**jwst_1d_flux**(*result_dict*, *plot=True*, *output_file='flux.html'*)
Plot flux rate in e/s

Parameters

- **result_dict** (*dict*) – Dictionary from pandexo output. If parameter space was run in run_pandexo make sure to restructure the input as a list of dictionaries without they key words that run_pandexo assigns.
- **plot** (*bool*) – (Optional) True renders plot, Flase does not. Default=True
- **output_file** (*str*) – (Optional) Default = 'flux.html'

Returns

- **x** (*numpy array*) – micron
- **y** (*numpy array*) – 1D flux rate in electrons/s

See also:

jwst_1d_spec(), *jwst_1d_bkg()*, *jwst_noise()*, *jwst_1d_snr()*, *jwst_2d_det()*, *jwst_2d_sat()*

engine.justplotit.**jwst_1d_snr**(*result_dict*, *plot=True*, *output_file='snr.html'*)
Plot SNR

Parameters

- **result_dict** (*dict*) – Dictionary from pandexo output. If parameter space was run in run_pandexo make sure to restructure the input as a list of dictionaries without they key words that run_pandexo assigns.
- **plot** (*bool*) – (Optional) True renders plot, Flase does not. Default=True
- **output_file** (*str*) – (Optional) Default = 'snr.html'

Returns

- **x** (*numpy array*) – micron
- **y** (*numpy array*) – 1D SNR

See also:

jwst_1d_bkg(), *jwst_noise()*, *jwst_1d_flux()*, *jwst_1d_spec()*, *jwst_2d_det()*, *jwst_2d_sat()*

engine.justplotit.**jwst_1d_spec**(*result_dict, model=True, title='Model + Data + Error Bars', output_file='data.html', legend=False, R=False, num_tran=False, plot_width=800, plot_height=400, x_range=[1, 10], y_range=None, plot=True*)
Plots 1d simulated spectrum and rebin or rescale for more transits

Plots 1d data points with model in the background (if wanted). Designed to read in exact output of run_pandexo.

Parameters

- **result_dict** (*dict or list of dict*) – Dictionary from pandexo output. If parameter space was run in run_pandexo make sure to restructure the input as a list of dictionaries without they key words that run_pandexo assigns.
- **model** (*bool*) – (Optional) True is default. True plots model, False does not plot model
- **title** (*str*) – (Optional) Title of plot. Default is "Model + Data + Error Bars".

- **output_file** (*str*) – (Optional) name of html file for you bokeh plot. After bokeh plot is rendered you will have the option to save as png.
- **legend** (*bool*) – (Optional) Default is False. True, plots legend.
- **R** (*float*) – (Optional) Rebin data from native instrument resolution to specified resolution. Dafult is False, no binning.
- **num_tran** (*float*) – (Optional) Scales data by number of transits to improve error by sqrt(*num_trans*)
- **plot_width** (*int*) – (Optional) Sets the width of the plot. Default = 800
- **plot_height** (*int*) – (Optional) Sets the height of the plot. Default = 400
- **y_range** (*list of int*) – (Optional) sets y range of plot. Defaut is +- 10% of max and min
- **x_range** (*list of int*) – (Optional) Sets x range of plot. Default = [1,10]
- **plot** (*bool*) – (Optional) Supresses the plot if not wanted (Default = True)

Returns **x,y,e** – Returns wave axis, spectrum and associated error in list format. x[0] will be correspond to the first dictionary input, x[1] to the second, etc.

Return type list of arrays

Examples

```
>>> jwst_1d_spec(result_dict, num_tran = 3, R = 35) #for a single plot
```

If you wanted to save each of the axis that were being plotted:

```
>>> x,y,e = jwst_1d_data([result_dict1, result_dict2], model=False, num_tran = 5,␣
↪R = 100) #for multiple
```

See also:

jwst_noise(), *jwst_1d_bkg()*, *jwst_1d_flux()*, *jwst_1d_snr()*, *jwst_2d_det()*, *jwst_2d_sat()*

engine.justplotit.**jwst_2d_det**(*result_dict*, *plot=True*, *output_file='det2d.html'*)

Plot 2d detector image

Parameters

- **result_dict** (*dict*) – Dictionary from pandexo output. If parameter space was run in run_pandexo make sure to restructure the input as a list of dictionaries without they key words that run_pandexo assigns.
- **plot** (*bool*) – (Optional) True renders plot, Flase does not. Default=True
- **output_file** (*str*) – (Optional) Default = 'det2d.html'

Returns 2D array of out of transit detector simulation

Return type numpy array

See also:

jwst_1d_spec(), *jwst_1d_bkg()*, *jwst_1d_flux()*, *jwst_1d_snr()*, *jwst_noise()*, *jwst_2d_sat()*

engine.justplotit.**jwst_2d_sat**(*result_dict*, *plot=True*, *output_file='sat2d.html'*)
Plot 2d saturation profile

Parameters

- **result_dict** (*dict*) – Dictionary from pandexo output. If parameter space was run in run_pandexo make sure to restructure the input as a list of dictionaries without they key words that run_pandexo assigns.
- **plot** (*bool*) – (Optional) True renders plot, Flase does not. Default=True
- **output_file** (*str*) – (Optional) Default = 'sat2d.html'

Returns 2D array of out of transit detector simulation

Return type numpy array

See also:

jwst_1d_spec(), *jwst_1d_bkg()*, *jwst_1d_flux()*, *jwst_1d_snr()*, *jwst_2d_det()*, *jwst_noise()*

engine.justplotit.**jwst_noise**(*result_dict*, *plot=True*, *output_file='noise.html'*)
Plot background

Parameters

- **result_dict** (*dict*) – Dictionary from pandexo output. If parameter space was run in run_pandexo make sure to restructure the input as a list of dictionaries without they key words that run_pandexo assigns.
- **plot** (*bool*) – (Optional) True renders plot, Flase does not. Default=True
- **output_file** (*str*) – (Optional) Default = 'noise.html'

Returns

- **x** (*numpy array*) – micron
- **y** (*numpy array*) – 1D noise (ppm)

See also:

jwst_1d_spec(), *jwst_1d_bkg()*, *jwst_1d_flux()*, *jwst_1d_snr()*, *jwst_2d_det()*, *jwst_2d_sat()*

engine.justplotit.**uniform_tophat_mean**(*xnew*, *x*, *y*)
Adapted from Mike R. Line to rebin spectra

Takes average of group of points in bin

Parameters

- **xnew** (*list of float or numpy array of float*) – New wavelength grid to rebin to
- **x** (*list of float or numpy array of float*) – Old wavelength grid to get rid of
- **y** (*list of float or numpy array of float*) – New rebinned y axis

Returns new y axis

Return type array of floats

Examples

```
>>> oldgrid = np.linspace(1,3,100)
>>> y = np.zeros(100)+10.0
>>> newy = uniform_tophat_sum(np.linspace(2,3,3), oldgrid, y)
>>> newy
array([ 240.,  250.,  130.])
```

engine.justplotit.**uniform_tophat_sum**(*xnew*, *x*, *y*)

Adapted from Mike R. Line to rebin spectra

Takes sum of group of points in bin of wave points :param xnew: New wavelength grid to rebin to :type xnew: list of float or numpy array of float :param x: Old wavelength grid to get rid of :type x: list of float or numpy array of float :param y: New rebinned y axis :type y: list of float or numpy array of float

Returns new y axis

Return type array of floats

Examples

```
>>> oldgrid = np.linspace(1,3,100)
>>> y = np.zeros(100)+10.0
>>> newy = uniform_tophat_sum(np.linspace(2,3,3), oldgrid, y)
>>> newy
array([ 240.,  250.,  130.])
```

12.3 engine.bintools

engine.bintools.**bin_wave_to_R**(*w*, *R*)

Creates new wavelength axis at specified resolution

Parameters

- **w** (*list of float or numpy array of float*) – Wavelength axis to be re-binned
- **R** (*float or int*) – Resolution to bin axis to

Returns New wavelength axis at specified resolution

Return type list of float

Examples

```
>>> newwave = bin_wave_to_R(np.linspace(1,2,1000), 10)
>>> print((len(newwave)))
11
```

engine.bintools.**binning**(*x*, *y*, *dy=None*, *binwidth=None*, *r=None*, *newx=None*, *log=False*, *nan=False*)

This contains functionality for binning spectroscopy given an x, y and set of errors. This is similar to IDL's regroup but in Python (obviously). Note that y is binned as the mean(ordinates) instead of sum(ordinates), as you would want for cmputing flux in a set of pixels. User can input a constant resolution, constant binwidth or

provide a user defined bin. The error is computed as sum(sqrt(sig1^2, sig2^2, sig3^2))/3, for example if there were 3 points to bin.

Parameters

- **x** (*array, float*) – vector containing abcissae.
- **y** (*array,float*) – vector containing ordinates.
- **dy** (*array,float*) – (Optional) errors on ordinates, can be float or array
- **binwidth** (*float*) – (Optional) constant bin width in same units as x
- **r** (*float*) – (Optional) constant resolution to bin to
- **newx** (*array, float*) – (Optional) new x axis to bin to
- **log** (*bool*) – (Optional) computes equal bin spacing logarithmically, Default = False
- **sort** (*bool*) – (Optional) sort into ascending order of x, default = True
- **nan** (*bool*) – (Optional) if true, this returns nan values where no points exist in a given bin Otherwise, all nans are dropped

Returns bin_y : binned ordinates bin_x : binned abcissae bin_edge : edges of bins (always contains len(bin_x)+1 elements) bin_dy : error on ordinate bin bin_n : number of points contained in each bin

Return type dict

Examples

```
>>> from bintools import binning
```

If you want constant resolution (using output dict from PandExo):

```
>>> pandexo = result['FinalSpectrum']
>>> x, y, err = pandexo['wave'], pandexo['spectrum_w_rand'], pandexo['error_w_
↪floor']
>>> final = binning(x, y, dy = err, r =100)
>>> newx, newy, newerr = final['bin_x'], final['bin_y'], final['bin_dy']
```

If you have a x axis that you want PandExo output to be binned to

```
>>> newx = np.linspace(1,5,10)
>>> final = binning(x, y, dy = err, newx =newx)
>>> newx, newy, newerr = final['bin_x'], final['bin_y'], final['bin_dy']
```

If you want a constant bin width and want everything to be spaced linearly

```
>>> final = binning(x, y, dy = err, binwidth = 0.1)
>>> newx, newy, newerr = final['bin_x'], final['bin_y'], final['bin_dy']
```

If you want constant bin width but want everything to spaced logarithmically

```
>>> final = binning(x, y, dy = err, binwidth = 0.1, log=True)
>>> newx, newy, newerr = final['bin_x'], final['bin_y'], final['bin_dy']
```

engine.bintools.**uniform_tophat_mean**(*newx*, *x*, *y*, *dy=None*, *nan=False*)

Adapted from Mike R. Line to rebin spectra

Takes mean of groups of points in certain wave bin

Parameters

- **newx** (*list of float or numpy array of float*) – New wavelength grid to rebin to
- **x** (*list of float or numpy array of float*) – Old wavelength grid to get rid of
- **y** (*list of float or numpy array of float*) – New rebinned y axis

Returns new wavelength grid

Return type array of floats

Examples

```
>>> from pandexo.engine.jwst import uniform_tophat_sum
>>> oldgrid = np.linspace(1,3,100)
>>> y = np.zeros(100)+10.0
>>> newy = uniform_tophat_sum(np.linspace(2,3,3), oldgrid, y)
>>> newy
array([ 240.,  250.,  130.])
```

12.4 engine.run_online

class engine.run_online.**AboutHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

get()
Render about PandExo Page

class engine.run_online.**Application**
Bases: tornado.web.Application

Gobal settings of the server This defines the global settings of the server. This parses out the handlers, and includes settings for if ever we want to tie this to a database.

class engine.run_online.**BaseHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler

Logic to handle user information and database access might go here.

buffer = OrderedDict()

executor = <concurrent.futures.process.ProcessPoolExecutor object>

write_error(*status_code*, ***kwargs*)
This renders a customized error page

class engine.run_online.**CalculationDownloadHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

Handlers returning the downloaded data of a particular calculation task. Handlers returning the status of a particular calculation task.

get(*id*)

class engine.run_online.**CalculationDownloadPandInHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

Handlers returning the downloaded data of a particular calculation task. Handlers returning the status of a particular calculation task.

get(*id*)

class engine.run_online.**CalculationNewHSTHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

This request handler deals with processing the form data and submitting a new HST calculation task to the parallelized workers.

get()

post()
The post method contains the retured data from the form data (accessed by using *self.get_argument(...)* for specific arguments, or *self.request.body* to grab the entire returned object.

class engine.run_online.**CalculationNewHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

This request handler deals with processing the form data and submitting a new calculation task to the parallelized workers.

get()

post()
The post method contains the returned data from the form data (accessed by using *self.get_argument(...)* for specific arguments, or *self.request.body* to grab the entire returned object.

class engine.run_online.**CalculationStatusHSTHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

Handlers returning the status of a particular HST calculation task.

get(*id*)

class engine.run_online.**CalculationStatusHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

Handlers returning the status of a particular JWST calculation task.

get(*id*)

class engine.run_online.**CalculationTask**(*id*, *name*, *task*, *cookie*, *count*)
Bases: tuple

cookie
Alias for field number 3

count
Alias for field number 4

id
Alias for field number 0

name
Alias for field number 1

task
Alias for field number 2

class engine.run_online.**CalculationViewHSTHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

This handler deals with passing the results from Pandeia to the *create_component_hst* function which generates the Bokeh interative plots.

get(*id*)

class engine.run_online.**CalculationViewHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

This handler deals with passing the results from Pandeia to the *create_component_jwst* function which generates the Bokeh interative plots.

get(*id*)

class engine.run_online.**DashboardHSTHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

Request handler for the dashboard page. This will retrieve and render the html template, along with the list of current task objects.

get()

class engine.run_online.**DashboardHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

Request handler for the dashboard page. This will retrieve and render the html template, along with the list of current task objects.

get()

class engine.run_online.**HelpfulPlotsHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

get()
Renders helpful bokeh plots

class engine.run_online.**HomeHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

get()
This sets an **unsecured** cookie. If user accounts gets implemented, this must be changed to a secure cookie.

class engine.run_online.**TablesHandler**(*application*, *request*, ***kwargs*)
Bases: *engine.run_online.BaseHandler*

get()
Render tables with confirmed candidates

engine.run_online.**main**()

12.5 engine.compute_noise

class engine.compute_noise.**ExtractSpec**(*inn*, *out*, *rn*, *extraction_area*, *timing*)
Bases: object

Different methods for computing noise.

PandExo has several different methods of computing noise. MULTIACCUM (slope method) assumses that each frame is fit for up the ramp and that the final nosie includes correlated noise from fitting each frame up the ramp. First Minus Last assumes that intermediate frames are not used and final noise is just the first frame minus the

last frame. 2d extract uses Pandeia's 2d simualtions to extracts the spectrum from the 2d extraction box. It extracts the entire postage stamp so it might be an overestimate of flux (in contrast pandeia requires background extraction region to be at least equal to the flux extraction region)

Noise Components Included:

- Shot
- Background and Dark Current
- Read noise

Parameters

- **inn** (*dict*) – In transit dictionary computed from PandExo
- **out** (*dict*) – Out of transit dictionary comptued from PandExo
- **rn** (*float*) – Read noise electrons
- **extraction_area** (*float*) – Number of extracted pixels (square pixels)
- **timing** (*dict*) – Dictionary of computed JWST timing products

loopingL()
extracts pixels from center to bottom

loopingU()
extracts pixels from center to top

sum_spatial()
sums pixels in optimal extraction region

extract_region()
determines optimal extraction region

run_2d_extract()
top level to extract spec from 2d pandeia output

run_slope_method()
computes noise using multiaccum formulation

run_f_minus_l()
computs noise using first minus last

run_phase_spec()
computs noise for phase curve observations

extract_region()
Determine extraction Region

Contains functionality to determine extraction region from Pandeia 2d noise simulations. Calls *self.loopingL* and *self.loopingU*.

Returns bounding regions, photon and noise to be summed

Return type dict

loopingL(*cen*, *signal_col*, *noise_col*, *bkg_col*)
Finds bottom of the optimal extraction region.

Find location where SNR is the highest and loop from highest value downward

Parameters

- **cen** (*float or int*) – Pixel where SNR is the highest

- **signal_col** (*array of float*) – Array of fluxes to be extracted in a single column on the detector
- **noise_col** (*array of float*) – Array of noise to be extracted in a single column on the detector
- **bkg_col** (*array of float*) – Array of background fluxes to be extracted in a single column on the detector

Returns Bottom most pixel to be extracted

Return type int

loopingU(*cen*, *signal_col*, *noise_col*, *bkg_col*)

Finds top of the optimal extraction region.

Find location where SNR is the highest and loop from highest value upward

Parameters

- **cen** (*float or int*) – Pixel where SNR is the highest
- **signal_col** (*array of float*) – Array of fluxes to be extracted in a single column on the detector
- **noise_col** (*array of float*) – Array of noise to be extracted in a single column on the detector
- **bkg_col** (*array of float*) – Array of background fluxes to be extracted in a single column on the detector

Returns Top most pixel to be extracted

Return type int

run_2d_extract()

Extract noise from 2d detector image

Contains functionality to extract noise from 2d detector image

Returns all optimally extracted 1d products

Return type dict

run_f_minus_l()

Compute noise using first minus last formula

Uses 1d exracted products from pandeia to compute noise without multiaccum noise formula from Rauscher 07. Includes readnoise background.

Returns all optimally extracted 1d products for a single transit

Return type dict

run_phase_spec()

Computes time dependent noise for phase curves.

Computes noise for phase curve analysis instead of spectroscopy. Using MULTIACCUM formula here but this could be changed in the future. Does not return a spectra for each time element... On your own for that.

Returns all optimally extracted 1d products

Return type dict

run_slope_method()
Compute noise using Pandeia 1d noise

Contains functionality to compute noise using Pandeia 1d noise output products (uses MULTIACCUM) noise formula

Returns all optimally extracted 1d products for a single occultation

Return type dict

sum_spatial(*extract_info*)
Sum pixel in the spatial direction

Takes extraction info from *extract_region* and sums pixels in that region taking into account integrations and number of transits

Parameters **extract_info** (*dict*) – Dictionary with information on extraction box, flux and noise products

Returns Dictionary with all extracted 1d products including noise, background, fluxes

Return type dict

12.6 engine.create_input

engine.create_input.**bothTrans**(*out_trans*, *planet*, *star=None*)
Calculates in transit flux

Takes output from *outTrans*, which is the normalized stellar flux, and creates either a transit transmission spectrum, phase curve or emission spectrum. Magnitude

Parameters

- **out_trans** (*dict*) – includes dictionary from *outTrans* output.
- **planet** (*dict*) – dictionary with direction to planet spectra, wavelength and flux units
- **star** (*dict*) – (Optional) dictionary within exo_input with stellar information. Only used when scaling Fortney Grid spectra to get (rp/r*)^2

Returns dictionary with out of transit flux, in transit flux, original model and corresponding wavelengths

Return type dict

engine.create_input.**hst_spec**(*planet*, *star*)
Calculates in transit flux

Takes output from *outTrans*, which is the normalized stellar flux, and creates either a transit transmission spectrum, phase curve or emission spectrum. Magnitude

Parameters

- **planet** (*dict*) – dictionary with direction to planet spectra, wavelength and flux units
- **star** (*dict*) – dictionary within exo_input with stellar information. Only used when scaling Fortney Grid spectra to get (rp/r*)^2

Returns dictionary with out of transit flux, in transit flux, original model and corresponding wavelengths

Return type dict

engine.create_input.**outTrans**(*input*)
Compute out of transit spectra

Computes the out of transit spectra by normalizing flux to specified magnitude and convert to specified Pandeia units of milliJy and microns.

Parameters **input** (*dict*) – stellar scene which includes parameters to extract phoenix database or a filename which points to a stellar spectrum

Returns contains wave and flux_out_trans

Return type dict

12.7 engine.hst

engine.hst.**calc_start_window**(*eventType*, *rms*, *ptsOrbit*, *numOrbits*, *depth*, *inc*, *aRs*, *period*, *windowSize*, *ecc=0*, *w=90.0*, *duration=None*, *offset=0.0*, *useFirstOrbit=False*)
Calculate earliest and latest start times

Plot earliest and latest possible spectroscopic light curves for given start window size

Parameters

- **eventType** (*str*) – 'transit' or 'eclipse'
- **rms** (*float*) – light curve root-mean-square
- **ptsOrbit** (*float*) – number of frames per HST orbit
- **numOrbits** (*float*) – number of HST orbits per visit
- **depth** (*float*) – transit/eclipse depth
- **inc** (*float*) – orbital inclination in degrees
- **aRs** (*float*) – Semi-major axis in units of stellar radii (a/R*)
- **period** (*float*) – orbital period in days
- **windowSize** (*float*) – observation start window size in minutes
- **ecc** (*float*) – (Optional) eccentricity
- **w** (*float*) – (Optional) longitude of periastron
- **duration** (*float*) – (Optional) full transit/eclipse duration in days
- **offset** (*float*) – (Optional) manual offset in observation start time, in minutes
- **useFirstOrbit** (*bool*) – (Optional) whether to use first orbit

Returns

- *float* – minphase–earliest observation start phase
- *float* – maxphase–latest observation start phase

engine.hst.**compute_sim_hst**(*dictinput*)
Sets up HST simulations

Function to set up explanet observations for HST only and compute simulated spectrum.

Parameters **dictinput** (*dict*) – instrument and pandexo dictionaries in format {"pandeia_input":dict1, "pandexo_input":dict2}

Returns All hst output info needed to plot simulated data, light curves timing info

Return type dict

engine.hst.**create_out_div**(*input_dict*, *minphase*, *maxphase*)
Function to render input dicts in html format for web front end

Parameters **input_dict** (*dict*) – any input dictionary

Returns html rendered table

Return type div

engine.hst.**planet_spec**(*planet*, *star*, *w_unit*, *disperser*, *deptherr*, *nchan*, *smooth=None*)
Plot exoplanet transmission/emission spectrum

Parameters

- **planet** (*dict*) – planet dictionary from exo_input
- **star** (*dict*) – star dictionary from exo_input
- **w_unit** (*str*) – wavelength unit (um or nm)
- **disperser** – grism (g102 or g141)
- **deptherr** (*float*) – simulated transit/eclipse depth uncertainty
- **nchan** (*float*) – number of spectrophotometric channels
- **smooth** (*float*) – (Optional)length of smoothing kernel

  returns contains following keys {'model_wave','model_spec','binwave','binspec', 'error','wmin','wmax'}

  rtype dict

engine.hst.**wfc3_GuessNOrbits**(*trdur*)
Predict number of HST orbits

Predict number of HST orbits for transit observation when not provided by the user.

Parameters **trdur** (*float*) – transit duration in days

Returns number of requested orbits per transit (including discarded thermal-settling orbit)

Return type float

engine.hst.**wfc3_GuessParams**(*hmag*, *disperser*, *scanDirection*, *subarray*, *obsTime*, *maxScanHeight=180.0*, *maxExptime=150.0*)
Predict nsamp and samp_seq when values not provided by the user.

Parameters

- **hmag** (*float*) – H-band magnitude
- **disperser** (*str*) – grism ('G141' or 'G102')
- **scanDirection** (*str*) – spatial scan direction ('Forward' or 'Round Trip')
- **subarray** (*str*) – Subarray aperture ('grism256' or 'grism512')
- **obsTime** (*float*) – Available observing time per HST orbit in seconds
- **maxScanHeight** (*float*) – (optional) maximum scan height in pixels
- **maxExptime** (*float*) – (Optional) default=150.0, maximum exposure time in seconds

Returns

- *float* – nsamp–number of up-the-ramp samples (1..15)
- *str* – samp_seq–time between non-destructive reads

engine.hst.**wfc3_TExoNS**(*dictinput*)
Compute Transit depth uncertainty

Compute the transit depth uncertainty for a defined system and number of spectrophotometric channels. Written by Kevin Stevenson October 2016

Parameters **dictinput** (*dict*) – dictionary containing instrument parameters and exoplanet specific parameters. {"pandeia_input":dict1, "pandexo_input":dict1}

Returns

- *float* – deptherr–transit depth uncertainty per spectrophotometric channel
- *float* – chanrms–light curve root mean squarerms
- *float* – ptsOrbit–number of HST orbits per visit

engine.hst.**wfc3_obs**(*hmag*, *disperser*, *scanDirection*, *subarray*, *nsamp*, *samp_seq*)
Determine the recommended exposure time, scan rate, scan height, and overheads.

Parameters

- **hmag** (*float*) – H-band magnitude
- **disperser** (*str*) – Grism ('G141' or 'G102')
- **scanDirection** (*str*) – spatial scan direction ('Forward' or 'Round Trip')
- **subarray** (*str*) – Subarray aperture ('grism256' or 'grism512')
- **nsamp** (*float*) – Number of up-the-ramp samples (1..15)
- **samp_seq** (*str*) – Time between non-destructive reads ('SPARS5', 'SPARS10', or 'SPARS25')

Returns

- *float* – exptime–exposure time in seconds
- *float* – tottime–total frame time including overheads in seconds
- *float* – scanRate–recommented scan rate in arcsec/s
- *float* – scanHeight–scan height in pixels
- *float* – fluence–maximum pixel fluence in electrons

12.8 engine.jwst

engine.jwst.**add_noise_floor**(*noise_floor*, *wave_bin*, *error_spec*)
Add in noise floor

This adds in a user speficied noise floor. Does not add the noise floor in quadrature isntead it sets error[error<noise_floor] = noise_floor. If a wavelength dependent noise floor is given and the wavelength ranges are off, it interpolates the out of range noise floor.

Parameters

- **noise_floor** (*str or int*) – file with two column [wavelength, noise(ppm)] or single number with constant noise floor in ppm

- **wave_bin** (*array of float*) – final binned wavelength grid from simulation
- **error_spec** (*array of float*) – final computed error on the planet spectrum in units of rp^2/r*^2 or fp/f*

Returns error_spec– new error

Return type array of float

Examples

```
>>> import numpy as np
>>> wave = np.linspace(1,2.7,10)
>>> error = np.zeros(10)+1e-6
>>> newerror = add_noise_floor(20, wave, error)
>>> print(newerror)
[  2.00000000e-05   2.00000000e-05   2.00000000e-05   2.00000000e-05
   2.00000000e-05   2.00000000e-05   2.00000000e-05   2.00000000e-05
   2.00000000e-05   2.00000000e-05]
```

engine.jwst.**add_warnings**(*pand_dict*, *timing*, *sat_level*, *flags*, *instrument*)

Add warnings for front end

Adds in necessary warning flags for a JWST observation usually associated with too few or too many groups or saturation. Alerts user if saturation level is higher than 80 percent and if the number of groups is less than 5. Or, if the full well is greater than 80. These warnings are currently very arbitrary. Will be updated as better JWST recommendations are made.

Parameters

- **pand_dict** – output from pandeia run
- **timing** (*dict*) – output from **compute_timing**
- **sat_level** (*int or float*) – user specified saturation level in fractional (00/100)
- **flags** (*dict*) – warning flags taken from output of **compute_timing**
- **instrument** (*str*) – Only allowable strings are: “nirspec”, “niriss”, “nircam”, “miri”

Returns all warnings

Return type dict

Notes

These are warnings are just suggestions and are not yet required.

engine.jwst.**as_dict**(*out*, *both_spec*, *binned*, *timing*, *mag*, *sat_level*, *warnings*, *punit*, *unbinned*, *calculation*)

Format dictionary for output data

Takes all output from jwst run and converts it to simple dictionary

Parameters

- **out** (*dict*) – output dictionary from **compute_out**
- **both_spec** (*dict*) – output dictionary from **createInput.bothTrans**
- **binned** (*dict*) – dictionary from **wrapper**

- **timing** (*dict*) – dictionary from **compute_timing**
- **mag** (*dict*) – magnitude of system
- **sat_level** (*float or int*) – saturation level in electrons
- **warnings** (*dict*) – warning dictionary from **add_warnings**
- **punit** (*"fp/f*" or "rp^2/r*^2"*) – unit of supplied spectra options are: only options are fp/f* or rp^2/r*^2
- **unbinned** (*dict*) – unbinned raw data from **wrapper**
- **calculation** (*str*) – noise calculation type

Returns compressed dictionary

Return type dict

engine.jwst.**bin_wave_to_R**(*w*, *R*)

Creates new wavelength axis at specified resolution

Parameters

- **w** (*list of float or numpy array of float*) – Wavelength axis to be rebinned
- **R** (*float or int*) – Resolution to bin axis to

Returns New wavelength axis at specified resolution

Return type list of float

Examples

```
>>> newwave = bin_wave_to_R(np.linspace(1,2,1000), 10)
>>> print((len(newwave)))
11
```

engine.jwst.**compute_full_sim**(*dictinput*)

Top level function to set up exoplanet obs. for JW

Function to set up explanet observations for JWST only and compute simulated spectrum. It uses STScI's Pandeia to compute instrument throughputs and WebbPSF to compute PSFs.

Parameters **dictinput** (*dict*) – dictionary containing instrument parameters and exoplanet specific parameters. {"pandeia_input":dict1, "pandexo_input":dict1}

Returns large dictionary with 1d, 2d simualtions, timing info, instrument info, warnings

Return type dict

Examples

```
>>> from .pandexo.engine.jwst import compute_full_sim
>>> from .pandexo.engine.justplotit import jwst_1d_spec
>>> a = compute_full_sim({"pandeia_input": pandeiadict, "pandexo_input":exodict})
>>> jwst_1d_spec(a)
.. image:: 1d_spec.png
```

Notes

It is much easier to run simulations through either **run_online** or **justdoit**. **justdoit** contains functions to create input dictionaries and **run_online** contains web forms to create input dictionaries.

See also:

pandexo.engine.justdoit.run_pandexo() Best function for running pandexo runs

pandexo.engine.run_online() Allows running functions through online interface

engine.jwst.**compute_maxexptime_per_int**(*pandeia_input*, *sat_level*)
Computes optimal maximum exposure time per integration

Function to simulate 2d jwst image with 2 groups, 1 integration, 1 exposure and return the maximum time for one integration before saturation occurs. If saturation has already occured, returns maxexptime_per_int as np.nan. This then tells Pandexo to set min number of groups (ngroups =2). This avoids error if saturation occurs. This routine assumes that min ngroups is 2.

Parameters

- **pandeia_input** (*dict*) – pandeia dictionary input
- **sat_level** (*int or float*) – user defined saturation level in units of electrons

Returns Maximum exposure time per integration before specified saturation level

Return type float

Examples

```
>>> max_time = compute_maxexptime_per_int(pandeia_input, 50000.0)
>>> print(max_time)
12.0
```

engine.jwst.**compute_timing**(*m*, *transit_duration*, *expfact_out*, *noccultations*)
Computes all timing info for observation

Computes all JWST specific timing info for observation including. Some pertinent JWST terminology:

- frame: The result of sequentially clocking and digitizing all pixels in a rectangular area of an SCA. **Full-fame readout** means to digitize all pixels in an SCA, including reference pixels. **Frame** also applies to the result of clocking and digitizing a subarray on an SCA.
- group: One or more consecutively read frames. There are no intervening resets. Frames may be averaged to form a group but for exoplanets the read out scheme is always 1 frame = 1 group
- integration: The end result of resetting the detector and then non-destructively sampling it one or more times over a finite period of time before resetting the detector again. This is a unit of data for which signal is proportional to intensity, and it consists of one or more GROUPS.
- exposure: The end result of one or more INTEGRATIONS over a finite period of time. EXPOSURE defines the contents of a single FITS file.

Parameters

- **m** (*dict*) – Dictionary output from **compute_maxexptime_per_int**
- **transit_duration** (*float or int*) – transit duration in seconds
- **expfact_out** (*float or int*) – fraction of time spent in transit versus out of transit

- **noccultations** (*int*) – number of transits

Returns

- **timing** (*dict*) – All timing info
- **warningflag** (*dict*) – Warning flags

Examples

```
>>> timing, flags = compute_timing(m, 2*60.0*60.0, 1.0, 1.0)
>>> print((list(timing.keys())))
['Number of Transits', 'Num Integrations Out of Transit', 'Num Integrations In␣
↪Transit',
'APT: Num Groups per Integration', 'Seconds per Frame', 'Observing Efficiency (%)
↪', 'On Source Time(sec)',
'Exposure Time Per Integration (secs)', 'Reset time Plus 30 min TA time (hrs)',
'APT: Num Integrations per Occultation', 'Transit Duration']
```

engine.jwst.**perform_in**(*pandeia_input*, *pandexo_input*, *timing*, *both_spec*, *out*, *calculation*)

Computes in transit data

Runs Pandeia for the in transit data or computes the in transit simulation from the out of transit pandeia run

Parameters

- **pandeia_input** (*dict*) – pandeia specific input info
- **pandexo_input** (*dict*) – exoplanet specific observation info
- **timing** (*dict*) – timing dictionary from **compute_timing**
- **both_spec** (*dict*) – dictionary transit spectra computed from **createInput.bothTrans**
- **out** (*dict*) – out of transit dictionary from **perform_in**
- **calculation** (*str*) – key which speficies the kind of noise calcualtion (2d extract, slope method, fml, phase_spec). Recommended for transit transmisstion spectra = fml

Returns pandeia output dictionary

Return type dict

engine.jwst.**perform_out**(*pandeia_input*, *pandexo_input*, *timing*, *both_spec*)

Runs pandeia for the out of transit data

Parameters

- **pandeia_input** (*dict*) – pandeia specific input info
- **pandexo_input** (*dict*) – exoplanet specific observation info
- **timing** (*dict*) – timing dictionary from **compute_timing**
- **both_spec** (*dict*) – dictionary transit spectra computed from **createInput.bothTrans**

Returns pandeia output dictionary for out of transit data

Return type dict

engine.jwst.**remove_QY**(*pandeia_dict*, *instrument*)

Removes Quantum Yield from Pandeia Fluxes. Place Holder.

Parameters

- **pandeia_dict** (*dict*) – pandeia output dictionary
- **instrument** (*str*) – instrument running

Returns same exact dictionary with extracted_flux = extracted_flux/QY

Return type dict

engine.jwst.**target_acq**(*instrument*, *both_spec*, *warning*)

Contains functionality to compute optimal TA strategy

Takes pandexo normalized flux from create_input and checks for saturation, or if SNR is below the minimum requirement for each. Then adds warnings and 2d displays and target acq info to final output dict

Parameters

- **instrument** (*str*) – possible options are niriss, nirspec, miri and nircam
- **both_spec** (*dict*) – output dictionary from **create_input**
- **warning** (*dict*) – output dictionary from **add_warnings**
- **Retruns** –
- **-------** –

engine.jwst.**uniform_tophat_sum**(*newx*, *x*, *y*)

Adapted from Mike R. Line to rebin spectra

Sums groups of points in certain wave bin

Parameters

- **newx** (*list of float or numpy array of float*) – New wavelength grid to rebin to
- **x** (*list of float or numpy array of float*) – Old wavelength grid to get rid of
- **y** (*list of float or numpy array of float*) – New rebinned y axis

Returns new wavelength grid

Return type array of floats

Examples

```
>>> from .pandexo.engine.jwst import uniform_tophat_sum
>>> oldgrid = np.linspace(1,3,100)
>>> y = np.zeros(100)+10.0
>>> newy = uniform_tophat_sum(np.linspace(2,3,3), oldgrid, y)
>>> newy
array([ 240.,  250.,  130.])
```

12.9 engine.load_modes

class engine.load_modes.**SetDefaultModes**(*inst*)

Bases: object

Class to load default instrument mode dicts

This class contains functionality for loading observing modes for exoplanet observations This is NOT a complete set of ALL possibleobserving modes. Instead, it offers a starting point for choosing one instrument specification. There is one function for each instrument. For example, It sets slitless mode for MIRI LRS. If users are interested in other specific observation modes they should load in the dictionary and then edit individual keys.

Included modes are:

- "MIRI LRS"
- "NIRISS SOSS"
- "NIRSpec G140M"
- "NIRSpec G140H"
- "NIRSpec G235M"
- "NIRSpec G235H"
- "NIRSpec G395M"
- "NIRSpec G395H"
- "NIRSpec Prism"
- "NIRCam F322W2"
- "NIRCam F444W"
- "WFC3 G102"
- "WFC3 G141"

Parameters **inst** (*str*) – Allowable strings listed above

miri()
Handles MIRI template

nircam()
Handles NIRCam template

niriss()
Handles NIRISS template

nirspec()
Handles NIRSpec template

pick()
Points to specific instrument based on key choice

wfc3()
Handles WFC3 template

12.10 engine.pandexo

engine.pandexo.**wrapper**(*dictinput*)
Tpo level function to call either jwst, hst, wfirst

Top level function which calls either jwst, hst or wfirst noise simulation.

Parameters **dictinput** – dictionary containing instrument parameters and exoplanet specific parameters. {"pandeia_input":dict1, "pandexo_input":dict1}

Returns output specific to observatory requested

Return type dict

Notes

You should not run simulations through this. It is much easier to run simulations through either **run_online** or **justdoit**. **justdoit** contains functions to create input dictionaries and **run_online** contains web forms to create input dictionaries.

See also:

`pandexo.engine.justdoit()` gives ability to simply submit runs

`pandexo.engine.run_online()` submit runs through user interface

12.11 engine.elements

Properties of the chemical elements.

Each chemical element is represented as an object instance. Physicochemical and descriptive properties of the elements are stored as instance attributes.

Author Christoph Gohlke

Version 2015.01.29

12.11.1 Requirements

- CPython 2.7 or 3.4

References

1. http://physics.nist.gov/PhysRefData/Compositions/
2. http://physics.nist.gov/PhysRefData/IonEnergy/tblNew.html
3. http://en.wikipedia.org/wiki/%(element.name)s
4. http://www.miranda.org/~jkominek/elements/elements.db

Examples

```
>>> from elements import ELEMENTS
>>> len(ELEMENTS)
109
>>> str(ELEMENTS[109])
'Meitnerium'
>>> ele = ELEMENTS['C']
>>> ele.number, ele.symbol, ele.name, ele.eleconfig
(6, 'C', 'Carbon', '[He] 2s2 2p2')
>>> ele.eleconfig_dict
{(1, 's'): 2, (2, 'p'): 2, (2, 's'): 2}
```

```
>>> sum(ele.mass for ele in ELEMENTS)
14659.1115599
>>> for ele in ELEMENTS:
...     ele.validate()
...     ele = eval(repr(ele))
```

12.12 engine.hst_smooth

engine.hst_smooth.**medfilt**(*x*, *window_len*)
Apply a length-k median filter to a 1D array x. Boundaries are extended by repeating endpoints.

engine.hst_smooth.**smooth**(*x*, *window_len=10*, *window='hanning'*)
smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

Parameters

- **x** (*array of floats*) – the input signal
- **window_len** (*int*) – (Optional) Default=10. The dimension of the smoothing window
- **window** (*str*) – the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman' flat window will produce a moving average smoothing.

Returns The smoothed signal

Return type array of floats

Examples

```
>>> t=linspace(-2,2,0.1)
>>> x=sin(t)+randn(len(t))*0.1
>>> y=smooth(x)
```

See also:

numpy.hanning(), numpy.hamming(), numpy.bartlett(), numpy.blackman(), numpy.convolve(), scipy.signal.lfilter(), Todos(), -----(), The()

Source() http://www.scipy.org/Cookbook/SignalSmooth 2009-03-13

12.13 engine.logs

engine.logs.**hst_log**(*final*)
Logging for Website;Used for tracking usage;Used for tracking usage;Only storing instrument data

engine.logs.**jwst_log**(*final*)
Logging for Website;Used for tracking usage;Only storing instrument data

12.14 Module contents

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

H

I

J

L

M

N

O

P

R

S

T

U

W