APPENDIX

CERES FUNCTIONS

In this appendix we briefly describe the functions that are used by the CERES pipelines. CERES contains two types of functions. The functions from the `GLOBALutils` module can be called by the pipelines of any instrument, and in this case the parameters of the functions can vary according to the properties of each instrument.The second set of functions correspond to the ones that change in structure from instrument to instrument.

Global Functions

- `JPLR0`: computes, from geographical coordinates, the geocentric radius of the observer and its distance to the declination 0 plane. It uses the SSephem package.

  Input

  -`lat`: geographic latitude of the telescope in degrees.
  -`altitude`: geographic altitude of the telescope in meters.

  Output

  -`GeoR`: geocentric radius at altitude of the observatory in meters.
  -`R0`: distance from the observatory to the declination 0 deg plane in meters.

- `obspos`: sets the observatory position with respect to geocenter in the coordinates (x,y,z) required by the SSephem package.

  Input

  -`longitude`: geographical longitud of the telescope in degrees.
  -`obsradius`: geocentric radius at altitude of the telescope in meters.
  -`R0`: distance from the observatory to the declination 0 deg plane in meters.

  Output

  -`obspos`: list with the x,y,z coordinates of the observer with respect to de geocenter.

- `PCoeff2`: given a reference image and the polynomial coefficients of the trace of a particular echelle order, this function determines the weights for the optimal extraction algorithm.

  Input

  -`pars`: list with a set of input parameters: (0) polynomial coefficients of the particular trace, (1) width measured in pixels of the echelle order in the cross-dispersion direction, (2) CCD readout noise, (3) CCD gain, (4) number of standard deviations above which flux values are considered cosmic rays or instrumental artifacts which are not used to compute the weights, (5) fraction of a pixel that is considered for the interpolation, (6) degree of the fitted polynomial, (7) optimal extraction algorithm, (8) initial pixel to be considered, (9) final pixel to be considered.

  Output

  -`FinalMatrix`: 2D array that contains the optimal weights in the regions of the particular echelle order, and zeros elsewhere.

- `obtain_P`: this is the function that calls the function `PCoeff2` and computes the weights for all the traces using the `multiprocessing` Python routine that enables parallelization.

  Input

  -`data`: 2D array that corresponds to the reference image used to compute the weights.
  -`trace_coefs`: 2D array that contains the polynomial coefficients for the traces of all the identified echelle orders.
  -`aperture`: width measured in pixels of the echelle orders in the cross-dispersion direction.
  -`RON`: CCD readout noise.
  -`GAIN`: CCD gain.
  -`NSigma`: number of standard deviations above which flux values are considered cosmic rays and artefacts which are not used to compute the weights.
  -`S`: fraction of a pixel that is considered for the interpolation.
  -`N`: degree of the fitted polynomial.
  -`Marsh_alg`: optimal extraction algorithm (0: Marsh, 1: Horne).
  -`min_col`: initial pixel to be considered.
  -`max_col`: final pixel to be considered.
  -`npools`: number of CPU cores that will be used.

  Output

  -`P_matrix`: 2D array that contains the optimal weights in the regions of the traces and zeros elsewhere.

- `getSimpleSpectrum2`: extracts a given echelle trace of an image without the optimal extraction algorithm.

  Input

  -`pars`: list with a set of input parameters: (0) polynomial coefficients of a given echelle trace, (1) width measured in pixels of the echelle order in the cross-dispersion direction, (2) initial pixel to be considered, (3) final pixel to be considered.

  Output

  -`spec`: 1D array containing the extracted spectrum of the corresponding echelle trace.

- `simple_extraction`: this is a function that calls `getSimpleSpectrum2` and extracts all the echelle orders using the `multiprocessing` Python routine that enables parallelization.

  Input

  -`data`: 2D array that corresponds to the image that will be extracted.
  -`coefs`: 2D array that contains the polynomial coefficients for the traces of all the identified echelle orders.
  -`ext_aperture`: width measured in pixels of the echelle orders in the cross-dispersion direction.
  -`min_extract_col`: initial pixel to be considered.
  -`max_extract_col`: final pixel to be considered.
  -`npools`: number of CPU cores that will be used.

  Output

  -`spec`: 2D array containing the extracted spectrum for all the identified echelle orders.

- `getSpectrum2`: extracts a particular echelle order of an image with an optimal extraction algorithm.

  Input

  -`pars`: list with a set of input parameters: (0) polynomial coefficients of a given echelle trace, (1) width in pixels of the echelle trace, (2) CCD readout noise, (3) CCD gain, (4) fraction of a pixel that is considered for the interpolation, (5) number of standard deviations above which flux values are considered cosmic rays and artefacts which are interpolated when extracted, (6) initial pixel to be considered, (7) final pixel to be considered.

  Output

  -`spec`: 2D array that contains the optimal extracted spectrum of the corresponding echelle order and the associated error.

- `optimal_extraction`: this is a function that calls `getSpectrum2` and optimally extracts all the orders using the multiprocessing Python routine that enables parallelization.

  Input

  -`data`: 2D array that corresponds to the image that will be extracted.
  -`Pin`: 2D array containing the weights obtained from the `obrain_P` function that will be used for the optimal extraction.
  -`coefs`: 2D array that contains the polynomial coefficients for the traces of all the identified echelle orders.
  -`ext_aperture`: width measured in pixels of the echelle orders in the cross-dispersion direction.
  -`RON`: CCD readout noise.
  -`GAIN`: CCD gain.
  -`MARSH`: fraction of a pixel that is considered for the interpolation.
  -`COSMIC`: number of standard deviations above which flux values are considered cosmic rays and instrumental artifacts which are interpolated when extracted.
  -`min_extract_col`: initial pixel to be considered.
  -`max_extract_col`: final pixel to be considered.
  -`npools`: number of CPU cores that will be used.

  Output

  -`spec`: 3D array containing the extracted spectrum for all the identified echelle orders and their corresponding errors.

- `XCor`: computes the cross-correlation function between a spectrum and a reference binary mask.

  Input

  -`spectra`: 3D array that corresponds to the output spectrum of the CERES pipeline.
  -`mask_l`: 1D array containing the lower wavelength limit of the absorption lines of the binary mask.
  -`mask_h`: 1D array containing the upper wavelength limit of the absorption lines of the binary mask.
  -`mask_w`: 1D array containing the weights of the absorption lines of the binary mask.
  -`vel`: central velocity of the cross-correlation function.
  -`lbary_ltopo`: if the input wavelengths have been corrected by the barycentric velocity, this parameter should

correspond to the multiplicative factor containing the barycentric correction.
-`vel_width`: velocity limits of the cross-correlation function.
-`vel_step`: velocity sampling of the cross-correlation function.
-`start_order`: first echelle order considered in the computation of the cross-correlation function.
-`spec_order`: dimension of the 3D array containing the continuum normalised spectrum.
-`iv_order`: dimension of the 3D array containing the inverse variances.
-`sn_order`: dimension of the 3D array containing the signal-to-noise ratio.

Output

-`velocities`: 1D array containing the velocities at which the cross-correlation function was sampled.
-`Xcor_full`: 2D array containing the cross-correlation function obtained for each echelle order.
-`sn`: 1D array containing the median signal-to-noise ratio of each echelle order.
-`nlines_used`: number of lines of the binary mask that were used to compute the cross-correlation function.
-`W`: 1D array containing the median weights of the cross-correlation function for each echelle order.

- `Average_CCF`: uses the output of `XCor` to combine the cross-correlation functions of the different orders into a single cross-correlation function.

  Input

  -`xc_full`: 2D array containing the cross-correlation function obtained for each echelle order.
  -`sn`: 1D array containing the median signal-to-noise ratio of each echelle order.
  -`start_order`: first echelle that is considered in the combination.
  -`sn_min`: echelle orders having a median signal-to-noise ratio lower than this value are not considered.

  Output

  -`xc_av`: 1D array containing the combined cross-correlation function.

- `XC_Final_Fit`: performs a Gaussian fit to the cross-correlation function. In the case of the presence of a secondary peak due to scattered moonlight, the function fits a double gaussian.

  Input

  -`X`: 1D array with the velocity values of the cross-correlation function.
  -`Y`: cross-correlation function.
  -`moonv`: radial velocity of the peak produced by the moonlight.
  -`moons`: predicted width of the peak produced by the moonlight.
  -`moon`: if `True` the function fits for the secondary peak. If `False` the function only fits a single peak.

  Output

  -`p1_gau`: parameters of the gaussian fit.
  -`predicted_gau`: gaussian fit.

- `get_rough_offset`: determines the drift in pixels in the dispersion direction between the reference wavelength calibration files and a particular extracted wavelength calibration spectrum.

  Input

  -`sc`: 2D array containing the extracted spectrum for each echelle order of the wavelength calibration source.
  -`files`: list with the paths to the reference wavelength calibration files.
  -`window`: extension in pixels of the cross-correlation function.

  Output

  -`delta`: drift in pixels.

- `Initial_Wav_Calibration`: this function uses a single order of the wavelength comparison spectrum to determine a wavelength solution for that particular order.

  Input

  -`filename`: path to the corresponding wavelength reference file for a given echelle order.
  -`spec`: extracted spectrum for the corresponding order of the wavelength calibration image (1D array).
  -`order`: number of the corresponding echelle order.
  -`wei`: 1D array that allows to give weights to the extracted spectrum for the determination of the centroids of the selected emission lines of the wavelength calibration source.
  -`porder`: polynomial degree of the single wavelength solution.
  -`rmsmax`: maximum rms allowed for the residuals of the wavelength solution in ms^{-1}. If the rms is higher than this value, the function will remove emission lines until this limit is reached.
  -`minlines`: minimum number of emission lines required to perform the fit of the wavelength solution.
  -`FixEnds`: if `True` the emission lines in both edges cannot be removed in the iterative process.
  -`Dump_Argon`: if `True` the emission lines identified as Ar (Argon) in the reference file are not considered in the

fit.
-`Cheby`: if `True`, the fit is performed with a Chebyshev polynomial. If `False` an ordinary polynomial is used.
-`rough_shift`: if the pixel position of the emission lines has significantly changed with respect to the positions defined in the reference wavelength file, they can be corrected with this value.
-`del_width`: number of pixels that are used to determine slight drifts between the observed spectrum and the reference file via cross-correlation.
-`binning`: binning factor of the wavelength calibration image in the dispersion direction.
-`line_width`: number of pixels that are considered in the gaussian fit of each emission line.

Output

-`coeffs_pix2wav`: list with the polynomial coefficients of the wavelength solution for the corresponding single order.
-`coeffs_pix2sigma`: coefficients of a polynomial that delivers the width of the emission lines as function of the pixel position.
-`pixel_centers`: 1D array containing the pixel position for each emission line obtained from the gaussian fit.
-`wavelengths`: 1D array containing the wavelength position for each emission lines used in the polynomial fit.
-`rmsms`: *rms* in ms^{-1} of the residuals of the polynomial fit of the single order wavelength solution.
-`residuals`: 1D array containing the residuals of the polynomial fit.
-`centroids`: 1D array containing the centroids of each emission line used in the polynomial fit.
-`sigmas`: 1D array containing the standard deviations of each emission line used in the polynomial fit.
-`intensities`: 1D array containing the intensities of each emission line used in the polynomial fit.

- `XCorPix`: computes a cross-correlation in pixel space between a single echelle order and a binary mask. It is used by the `Initial_Wav_Calibration` function to identify slight drifts of the wavelength comparison spectram with respect to the reference wavelength calibration files.

  Input

  -`spectra`: 1D array containing the spectrum of a single echelle order.
  -`mask_l`: 1D array containing lower pixel position of each line of the binary mask.
  -`mask_h`: 1D array containing higher pixel position of each line of the binary mask.
  -`del0`: central pixel position of the cross-correlation function.
  -`del_width`: extension in pixels of the cross-correlation function.
  -`del_step`: sampling in pixels of the cross-correlation function.

  Output

  -`Xcor`: computed cross-correlation function.
  -`deltas`: pixel values at which the cross-correlation function is evaluated.

- `rms_ms`: computes the *rms* deviation of the best fit single-order wavelength solution in ms^{-1} and its residuals.

  Input

  -`coeffs_pix2wav`: polynomial coefficients of the wavelength solution for the corresponding single order.
  -`pixel_centers`: 1D array containing the pixel positions for each emission line obtained from the gaussian fit.
  -`wavelengths`: 1D array containing the wavelength positions for each emission line used in the polynomial fit.
  -`npix`: number of pixels of the spectrum.
  -`Cheby`: if `True`, the fit is performed with Chebyshev polynomial. If `False` a ordinary polynomial is used.

  Output

  -`rms_ms`: *rms* in ms^{-1} of the residuals of the polynomial fit of the single order wavelength solution.
  -`residuals`: 1D array containing the residuals of the polynomial fit.

- `Cheby_Fit`: fits Chebyshev polynomials to `y` as a function of `x`.

  Input

  -`x`: 1D array with the values at which the values to fit are sampled.
  -`y`: 1D array with the values that will be fitted.
  -`order`: degree of the Chebyshev polynomial.
  -`npix`: number of pixels of the spectrum.

  Output

  -`p1`: list containing the coefficients of the fitted Chebyshev polynomial.

- `Cheby_eval`: given a set of coefficients, this function evaluates a Chebyshev polynomial at the given `x` values.

  Input

  -`p`: list containing the polynomial coefficients.
  -`x`: 1D array containing the values at which the polynomial will be evaluated.
  -`npix`: number of pixels of the spectrum.

Output

-`ret_val`: Chebyshev polynomial evaluated at the given `x` values.

- `LineFit_SingleSigma`: fits a series of Gaussians simultaneously, given a set of input pixels, sigmas (Gaussian standard deviations), and intensities.

  Input

  -`X`: 1D array containing the sampling values.
  -`Y`: 1D array containing the signal where the gaussians will be fitted.
  -`B`: 1D array containing an estimation of the background of the signal.
  -`mu`: 1D array containing an initial guess of the positions of the gaussians.
  -`sigma`: 1D array containing the width of the gaussians.
  -`weight`: 1D array containing the weights for each gaussian.

  Output

  -`p_output`: 2D array containing the coefficients of all the fitted gaussians.
  -`success`: `True` if the fit was succesful.

- `Fit_Global_Wav_Solution`: computes the global wavelength solution as function of the pixel position and order number using the emission lines identified with `Initial_Wav_Calibration`.

  Input

  -`pix_centers`: 1D array containing the pixel positions for each emission line.
  -`wavelengths`: 1D array containing the wavelength positions for each emission line.
  -`orders`: 1D array containing the order numbers given by the grating equation for each emission line.
  -`Wgt`: 1D array containing the weights for each emission line.
  -`p0`: list containing an initial guess for the coefficients of the fit.
  -`minlines`: minimum number of emission lines required to perform the fit of the wavelength solution.
  -`maxrms`: maximum *rms* allowed for the residuals of the wavelength solution in ms^{-1}. If the *rms* is higher than this value, the function will remove emission lines until this limit is reached.
  -`order0`: real order number given by the grating equation for the reddest extracted echelle order.
  -`total`: number of extracted echelle orders.
  -`npix`: number of pixels in the dispersion direction.
  -`Cheby`: if `True`, the fit is performed with Chebyshev polynomial. If `False` a ordinary polynomial is used.
  -`nx`: number of polynomial coefficients in the pixel direction.
  -`nm`: number of polynomial coefficients in the direction of the echelle orders.

  Output

  -`p1`: polynomial coefficients of the global wavelength solution.
  -`pix_centers`: 1D array containing the pixel position for the emission lines that were not removed in the fitting procedure.
  -`orders`: 1D array containing the order number for the emission lines that were not removed in the fitting procedure.
  -`wavelengths`: 1D array containing the wavelength position for the emission lines that were not removed in the fitting procedure.
  -`I`: 1D array containing the positions of the input array that were not removed in the fitting procedure.
  -`rms_ms`: *rms* in ms^{-1} of the residuals of the polynomial fit of the global wavelength solution.
  -`residuals`: 1D array containing the residuals of the polynomial fit.

- `Global_Wav_Solution_vel_shift`: determines the velocity shift between a reference global wavelength solution (a list of polynomial coefficients obtained with `Fit_Global_Wav_Solution`) and a set of emission lines identified for another wavelength calibration image.

  Input

  -`pix_centers`: 1D array containing the pixel positions for each emission line.
  -`wavelengths`: 1D array containing the wavelength positions for each emission line.
  -`orders`: 1D array containing the order numbers given by the grating equation for each emission lines.
  -`Wgt`: 1D array containing the weights for each emission line.
  -`p_ref`: list containing the polynomial coefficients of the reference wavelength solution.
  -`minlines`: minimum number of emission lines required to perform the fit of the wavelength solution.
  -`maxrms`: maximum *rms* allowed for the residuals of the wavelength solution in ms^{-1}. If the *rms* is higher than this value, the function will remove emission lines until this limit is reached.
  -`ntotal`: number of extracted echelle orders.
  -`npix`: number of pixels in the dispersion direction.

-`Cheby`: if `True`, the fit is performed with Chebyshev polynomial. If `False` an ordinary polynomial is used.
-`nx`: number of polynomial coefficients in the pixel direction.
-`nm`: number of polynomial coefficients in the direction of the echelle orders.

Output

-`p1`: fractional velocity drift of the new wavelength calibration image with respect to the reference wavelength solution. To express it in units of ms^{-1}, it has to be multiplied by 299.792458.
-`pix_centers`: 1D array containing the pixel position for the emission lines that were not removed in the fitting procedure.
-`orders`: 1D array containing the order number for the emission lines that were not removed in the fitting procedure.
-`wavelengths`: 1D array containing the wavelength position for the emission lines that were not removed in the fitting procedure.
-`I`: 1D array containing the positions of the input array that were not removed in the fitting procedure.
-`rms_ms`: *rms* in ms^{-1} of the residuals of the polynomial fit of the global wavelength solution.
-`residuals`: 1D array containing the residuals of the polynomial fit.

- `Calculate_chebs`: computes a set of Chebyshev functions.

  Input

  -`x`: 1D array containing the pixel values at which the Chebyshev functions will be computed.
  -`m`: order numbers at which the chebyshev functions will be computed.
  -`order0`: real order number given by the grating equation for the reddest extracted echelle order.
  -`total`: number of extracted echelle orders.
  -`npix`: number of pixels in the dispersion direction.
  -`nx`: number of polynomial coefficients in the pixel direction.
  -`nm`: number of polynomial coefficients direction of the echelle orders.

  Output

  -`coefs`: list containing the Chebyshev functions.

- `Joint_Polynomial_Cheby`: evaluates the Chebyshev polynomials using a set of coefficients and Chebyshev functions.

  Input

  -`p`: coefficients of the Chebyshev polynomial that will be evaluated.
  -`chebs`: list containing the Chebyshev functions that were obtained with `Calculate_chebs`.
  -`nx`: number of polynomial coefficients in the pixel direction.
  -`nm`: number of polynomial coefficients in the direction of the echelle orders.

  Output

  -`ret_val` Evaluation of the Chebyshev polynomials.

- `fp_base`: this function cleans a Fabry-Perot extracted spectrum by estimating its background.

  Input

  -`f`: Fabry-Perot spectrum for a single echelle order (1D array).
  -`n`: degree of the polynomial that is fitted to estimated the background.

  Output

  -`ret`: Fabry-Perot spectrum for a single echelle order after subtracting the background.

- `ccf_fp`: this function determines the velocity shift between two Fabry-Perot spectra via the computation of the cross-correlation function.

  Input

  -`fp`: 1D array containing the extracted spectrum of single order of a Fabry-Perot image.
  -`fpr`: 1D array containing the reference Fabry-Perot spectrum.
  -`p1`: polynomial coefficients of the wavelength solution of the reference Fabry-Perot spectrum.
  -`order`: corresponding echelle order number of the given spectrum.
  -`order0`: real order number given by the grating equation for the reddest extracted echelle order.
  -`ntotal`: number of extracted echelle orders.
  -`npix`: number of pixels in the dispersion direction.
  -`nx`: number of polynomial coefficients in the pixel direction.
  -`nm`: number of polynomial coefficients in the order direction.

  Output

  -`pg`: velocity shift in ms^{-1} of the input Fabry-Perot spectrum with respect to the reference Fabry-Perot spectrum.

- `simbad_query_obname` performs an online query to SIMBAD and obtains the spectral type of a given object name.

  Input

  -`obname`: name of the object.

  Output

  -`query_success`: if `True` the query returned a valid entry.
  -`sp_type_query`: spectral type of the object as returned by SIMBAD.

- `simbad_query_coords` this function performs an online query to SIMBAD and obtains the spectral type of an object given its coordinates.

  Input

  -`ra`: right ascention of the object.
  -`dec`: declination of the object.

  Output

  -`query_success`: if `True` the query returned a valid entry.
  -`sp_type_query`: spectral type of the object as returned by SIMBAD.

- `XC_ThAr`: computes the cross-correlation function in pixel space between two wavelength comparison spectra.

  Input

  -`thar_ref`: 1D array corresponding to the wavelength comparison spectrum for which the shift will be computed.
  -`thar_comp`: 1D array corresponding to the reference wavelength comparison spectrum.
  -`pixel_span`: number of pixels at the beginning and end of the spectrum that will be not considered in the computation of the cross-correlation function.

  Output

  -`XCor`: cross-correlation function computed between the two wavelength calibration spectra.
  -`shifts`: pixel values at which the cross-correlation function is evaluated.

- `XC_Gau_Fit`: fits a Gaussian to a cross-correlation function obtained from `XC_ThAr`.

  Input

  -`X`: 1D array that contains the pixel values at which the cross-correlation function is evaluated.
  -`Y`: 1D array that contains the cross-correlation function.

  Output

  -`p1`: list that contains the coefficients of the gaussian fit to the cross-correlation function.

- `get_cont_single`: performs the continuum normalisation of a single echelle order by rejecting absorption lines and fitting a polynomial to the continuum. In order to identify the continuum regions of the spectrum, the *rms* of the flux is computed at many spectral zones. The spectral zones where the *rms* is greater than a fixed number of times the value expected from the estimated measurement errors are considered as regions containing an important number of absorption lines and will no be considered for the polynomial fit of the continuum.

  Input

  -`W`: 1D array containing the wavelength values at which the spectrum was measured.
  -`F`: 1D array containing the flux measurements for the spectrum that will be normalised.
  -`E`: 1D array containing the error associated to the flux of the spectrum that will be normalised.
  -`nc`: degree of the fitted polynomial.
  -`ll`: spectral regions with residual values below `-ll` times the *rms* of the residuals are not considered in the next iteration of the fit.
  -`lu`: spectral regions with residual values higher than `lu` times the *rms* of the residuals are not considered in the next iteration of the fit.
  -`span`: number of pixels that will be used to compute the *rms* of the different spectral zones that are compared with the expected measurement errors for rejecting regions containing absorption lines.
  -`fact`: the spectral regions having *rms* values greater than `fact` times the expected measurements errors are rejected.
  -`frac`: a minimum number of pixels is mandatory for performing the fit. This minimum number is `frac` times the total number of pixels of the spectrum.

  Output

  -`coef`: list with the polynomial coefficients of the fit to the continuum.

- `get_cont`: performs continuum normalisation for all the orders of an echelle spectra. This function should only be used if the edges of contiguous echelle orders present a small degree of mismatch, because, in order to find the continuum for each order, information of the contiguous orders is also used.

  Input

  `-W`: 2D array containing the wavelength values at which the spectrum was measured.
  `-F`: 2D array containing the flux measurements for the spectrum that will be normalised.
  `-nc`: degree of the fitted polynomials.
  `-ll`: spectral regions with residual values below `-ll` times the *rms* of the residuals are not considered in the next iteration of the fit.
  `-lu`: spectral regions with residual values higher than `lu` times the *rms* of the residuals are not considered in the next iteration of the fit.
  `-frac`: a minimum number of pixels is mandatory for performing the fit. This minimum number is `frac` times the total number of pixels of the spectrum.

  Output

  `-coefs`: 2D array with the polynomial coefficients of the fit to the continuum for all the echelle orders.

- `get_lunar_props`: determines the properties of the Moon at the time when the observations were performed. In uses the `ephem` Python package

  Input

  `-gobs`: object containing the properties of the observatory (geographical coordinates, date of observation).
  `-Mcoo`: dictionary containing the coordinates (RA, DEC) of the moon at the moment of the observation.
  `-Mp`: dictionary containing the (x,y,z) position and velocity of the moon with respect to the barycentre of the solar system obtained with the SSephem package.
  `-Sp`: dictionary containing the (x,y,z) position and velocity of the sun with respect to the barycentre of the solar system obtained with the SSephem package.
  `-res`: dictionary containing the barycentric correction in the direction of the moon obtained with the SSephem package.
  `-RA`: right ascension of the observed object.
  `-DEC`: declination of the observed object.

  Output

  `-lunation`: illumination of the moon at the time of the observation (percentage).
  `-moon_state`: waining or crescent moon.
  `-moonsep2`: angular separation between the moon and the observed object.
  `-moonvel`: radial velocity predicted for the spectrum of the sun reflected on the moon.

- `calc_bss`: computes the bisector span for the peak of a cross-correlation function. Two regions are defined in the cross-correlation peak at different depths.

  Input

  `-vels`: 1D array containing the velocities at which the cross-correlation function is sampled.
  `-xc_av`: 1D array containing the cross-correlation function values.
  `-bot_i`: lower depth limit of the lower defined region of the cross-correlation function.
  `-bot_f`: upper depth limit of the lower defined region of the cross-correlation function.
  `-top_i`: lower depth limit of the upper defined region of the cross-correlation function.
  `-top_f`: upper depth limit of the upper defined region of the cross-correlation function.

  Output

  `-span`: velocity difference of the mean bisectors of the two defined regions.
  `-der_bottom`: mean velocity value for the bisectors in the lower region of the cross-correlation function.
  `-der_top`: mean velocity value for the bisectors in the upper region of the cross-correlation function.
  `-slope`: slope obtained from a linear fit between the bisectors obtained in both regions.
  `-stat`: if equal to 1 the determination of the bisector span was unsuccessful.

- `convolve`: degrades an observed spectrum to a lower resolution.

  Input

  `-wav`: 1D array containing the wavelength values.
  `-flx`: 1D array with the flux measurements of the spectrum.
  `-R`: Resolution to which the spectrum will be degraded.

  Output

  `-NF`: 1D array containing the degraded spectrum.

- `cor_thar`: computes the cross-correlation function between a particular echelle order and a binary mask generated from a reference wavelength calibration file.

  Input

  -`spec`: 1D array corresponding to the observed spectrum.
  -`span`: width in pixels that the cross-correlation function will have.
  -`filename`: path to the reference wavelength calibration file.
  -`binning`: binning in the dispersion direction of the observed spectrum.

  Output

  -`CCF`: cross-correlation function.
  -`DEL`: pixel shifts at which the the cross-correlation function was calculated.

- `plot_CCF`: generates the ".pdf" files containing the plots of the cross-correlation function.

  Input

  -`xc_dict`: dictionary containing the properties of the cross-correlation function.
  -`moon_dict`: dictionary containing the properties of the moon at the moment of the observation.
  -`path`: path and name of the output figure.

- `get_disp`: searches in the auxiliary file for the dispersion value that broaden the lines of the binary mask that are used to compute the radial velocity of the spectrum.

  Input

  -`obname`: name of the object as specified in the auxiliary file.
  -`reffile`: name of the auxiliary file containing the dispersion value.

  Output

  -`disp`: dispersion value.

- `get_mask_reffile`: searches in the auxiliary file for the name of the binary mask that will be used to compute the radial velocity of the spectrum.

  Input

  -`obname`: name of the object as specified in the auxiliary file.
  -`reffile`: name of the auxiliary file containing the dispersion value.
  -`base`: path to the directory containing the binary masks.

  Output

  -`sp_type`: spectral type of the mask that will be used to compute the cross-correlation function.
  -`xc_mask`: path to the corresponding binary mask.

- `get_mask_query`: based on a SIMBAD query, this function determines which of the binary masks should be used to determine the radial velocity of the observed star.

  Input

  -`sp_type_query`: spectral type of the object as returned by SIMBAD.
  -`base`: path to the directory containing the binary masks.

  Output

  -`sp_type`: spectral type of the mask that will be used to compute the cross-correlation function.
  -`mask`: path to the corresponding binary mask.

- `get_mask_teff`: based on the estimated stellar effective temperature, this function determines which of the binary masks should be used to determine the radial velocity of the observed star.

  Input

  -`T_eff`: estimated stellar effective temperature.
  -`base`: path to the directory containing the binary masks.

  Output

  -`sp_type`: spectral type of the mask that will be used to compute the cross-correlation function.
  -`mask`: path to the corresponding binary mask.

- `iau_cal2jd`: transforms calendar dates into Julian days.

  Input

  -`IY`: year.
  -`IM`: month.
  -`ID`: day.

  Output

  -`DJMO`: truncated julian day.
  -`DJM`: Julian day.

- `update_header`: updates the header of a fits object by adding a new entry or replacing and existing one.

  Input

  -`hdu`: input image structure containing data and header.
  -`k`: keyword of a new header entry.
  -`v`: value of the new header entry.

  Output

  -`hdu`: image structure with the modified header.

- `get_dark_times`: taking as input a list containing the names of all the dark frames, this function returns an array with the different exposure times without repetition.

  Input

  -`darks`: list with the names of the dark images.
  -`key`: keyword used in the header for the exposure time.

  Output

  -`times`: list with the different exposure times.

- `n_Edlen`: modified Edlen equation for computing the refractive index of light.

  Input

  -`l`: 1D array containing the input wavelengths.

  Output

  -`n`: refractive index at the corresponding wavelengths.

- `ToAir`: this function transforms an array of wavelength values from vacuum to air.

  Input

  -`l`: wavelength in vacuum.

  Output

  -`l`: wavelength in air.

- `ToVacuum`: transforms an array of wavelength values from air to vacuum.

  Input

  -`l`: wavelength in air.

  Output

  -`l_prev`: wavelength in vacuum.

- `getcoords`: determines the coordinates at a given time given an auxiliary file that specifies the J2000 coordinates and proper motions.

  Input

  -`obname`: object name as specified in the auxiliary file.
  -`mjd`: modified julian date.
  -`filen`: path to the text file containing the J2000 coordinates and proper motions of the target.

  Output

  -`RA`: right ascension of the object at the selected mjd.
  -`DEC`: declination of the object at the selected mjd.

- `get_them`: identifies echelle orders and traces them.

  Input

  `-sc`: 2D array with the image that will be used to identify the orders.
  `-exap`: approximate width in pixels of the orders.
  `-ncoef`: number of coefficients of the polynomial that will be fitted to the centroid of the traces.
  `-maxorders`: integer that if it is different than -1 will truncate the number of identified orders to that value.
  `-startfrom`: initial pixel to consider in the cross-dispersion direction from which the orders can be identified.
  `-endat`: final pixel to consider in the cross-dispersion direction.
  `-nsigmas`: identified peaks with amplitude values lower than this number of times the standard deviation of the flux in the inter-order regions will be rejected. `-mode`: if equal to 1, a single gaussian will be fitted to identify the centroid of the trace. if equal to 2, a double gaussian will be fitted to identify the centroid of the trace, which can be done if the spectrograph uses and image slicer.

  Output

  `-acoefs`: 2D array with the polynomial coefficients for each identified echelle order.
  `-ncoefs`: number of identified orders.

- `good_orders`: this function takes the 2D array of the polynomial coefficients of the identified orders and removes the orders that go out of the image dimensions.

  Input

  `-coef`: 2D array with the polynomial coefficients for each identified echelle order.
  `-nord`: initial number of echelle orders.
  `-ny`: number of pixels in the normal dispersion direction of the image.
  `-nx`: number of pixels in the cross-dispersion direction of the image.
  `-ext_aperture`: width of the traces in pixels.

  Output

  `-acoefs`: 2D array with the polynomial coefficients of the orders that were not removed.
  `-ncoefs`: number of echelle orders.

- `get_zero_order_number`: given an approximate wavelength value for the central pixel of each echelle order, this function returns the real order number of the first order given by the grating equation.

  Input

  `-ords`: 1D array enumerating the echelle orders.
  `-wavs`: 1D array with the central wavelength value of each order

  Output

  `-o0`: real order number for the first input order.

- `Mesh`: merges the coefficients of 2 different arrays of echelle orders into a single one.

  Input

  `-ycpoly_ob`: first 2D array of coefficients.
  `-ycpoly_co`: second 2D array of coefficients.

  Output

  `-npoly`: merged 2D array with all the polynomial coefficients.

- `get_scat`: estimates the background of an image by interpolating the counts in-between the echelle orders.

  Input

  `-sc`: 2D array with the input image.
  `-lim`: 2D array with the traces of each order.
  `-span`: approximate width in pixels of the echelle orders.
  `-type`: if `type` = "median", then the median value of the inter-order regions will be used to perform the interpolation. if `type`="min", the minimum value of the inter order regions will be used.
  `-allow_neg`: if `True`, the background image can take negative values. if `False`, negative values are replaced with zeros.

  Output

  `-scat`: 2D array with the output background image

- `FlatNormalize_single`: normalises each extracted order of a fibre flat by its maximum.

  Input

  -`S_flat`: array with extracted flat spectra.
  -`mid`: central pixel of each order.
  -`span`: numbers of pixels from the central pixel that are used to search for the maximum.

  Output

  -`S_flat_n`: array with the normalised flat spectrum.
  -`max_vals`: array with the determined maxima of each order

- `retrace`: updates an initial set of echelle traces by cross correlating them with a new image and determining a drift in the cross-dispersion direction.

  Input

  -`dat`: 2D array corresponding to the new image that will be used to compute the drift with respect to the reference traces.
  -`c_all`: 2D array with the polynomial coefficients of the traces.

  Output

  -`c_new`: 2D array with the new polynomial coefficients of the traces after applying the shift.
  -`shift`: obtained drift of the traces in the cross-dispersion direction.

In addition to the functions contained in the `GLOBALutils` module, the function that performs the estimation of the atmospheric parameters is also a global function in the sense that the same function works for the different spectrographs. This function is located in the `correlation module` and is called `CCF`.

- `CCF`

  Input

  -`spec`: 3D array that corresponds to the output spectrum of the pipeline.
  -`model_path`: path to the directory containing the synthetic spectra.
  -`Li`: initial wavelength of the spectrum that will be considered to perform the estimation of the stellar parameters.
  -`Lf`: final wavelength of the spectrum that will be considered to perform the estimation of the stellar parameters.
  -`pools`: number of CPU cores that will be used.

  Output

  -`intert`: effective temperature of the synthetic spectrum that produced the maximum cross-correlation with the observed spectrum.
  -`interg`: $\log g$ of the synthetic spectrum that produced the maximum cross-correlation with the observed spectrum.
  -`interz`: metallicity of the synthetic spectrum that produced the maximum cross-correlation with the observed spectrum.
  -`interv`: projected rotational velocity of the synthetic spectrum that produced the maximum cross-correlation with the observed spectrum.
  -`elfin`: radial velocity of the synthetic spectrum that produced the maximum cross-correlation with the observed spectrum.
  -`maximCCF`: cross-correlation value for the model that produced the maximum cross-correlation.

OTHER FUNCTIONS

In this section of the appendix we describe the functions that have important structural changes for different instruments. For this reason, some of the output and input parameters that are described below are only present for some particular instruments and they are identified as optional.

- `FileClassify`: analyses the header of all the images and classifies them according to their type.

  Input

  -`dir`: path to the directory containing the unprocessed images.
  -`log`: name of the log file that will be created with the information of each image.
  -`binning`: binning of the images that will be processed (optional).
  -`mode`: observing mode of the images that will be processed (optional).
  -`amps`: only images registered with these particular amplifiers will be processed (optional).

  Output

  -`biases`: list with the names of all the bias frames that will be processed (optional).
  -`flats`: list with the names of all the bias frames that will be processed (optional).
  -`science`: list with the names of all the science frames that will be processed (optional).
  -`thars`: list with the names of all the wavelength calibration frames that will be processed (optional).
  - Several other type of images can be identified.

- `OverscanTrim`: computes and applies the overscan of the image and also trims it. Given that the position and number of overscan regions can be significantly different for different instruments, each pipeline has its own function to perform this process.

  Input

  `-d`: 2D array containing to the image to be processed.

  Output

  `-newdata`: 2D array containing to the processed image.

- `MedianCombine`: combines a set of images using their median values. Given that it uses the `OverscanTrim` function, it is defined for each instrument.

  Input

  `-ImgList`: list containing the names of the images that will be combined.
  `-bias`: 2D array containing the master bias frame (optional).
  `-dark`: 2D array containing the master dark frame (optional).

  Output

  `-d`: 2D array containing the median combined image.
  `-ronoise`: CCD readout noise of the combined image.
  `-gain`: CCD gain of the combined image.

- `mjd_fromheader`: returns the modified julian day of a particular image by taking into account the exposure time and exposure meter if available. Given that the format in which the time of the observation reported in the header varies for different instruments, this function changes for different pipelines.

  Input

  `-h`: dictionary containing the header of the image.

  Output

  `-mid`: modified julian day.
  `-mjd0`: truncated modified julian day.