Creators:
Julien de-LanVersin, Polina Kanel, Christina Kreisch, Brianna Lacy, Heather Prince, Kengran (Blake) Yang
Princeton University, Princeton NJ

ExoSpec User Manual

13th January 2017

OVERVIEW

ExoSpec is a python tool for fitting your multi-wavelength transit light-curves. It can accept an arbitrary number of wavelength channels and an arbitrary number of auxiliary measurements. Currently the fitting has two Gaussian Process kernel options: the kernel outlined in Gibson 2011 which incorporates auxiliary measurements made over the course of transit observation and a more general squared exponential. This work utilizes the Python packages: batman, emcee, corner and george.

Installation

Dependencies: In addition to standard Python libraries, ExoSpec requires the following packages to run: numpy, scipy, matplotlib, virtualenv, mpi4py, emcee, george, batman-package, corner, and pandas. These packages are automatically installed when installing ExoSpec. To run tests with the setup.py file, the nose package is also required and installs automatically.

Runs with Python version 2.7.

0. Before installing ExoSpec, dependencies for george and mpi4py must be installed since these dependencies are not Python packages. You must install the following:

    a. Eigen3:
        i. On linux: `sudo apt-get install l ibeigen3-dev`
        ii. On mac: `brew install eigen`
        iii. On Windows: the developers of george say they did not test george on Windows, so it may not work but you can still try. We have not tested ExoSpec on Windows
    b. OpenMPI:

   i. On linux: `sudo apt-get install o penmpi-bin openmpi-common openssh-client openssh-server libopenmpi1.3 libopenmpi-dbg libopenmpi-dev`
   ii. On mac: `brew install openmpi`
c. Batman issues: If after moving to step 1 and running the setup.py file you receive an error from batman, you can install it from the source file instead
   i. Download the stable release here, and then run `sudo python setup.py install`

1. To install ExoSpec, download and unpack the source file. Then run `python setup.py install` and all dependencies and packages will be built.
2. A suite of tests are included in exospec/tests/

Data Input and Parameter File

Light curves must be saved in text files with columns for orbital phase or time, flux, error, and then columns of additional auxiliary measurements. Lines at the beginning of the file starting with a '#' will be ignored. An excerpt from an example light curve file is included in the appendix. The light curve files should be saved in a directory by themselves. They need to be named according to the following pattern: string + '_' + string + '_' + central wavelength + '.txt' . For example: "gj1214_lightcurve_600.txt" would be an appropriate name. The critical point is that the wavelength occurs directly after the second '_' in the name and before '.txt' . Note that in the current implementation, the length of the light curves, that is the number of times steps, has a very steep affect on the runtime. >>insert some actual values here<<

The parameter file contains entries in the form: <value name> = <actual value> . The exact order of these values do not matter. Comments can be made with a '#'. If the actual value is a list, this is handled by entering the parts of the list on a single line separated by a space. Values can be either strings or floats as is appropriate. It is critical that all expected value names are present. An example input parameter file is included in the appendix. See the following table for expected value names:

Required Entries in the Input Parameter File	
Parameter name	**Explanation**
lc_path	Path to directory where light curve files are stored
output_dir	Path to directory where user would like exospec to save output chains, tables and figures
wave_bin_size	Number of light curve files to combine into each wave

	bin, 1 corresponds to no binning at all. Note that in the current implementation the user must have a number of light curves which is divisible by the wave_bin_size parameter
n_errors	The number of auxiliary measurements the user wishes to include in the Gaussian Process kernel. Note that it is only worthwhile to add in an auxiliary measurement if it actually correlates significantly with red noise. Otherwise one simply adds unnecessary additional GP hyper parameters to the model.
ndim	The total number of parameters to be fit. This will change depending on the limb-darkening model chosen and the number of auxiliary measurements included. There will always be 5 parameters (radius of planet, 4 GP hyper parameters) + the number of parameters needed for the limb-darkening model + an additional GP hyper parameter for each auxiliary measurement. For example, a user wanting to include 4 auxiliary measures and 2 limb darkening parameters for the quadratic model would set ndim to 11.
kernel_type	Currently this can be set to “Custom” which incorporates the user’s auxiliary measures, or to “standard” which does not.
nwalkers	The number of walkers to send through the MCMC process. The emcee documentation recommends 3-4 times the number of parameters being fit.
nburnin	An initial number of steps which will be left off of the final chain. This allows the walkers to disperse in parameter space, and prevents the initial guesses from influencing the final fit.
nsteps	The number of post-burnin steps for the walkers to take.
mpi_flag	A boolean flag, if set to True the program will try to fit light curves for different wavelengths in parallel. If it is set to False, the program will fit the wavelengths in serial.
nthreads	This sets the number of threads used on each node, parallelizing the mcmc sampling of an individual light curve

visualization	A boolean flag, if set to True the program will save a simple summary table of best fit radii and limb-darkening parameters, a latex-able table, plots of walkers and triangle plots for each wavelength, and a plot of the transmission spectrum, along with the usual mcmc chains. If set to false the program will only the mcmc chains, which the user can then use to calculate their own best fits and make any desired plots.
confidence	The percentile for the confidence intervals reported in the latex table
separate_flag	A boolean flag, if set to True the latex_table output file will put GP hyper parameters and physical transit parameters into separate tables.
t0	time of inferior conjunction (t0, per, and the times input to the batman model must have same units. Days or orbital phase are fine)
per	orbital period (t0, per, and the times input to the batman model must have same units, e.g. days or orbital phase)
a	semi-major axis (in units of stellar radii)
inc	orbital inclination (in degrees)
ecc	eccentricity
w	longitude of periastron (in degrees)
limb_dark	limb darkening model, note options are quadratic and nonlinear.
rp	planet radius (in units of stellar radii)
u0, u1, ... uN	Include a separate parameter up to the number required by the limb darkening model selected. Quadratic takes only two, while nonlinear requires four.
p0	A list of the initial guess for all parameters that will be fit in the order: rp u0 u1 (u2 u3...) ksi variance gamma_time gamma_flux_error gamma_auxiliary_measures... The length of this list should be equal to ndim

transit_par_names	List of the string labels for the transit parameters being fit. This should be in the same order as p0
gp_hyper_par_names	List of the string labels for the GP hyper parameters being fit. This should again be in the same order as p0
kernel_a	Initial guess for the amplitude GP hyper parameter
kernel_gamma	List of initial guesses for all the scaling factors, in order time, flux error, then the additional auxiliary parameters included. This should have length 2 + the number of auxiliary measurements
kernel_variance	Initial guess for the variance GP hyper parameter
rp_prior_lower	Lower bound for the radius of the planet
rp_prior_upper	Upper bound for the radius of the planet
u_prior_lower	Lower bound for limb darkening coefficients
u_prior_upper	Upper bound for limb darkening coefficients
kernel_a_prior_lower	Lower bound for a, the amplitude GP hyper parameter. **This must not be set below 0.**
kernel_a_prior_upper	Upper bound for a, the amplitude GP hyper parameter
kernel_gamma_prior_lower	Lower bound for all scale factors. **This must not be set below 0.**
kernel_gamma_prior_lower	Upper bound for all scale factors
kernel_variance_prior_lower	Lower bound for the variance GP hyper parameter. **This must not be set below 0.**
kernel_variance_prior_upper	Upper bound for the variance GP hyper parameter

Additional entries may be included in the parameter file, but they will have no effect on the code.

Usage

The Basics: Fitting Your Light Curve

Once you have your data files formatted and named according to the preceding section, and you have made a corresponding input parameter file, fitting a light curve is done in one easy step. Simply type the following into the command line:

```
$ python exospec_main.py <input parameter filename>
```

Exospec will proceed through the fitting process according to the instructions you set in the input parameter file.

We find the tidiest way to work is to make a directory for your project, then make a subdirectory with the light curve files and a sub-directory for exospec output. Point the program towards these two directories via the lc_path and outout_dir entries of the input parameter file.

Suggestions for MPI

If you are running ExoSpec on (i) a cluster, or (ii) a multi-core PC with MPI library installed, you can get a substantial improvement in performance by exploiting the MPI feature of ExoSpec:

(i) on a cluster

Here an example of running ExoSpec on a cluster located at Princeton University (Adroit: https://www.princeton.edu/researchcomputing/computational-hardware/adroit/) is provided. Even though different clusters are configured in different ways, the principle to run ExoSpec would be similar.

An example job submission file (supported by Slurm) is shown below:

```
#!/bin/bash
# parallel job using 2 processors. and runs for 60  mins (max)
#SBATCH -N 2 # number of nodes you would like to use
#SBATCH --ntasks-per-node=16 # number of cores you would like to use per node
#SBATCH -t 01:00:00 # time limit for this job
# sends mail when process begins, and
# when it ends. Make sure you define your email
#SBATCH --mail-type=begin
#SBATCH --mail-type=end
#SBATCH --mail-type=fail
#SBATCH --mail-user=user@youremail.com
```

```
# Load openmpi environment
module load anaconda intel/16.0 openmpi/intel-16.0/1.10.2/64

# Make sure you are in the correct directory
cd ~/ExoplanetSpectra/bin

time srun --ntasks-per-node=1 --ntasks=$SLURM_JOB_NUM_NODES --cpu_bind=none
python exospec_main.py <input parameter filename>
```

Important information has been bolded or highlighted. A few things to note:

1. The user should be able to install all the packages ExoSpec is dependent on, and load them before running the code (e.g., module load anaconda).
2. Sometimes the code might not run due to incorrect MPI versions. It is hence suggested to coordinate with your cluster administrator to find out the proper MPI version (e.g., module load intel/16.0 openmpi/intel-16.0/1.10.2/64).
3. In order to run the code in hybrid scheme (MPI + openMP), the user needs to specify that only one node is running one copy of the code, and the number of the cores used on each node (e.g., extra flags of the srun command: --ntasks-per-node=1 --ntasks=$SLURM_JOB_NUM_NODES --cpu_bind=none).
4. Don't forget to set the "mpi_flag" to be "True", and "nthreads" to be the correct number of cores (e.g., 16 in this example) in the input parameter file.

(ii) on a multicore device with MPI ready

Simply type the following into the command line:

```
$ mpirun -np # python exospec_main.py <input parameter filename>
```

Where # is an integer specifying the number of cores/threads the user would like to use. In this case it is suggested to set the "nthreads" to be 1 in the input parameter file, and use all the cores to process lightcurves in parallel instead of running the emcee package only.

Additional Functionality of LightCurve Class

Within the main driving program of ExoSpec, the light curve class defined in lc_class.py stores all the data from the light curve files provided by the user. In addition to that, if imported into their own script, this class also enables the user to play around with their data. For example, if a light curve has a high sampling rate, the flux (brightness) plotted against time may be very noisey simply due to white (photon counting) noise. In order to smooth the curve, the user can choose a different time resolution to plot the flux against. Figure 1 gives an example of this.

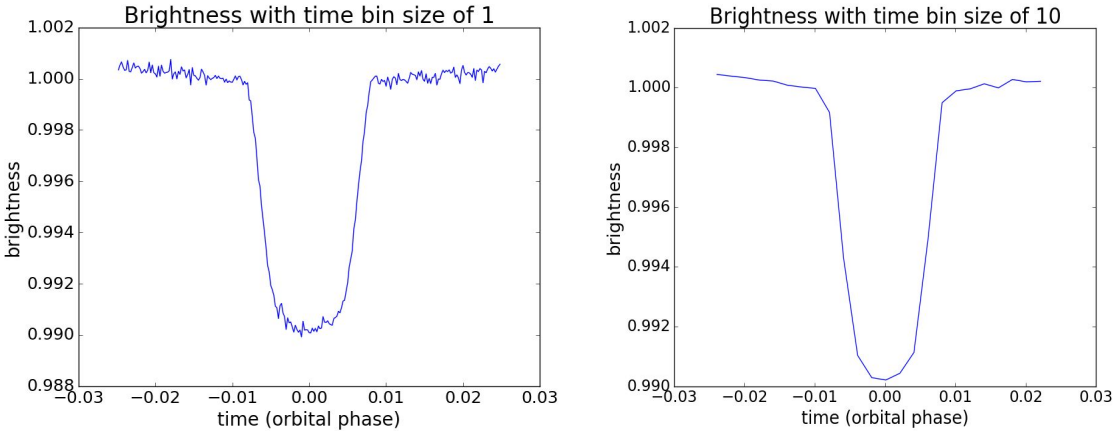

Figure 1. Left is a plot of the brightness where the time resolution is the same one as found in the light curve file. Right is a plot where the time resolution has been reduced by a factor of 10 (every 10 time points have been lumped together).

Another capability for the user of the light curve class object is to change the wavelength resolution of the measurements. Individual light curve files provide measurements for photons which fell in a fixed width around a central wavelength. The user hence provides multiple of these files, each one corresponding to a specific central wavelength. When the light curve class reads these files, it will create one light curve dictionary for each of these files. However, the user can choose to change the wavelength bin size and lump together the data of a sub-group of these files. For instance, if the user provides 8 light curve files but decides to set the wavelength bin size to 2 (every two light curve files are lumped together), only 4 light curve dictionaries for the resulting 4 new wavelength are going to be created.

Generating Synthetic Data

To generate synthetic data, there is a script available in exospec/tests/. In the same directory as this script, type the following on the command line

```
$ gen_syn_dat.py <output_directory> <number of time steps> <white noise scale> <red noise scale>
```

The above command will create output_directory if it does not already exist, a subdirectory named light_curves/ which contains light curves in 6 wavelength channels named synth_lc_1100.txt, synth_lc_950.txt, etc. These light curve files will have the times steps and white and red noise levels specified by the user. They will be saved in the format needed to for input data to exospec. Reasonable ranges for these two scale factors range from 0.0 to 0.1. Inside output_directory there will be two additional files created:

synth_expected_values.txt and visual.png. Synth_expected_values.txt contains a simple summary of the parameters used to generate the light curves and can be compared to fit results in benchmark testing. Visual.png is a simple plot of the light curves so user can quickly determine if the data looks as they expected. Immediately upon running this plot will open. One can then re-run the above command with different white noise scales and red noise scales until the light curve looks as desired. **Note that when generating synthetic data you must have an even number of time steps.**

References

Daniel Foreman-Mackey, David W. Hogg, Dustin Lang, Jonathan Goodman, “emcee: The MCMC Hammer”, (Submitted on 16 Feb 2012 (v1), last revised 25 Nov 2013 (this version, v4)): arXiv:1202.3665

Laura Kreidberg, “batman: BAsic Transit Model cAlculatioN in Python”, (Submitted on 29 Jul 2015 (v1), last revised 19 Aug 2015 (this version, v3)): arXiv:1507.08285

Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W. Hogg, Michael O'Neil, “Fast Direct Methods for Gaussian Processes”, arXiv:1403.6015 24 Mar 2014

N. P. Gibson, S. Aigrain, S. Roberts, T. M. Evans, M. Osborne and F. Pont, 2011, “A Gaussian process framework for modelling instrumental systematics: application to transmission spectroscopy”, https://arxiv.org/pdf/1109.3251v2.pdf

Appendix

```
# EXAMPLE USER INPUT PARAMETER FILE FOR EXOSPEC

# ----------------------------------------------------------------------------
# META DATA
lc_path = example/light_curves
output_dir = example/output
wave_bin_size = 1
n_errors = 4  # number of auxiliary measurements you want to include in the GP
kernel
ndim = 11

# ----------------------------------------------------------------------------
# GP model preferences, choose between "Custom" and "standard"
kernel_type = Custom

# ----------------------------------------------------------------------------
# MCMC instructions
nwalkers = 32   # emcee recommends 3-4 times the number of parameters
nburnin = 300  # these will be left off of plots and statistics
nsteps = 1000

# ----------------------------------------------------------------------------
# PARALLELIZATION FLAGS
mpi_flag = True # if use MPI
nthreads = 4 # set to num threads to use for each node

# ----------------------------------------------------------------------------
# POST-PROCESSING FLAGS (for interpreting MCMC results)
visualization = True # if want to see table output and plots
confidence = 95 #percentile confidence intervals in latex table
Separate_flag = True

# ----------------------------------------------------------------------------
# FIXED TRANSIT PARAMETERS (known by user from other methods)
t0 = 0.0     #time of inferior conjunction
per = 1.0   #orbital period
a = 15.0     #semi-major axis (in units of stellar radii)
inc = 87.0  #orbital inclination (in degrees)
ecc = 0.0   #eccentricity
w = 90.0   #longitude of periastron (in degrees)
limb_dark = quadratic   #limb darkening model, choose “quadratic or “nonlinear”

# ----------------------------------------------------------------------------
```

```
# TRANSIT PARAMETERS TO BE FIT
rp = 0.1   #planet radius (in units of stellar radii)
u0 = 0.5  #First limb darkening coefficient
u1 = 0.1  #Second limb darkening coefficient

# ----------------------------------------------------------------------------
# INITIAL GUESSES FOR MODEL
# order for p0: rp, u1, u2, ... uN, ksi, variance, gtime, gyerr, g3, g4, g5, g6, ... gN
p0 = 0.08 0.55 0.1 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
transit_par_names = rp u1 u2
gp_hyper_par_names = ksi variance g_time g_yerr g_fwhm g_sky g_pos g_airmass

# ----------------------------------------------------------------------------
# INITIAL GUESSES FOR KERNEL HYPER_PARAMETERS
kernel_a = 1
kernel_gamma = 0.1 0.1 0.1 0.1 0.1 0.1 # make sure that the length of the list is 2 +
n_errors
kernel_variance = 1

# ----------------------------------------------------------------------------
# PRIORS
rp_prior_lower = 0
rp_prior_upper = 1
u_prior_lower = 0
u_prior_upper = 0.99
kernel_a_prior_lower = 0        # NOTE: this should not be below zero
kernel_a_prior_upper = 1
kernel_gamma_prior_lower = 0     # NOTE: this should not be below zero
kernel_gamma_prior_upper = 10
kernel_variance_prior_lower = 0  # NOTE: this should not be below zero
kernel_variance_prior_upper = 10
```

Excerpt from example light curve file: “synth_lc_650.txt”
[0] time (phase) [1] normalized flux [2] ferr [3] fwhm [4] skynoise [5] position

-0.02500	1.00147	0.00011	0.00002	0.00000	9.76869
-0.024750	1.001859	0.000110	0.021482	0.021486	9.769478
-0.024500	1.001711	0.000110	0.042973	0.042969	9.770265
-0.024250	1.001850	0.000110	0.064448	0.064449	9.771057
-0.024000	1.001822	0.000110	0.085916	0.085923	9.771855
-0.023750	1.001398	0.000110	0.107378	0.107389	9.772658
-0.023500	1.001897	0.000110	0.128846	0.128845	9.773467
-0.023250	1.001827	0.000110	0.150296	0.150289	9.774282
-0.023000	1.001896	0.000110	0.171715	0.171720	9.775103
-0.022750	1.001715	0.000110	0.193126	0.193135	9.775930
-0.022500	1.001803	0.000110	0.214533	0.214533	9.776763
-0.022250	1.001722	0.000110	0.235915	0.235911	9.777602
-0.022000	1.001706	0.000110	0.257265	0.257267	9.778448
-0.021750	1.001329	0.000110	0.278618	0.278600	9.779300
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.