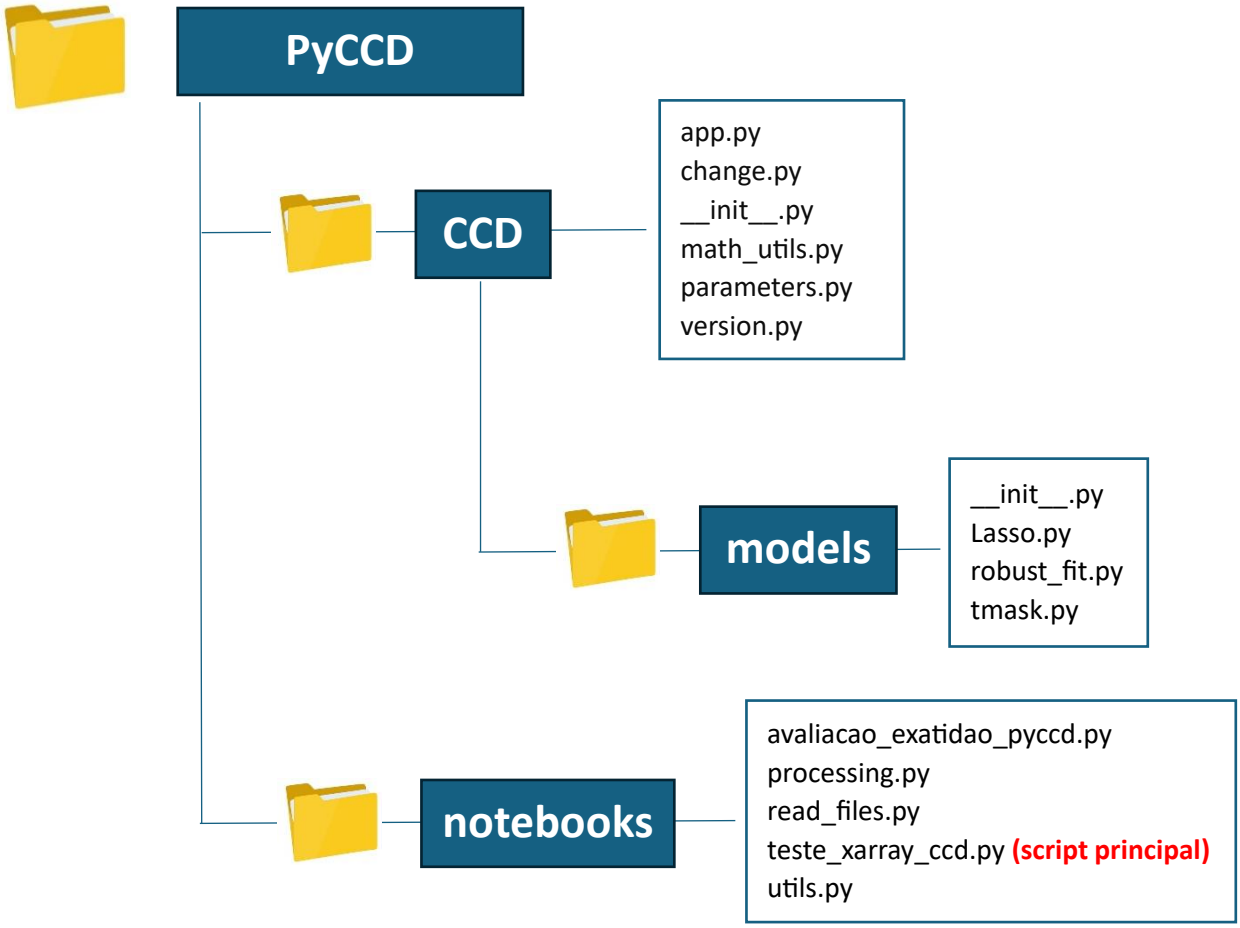
Entregável 1.1. Código modular PyCCD

Índice

1. PyCCD

Os scripts em Python do algoritmo PyCCD estão disponíveis em: https://github.com/manuelcampagnolo/S2CHANGE/tree/main/scripts/pyccd_theia/

Os scripts encontram-se divididos em pastas tendo a seguinte configuração:



2. CCD

2.1. app.py

```
1  ============ 'app.py' ============
2
3  """ Main bootstrap and configuration module for pyccd.  Any module that 4  requires configuration or services should import app and obtain the
5  configuration or service from here.
6
7      app.py enables a very basic but sufficient form of loose coupling
8      by setting names of services & configuration once, then allowing other modules
9      that require these services/information to obtain them by name rather than
10     directly importing or instantiating.
11
12 Module level constructs are only evaluated once in a Python application's 13  lifecycle, usually at the time of first import. This pattern is borrowed
14     from Flask.
```

```
    """
    import hashlib

from ccd import parameters


# Simplify parameter setting and make it easier for adjustment class Parameters(dict):
                    def __init__(self, params):

    super(Parameters, self).__init__(params)
            def __getattr__(self, name):
            if name in self:  return self[name]         else:
                                    raise AttributeError('No such attribute: ' + name)

                    def __setattr__(self, name, value):
                    self[name] = value

            def __delattr__(self, name):
            if name in self:  del self[name]            else:
                                    raise AttributeError('No such attribute: ' + name)


# Don't need to be going down this rabbit hole just yet   # mainly here as
reference def numpy_hashkey(array):
                        return hashlib.sha1(array).hexdigest()


    # This is a string.fully.qualified.reference to the fitter function.
    # Cannot import and supply the function directly or we'll get a
    # circular dependency
    FITTER_FN = 'ccd.models.lasso.fitted_model'


    def get_default_params():
    return Parameters(parameters.defaults)
```

2.2. change.py

```
========== 'change.py' ==========

    """
    Methods used by the change detection procedures. There should be no default
    values for input arguments, as all values should be supplied by the calling
    method.

These should be as close to the functional paradigm as possible.  """
import logging
import numpy as np  from scipy.stats
import chi2
 from ccd.models import
lasso
from ccd.math_utils import sum_of_squares

log = logging.getLogger(__name__)

```

```
19
20      def stable(models, dates, variogram, t_cg, detection_bands):
21      """Determine if we have a stable model to start building with 22
23                              Args:
24                              models: list of current representative/fitted models
25                              variogram: 1-d array of variogram values to compare against for the
26                              normalization factor
27                              dates: array of ordinal date values
28                              t_cg: change threshold
29                              detection_bands: index locations of the spectral bands that are used
30                              to determine stability
31
32                      Returns:
33                      Boolean on whether stable or not
34                      """
35                      # This could be written differently, or more performant using numpy in the
36                      # future
37                      check_vals = []
38                      for idx in detection_bands:
39                      rmse_norm = max(variogram[idx], models[idx].rmse)
40                      slope = models[idx].fitted_model.coef_[0] * (dates[-1] - dates[0]) 41
42                                      check_val = (abs(slope) + abs(models[idx].residual[0]) +
43                                      abs(models[idx].residual[-1])) / rmse_norm
44
45                              check_vals.append(check_val)
46
47              euc_norm = sum_of_squares(np.array(check_vals))
48              log.debug('Stability norm: %s, Check against: %s', euc_norm, t_cg) 49
50                      return euc_norm < t_cg
51
52
53              def change_magnitude(residuals, variogram, comparison_rmse):
54              """
55              Calculate the magnitude of change for multiple points in time.
56
57                      Args:
58                      residuals: predicted - observed values across the desired bands,
59                      expecting a 2-d array with each band as a row and the observations60                      as columns
61                              variogram: 1-d array of variogram values to compare against for the
62                              normalization factor
63                              comparison_rmse: values to compare against the variogram values64
65                      Returns:
66                      1-d ndarray of values representing change magnitudes
67                      """
68                      rmse = np.maximum(variogram, comparison_rmse) 69
70      magnitudes = residuals / rmse[:, None] 71
72                          change_mag = sum_of_squares(magnitudes, axis=0)
73
74                              log.debug('Magnitudes of change: %s', change_mag)
75
76                      return change_mag
77
78
79              def calc_residuals(dates, observations, model, avg_days_yr):
80              """
81              Calculate the residuals using the fitted model.
```

```
82
83                  Args:
84                  dates: ordinal dates associated with the observations
85                  observations: spectral observations
86                  model: named tuple with the scipy model, rmse, and residuals87
88                  Returns:
89                  1-d ndarray of residuals
90                  """
91                  # This needs to be modularized in the future.
92                  # Basically the model object should have a predict method with it.
93                  return np.abs(observations - lasso.predict(model, dates, avg_days_yr))
94
95
96          def detect_change(magnitudes, change_threshold):
97          """
98          Convenience function to check if the minimum magnitude surpasses the
99          threshold required to determine if it is change.
100
101                 Args:
102                 magnitudes: change magnitude values across the observations
103                 change_threshold: threshold value to determine if change has occurred
104
105                 Returns:
106                 bool: True if change has been detected, else False
107                 """
108
109         # print("Magnitudes:", magnitudes)
110         # print("Change Threshold:", change_threshold) 111
112                     return np.min(magnitudes) > change_threshold
113
114
115         def detect_outlier(magnitude, outlier_threshold):
116         """
117         Convenience function to check if any of the magnitudes surpass the
118         threshold to mark this date as being an outlier
119
120   This is used to mask out values from current or future processing 121
122                 Args:
123                 magnitude: float, magnitude of change at a given moment in time
124                 outlier_threshold: threshold value
125
126                 Returns:
127                 bool: True if these spectral values should be omitted
128                 """
129                 return magnitude > outlier_threshold
130
131
132    def find_time_index(dates, window, meow_size, day_delta):
133    """Find index in times at least one year from time at meow_ix. 134      Args:
135                        dates: list of ordinal day numbers relative to some epoch,
136                        the particular epoch does not matter.
137                        window: index into times, used to get day number for comparing
138                        times139          meow_size: minimum expected observation window needed to
140                        produce a fit.
141                        day_delta: number of days required for a years worth of data,142              defined to be 365
143                        Returns:
144                        integer: array index of time at least one year from meow_ix,
```

```
145                         or None if it can't be found.
146                         """
147
148     # If the last time is less than a year, then iterating through 149             # times to find an index is
futile.
150                 if not enough_time(dates, day_delta=day_delta):
151                 log.debug('Insufficient time: %s', dates[-1] - dates[0])
152                   return None
153
154     if window.stop: 155         end_ix =
window.stop 156             else:
157                          end_ix = window.start + meow_size
158
159     # This seems pretty naive, if you can think of something more 160       # performant and elegant,
have at it!
161                 while end_ix < dates.shape[0] - meow_size:
162                 if (dates[end_ix]-dates[window.start]) >= day_delta:
163                 break 164       else:
165                             end_ix += 1
166
167                              log.debug('Sufficient time from times[{0}..{1}] (day #{2} to #{3})'
168                              .format(window.start, end_ix, dates[window.start], dates[end_ix]))
169
170             return end_ix
171
172
173     def enough_samples(dates, meow_size):
174     """Change detection requires a minimum number of samples (as specified 175       by meow size).
176
177     This function improves readability of logic that performs this check.
178
179                         Args:
180                         dates: list of ordinal day numbers relative to some epoch,
181                         the particular epoch does not matter.
182                         meow_size: minimum expected observation window needed to
183                         produce a fit.
184
185                 Returns:
186                 bool: True if times contains enough samples
187                 False otherwise.
188                 """
189                 return len(dates) >= meow_size
190
191
192     def enough_time(dates, day_delta):
193     """Change detection requires a minimum amount of time (as specified by 194       day_delta).
195
196             This function, like `enough_samples` improves readability of logic
197             that performs this check.
198
199                         Args:
200                         dates: list of ordinal day numbers relative to some epoch,
201                         the particular epoch does not matter.
202                         day_delta: minimum difference between time at meow_ix and most
203                         recent observation.
204
```

```
205             Returns:
206             bool: True if the represented time span is greater than day_delta207       """
208                     return (dates[-1] - dates[0]) >= day_delta
209
210
211             def determine_num_coefs(dates, min_coef, mid_coef, max_coef, num_obs_factor):
212             """
213             Determine the number of coefficients to use for the main fit procedure214
215     This is based mostly on the amount of time (in ordinal days) that is 216       going to be covered by the model
217
218     This is referred to as df (degrees of freedom) in the model section 219
220                     Args:
221                     dates: 1-d array of representative ordinal dates
222                     min_coef: minimum number of coefficients223           mid_coef: mid number of coefficients
224                     max_coef: maximum number of coefficients
225                     num_obs_factor: used to scale the time span
226
227                     Returns:
228                     int: number of coefficients to use during the fitting process
229                     """
230                     span = dates.shape[0] / num_obs_factor
231
232       if span < mid_coef: 233     return
min_coef 234       elif span < max_coef:
235       return mid_coef 236         else:
237                         return max_coef
238
239
240             def update_processing_mask(mask, index, window=None):
241             """
242             Update the persistent processing mask.
243
244             Because processes apply the mask first, index values given are in relation
245             to that. So we must apply the mask to itself, then update the boolean
246             values.
247
248             The window slice object is to catch when it is in relation to some
249             window of the masked values. So, we must mask against itself, then look at
250             a subset of that result.
251
252     This method should create a new view object to avoid mutability issues.
253
254                             Args:
255                             mask: 1-d boolean ndarray, current mask being used
256                             index: int/list/tuple of index(es) to be excluded from processing,
257                             or boolean array
258                             window: slice object identifying a further subset of the mask259
260                     Returns:
261                     1-d boolean ndarray
262                     """
263                     new_mask = mask[:]
264                     sub_mask = new_mask[new_mask]
265
266             if window:
267             sub_mask[window][index] = False 268             else:
269                         sub_mask[index] = False
270
```

```
271                 new_mask[new_mask] = sub_mask
272
273             return new_mask
274
275
276     # def find_closest_doy(dates, date_idx, window, num):
277     #     """
278     #     Find the closest n dates based on day of year.
279
280     #     e.g. if the date you are looking for falls on July 1, then find
281     #     n number of dates that are closest to that same day of year.
282
283     #     Args:
284     #         dates: 1-d ndarray of ordinal day values
285     #         date_idx: index of date value
286     #         window: slice object identifying the subset of values used in the
287     #             current model
288     #         num: number of index values desired
289
290     #     Returns:
291     #         1-d ndarray of index values
292     #     """
293     #     # May be a better way of doing this
294     #     d_rt = dates[window] - dates[date_idx]
295     #     d_yr = np.abs(np.round(d_rt / 365.25) * 365.25 - d_rt)
296
297 #     return np.argsort(d_yr)[:num]
298
299             def find_closest_doy(dates, date_idx, window, num):
300             """
301             Find the closest n dates based on day of year.
302
303             e.g. if the date you are looking for falls on July 1, then find
304             n number of dates that are closest to that same day of year.
305
306                         Args:
307                         dates: 1-d ndarray of ordinal day values
308                         date_idx: index of date value
309                         window: slice object identifying the subset of values used in the
310                         current model
311                         num: number of index values desired
312
313                     Returns:
314                     1-d ndarray of index values
315                     """
316                     # Ensure date_idx is within bounds of the window
317                     date_idx = max(window.start, min(date_idx, window.stop - 1)) 318
319             # May be a better way of doing this
320             d_rt = dates[window] - dates[date_idx]
321             d_yr = np.abs(np.round(d_rt / 365.25) * 365.25 - d_rt) 322
323                     return np.argsort(d_yr)[:num]
324
325
326             def adjustpeek(dates, defpeek):
327             """
328             Adjust the number of observations looked at for the forward processing window
```

```
329             based on observation date characteristics
330
331                     Args:
332                     dates: 1-d ndarray of observation dates
333                     defpeek: default number of observations
334
335                     Returns:
336                     int number of observations to use
337                     """
338                     delta = np.median(np.diff(dates))
339                     adj_peek = int(np.round(defpeek * 16 / delta)) 340
341     return adj_peek if adj_peek > defpeek else defpeek 342
343
344             def adjustchgthresh(peek, defpeek, defthresh, chisquare_prob, deg_free):
345             """
346             Adjust the change threshold if the peek window size has changed347
348                     Args:
349                     peek: peek window size determined from adjustpeek
350                     defpeek: default window size
351                     defthresh: default change threshold
352                     chisquare_prob: default chi-square probability
353                     deg_free: degrees of freedom of the chi-square distribution354
355                     Returns:
356                     float change threshold to use
357                     """
358                     thresh = defthresh 359              if peek > defpeek:
360                     pt_cg = 1 - (1 - chisquare_prob) ** (defpeek / peek)
361                     thresh = chi2.ppf(pt_cg, deg_free)
362
363                 return thresh
364
365
366             def returnThresholdFromProb(chisquare_prob, deg_free):
367             """
368             Calculates the chi-square value based on the probability and degrees of freedom.
369
370                     Args:
371                     chisquare_prob: chi-square probability
372                     deg_free: degrees of freedom of the chi-square distribution373
373                     Returns:
374                     float change threshold to use
375                     """
376                     return chi2.ppf(chisquare_prob, deg_free)
377
```

2.3. __init__.py

```
1  ============ '__init__.py' ============
 2
 3  import time
 4  import logging
 5
 6      # from ccd.procedures import fit_procedure as __determine_fit_procedure
 7      from ccd.procedures import standard_procedure
 8  import numpy as np
 9  from ccd import app, math_utils, qa 10   import importlib
```

```
    from ccd.version import __version__
    from ccd.version import __algorithm__ as algorithm
from ccd.version import __name
log = logging.getLogger(__name)


        def attr_from_str(value):
        """Returns a reference to the full qualified function, attribute or class.

        Args:
        value = Fully qualified path (e.g. 'ccd.models.lasso.fitted_model')22
                Returns:
                A reference to the target attribute (e.g. fitted_model)
                """module, target = value.rsplit('.', 1) try:
                obj = importlib.import_module(module) return getattr(obj, target)
        except (ImportError, AttributeError) as e:
                log.debug(e) return None

        def __attach_metadata(procedure_results):
        """
        Attach some information on the algorithm version, what procedure was used,
        and which inputs were used

                Returns:
                A dict representing the change detection results

          {algorithm: 'pyccd:x.x.x',
          processing_mask: (bool, bool, ...),
          snow_prob: float,
          water_prob: float,
          cloud_prob: float,
          change_models: [
          {start_day: int,49          end_day: int,
                                    break_day: int,
                                    observation_count: int,
                                    change_probability: float,
                                    curve_qa: int,
                                    ndvi:     {magnitude: float,
                                    rmse: float,
                                    coefficients: (float, float, ...),
                                    intercept: float},
                                    green:    {magnitude: float,
                                    rmse: float,
                                    coefficients: (float, float, ...),
                                    intercept: float},
                                    red:     {magnitude: float,
                                    rmse: float,
                                    coefficients: (float, float, ...),
                                    intercept: float},
                                    nir:     {magnitude: float,
                                    rmse: float,
                                    coefficients: (float, float, ...),
                                    intercept: float},
                                    swir1:    {magnitude: float,
                                    rmse: float,
                                    coefficients: (float, float, ...),
```

```
                                    intercept: float},
                                    swir2:    {magnitude: float,
                                    rmse: float,
                                    coefficients: (float, float, ...),
                                    intercept: float},
                                    thermal:  {magnitude: float,
                                    rmse: float,
                                    coefficients: (float, float, ...),
                                    intercept: float}}
                                    ]
                                    }
                                    """
                                    change_models, processing_mask = procedure_results
                    return {'algorithm': algorithm,
                    'processing_mask': [int(_) for _ in processing_mask],
                    'change_models': change_models}
                    # 'cloud_prob': probs[0],
                    # 'snow_prob': probs[1],
                    # 'water_prob': probs[2]}


        def __split_dates_spectra(matrix):
        """ Slice the dates and spectra from the matrix and return """
        return matrix[0], matrix[1:6]


        def __sort_dates(dates):
        """ Sort the values chronologically """
        return np.argsort(dates)


        def __check_inputs(dates, spectra):
        """
        Make sure the inputs are of the correct relative size to each-other.

            Args:
            dates: 1-d ndarray
            quality: 1-d ndarray
            spectra: 2-d ndarray
            """
            # Make sure we only have one dimension
            assert dates.ndim == 1
        # Make sure quality is the same
        # assert dates.shape == quality.shape
        # Make sure there is spectral data for each date
        assert dates.shape[0] == spectra.shape[1]

        # def detect(dates, ndvis, greens, reds, nirs, swir1s, swir2s, params=None):
        def detect(dates, ndvis, greens, swir2s, params=None):
        """Entry point call to detect change

        No filtering up-front as different procedures may do things
        differently

            Args:
            dates:    1d-array or list of ordinal date values
```

```
130                 ndvis:    1d-array or list of ndvis  values
131                 greens:   1d-array or list of green band values
132                 reds:     1d-array or list of red band values
133                 nirs:     1d-array or list of nir band values
134                 swir1s:   1d-array or list of swir1 band values
135                 swir2s:   1d-array or list of swir2 band values
136
137                       params: python dictionary to change module wide processing
138                       parameters
139
140                 Returns:
141                 Tuple of ccd.detections namedtuples
142                 """
143                 t1 = time.time()
144
145                   proc_params = app.get_default_params()
146
147                 if params:
148                 proc_params.update(params)
149
150                  dates = np.asarray(dates)
151
152     # spectra = np.stack((ndvis, greens, reds,  nirs, swir1s, swir2s)) 153
154                       spectra = np.stack((ndvis, greens, swir2s))
155
156                   __check_inputs(dates, spectra)
157
158           indices = __sort_dates(dates)
159           dates = dates[indices]
160           spectra = spectra[:, indices]
161
162           # load the fitter_fn
163           fitter_fn = attr_from_str(proc_params.FITTER_FN)
164
165           results = standard_procedure(dates, spectra, fitter_fn, proc_params)
166           log.debug('Total time for algorithm: %s', time.time() - t1) 167
168           # call detect and return results as the detections namedtuple
169           return __attach_metadata(results)
```

2.4. math_utils.py

```
3     """
4     Contains commonly used math functions.
5
6     This file is meant to help code reuse, profiling, and look at speeding up
7     individual operations.
8
9  In the interest of avoiding circular imports, this should be kept to be fairly 10   stand-alone. I.e. it should not import
any other piece of the overall project.
11    """
12    from functools import wraps
13
```

```
14  import numpy as np 15   from scipy.stats
import mode
16
17      # TODO: Cache timings
18      # TODO: Numba timings
19
20
21              def adjusted_variogram(dates, observations):
22              """
23              Calculate a modified first order variogram/madogram.
24
25      This method differentiates from the standard calculate_variogram in that 26       it attempts to only use
observations that are greater than 30 days apart. 27
28      This attempts to combat commission error due to temporal autocorrelation.
29
30                              Args:
31                              dates: 1-d array of values representing ordinal day
32                              observations: 2-d array of spectral observations corresponding to the
33                              dates array
34
35                      Returns:
36                      1-d ndarray of floats
37                      """
38                      vario = calculate_variogram(observations)
39
40                      for idx in range(dates.shape[0]):
41                      var = dates[1 + idx:] - dates[:-idx - 1]
42
43                      majority = mode(var, axis=None, keepdims=True).mode[0] #explicitly called with keepdims=True to ensure
                        compatibility with newer scipy versions
44
45                              if majority > 30:
46                              diff = observations[:, 1 + idx:] - observations[:, :-idx - 1]
47                              ids = var > 30
48
49                              vario = np.median(np.abs(diff[:, ids]), axis=1)
50                              break
51
52                  return vario
53
54
55              def euclidean_norm(vector):
56              """
57              Calculate the euclidean norm across a vector58
59      This is the default norm method used by Matlab
60
61                      Args:
62                      vector: 1-d array of values
63
64                      Returns:
65                      float
66                      """
67                      return np.sum(vector ** 2) ** .5
68
69
70              def sum_of_squares(vector, axis=None):
```

```
    """
    Squares the values, then adds them up

        Args:
        vector: 1-d array of values, or n-d array with an axis set
        axis: numpy axis to operate on in cases of more than 1-d array77
        Returns:
        float
        """
        return np.sum(vector ** 2, axis=axis)


def calc_rmse(actual, predicted, num_pm=0):
    """
    Calculate the root mean square of error for the given inputs87
            Args:
            actual: 1-d array of values, observed
            predicted: 1-d array of values, predicted
            num_pm: number of parameters to use for the calculation if based on a
            smaller sample set

        Returns:
        float: root mean square value
        1-d ndarray: residuals
        """
        residuals = calc_residuals(actual, predicted) 99
            return ((np.sum(residuals ** 2) / (residuals.shape[0] - num_pm)) ** 0.5,
            residuals)


def calc_median(vector):
    """
    Calculate the median value of the given vector

        Args:
        vector: array of values

        Returns:
        float: median value
        """
        return np.median(vector)


def calc_residuals(actual, predicted):
    """
    Helper method to make other code portions clearer

        Args:
        actual: 1-d array of observed values
        predicted: 1-d array of predicted values

        Returns:
        ndarray: 1-d array of residual values
        """
        return actual - predicted
```

```
131     # def kelvin_to_celsius(thermals, scale=10):
132     #     """
133     #     Convert kelvin values to celsius
134     #
135     #     L2 processing for the thermal band (known as Brightness Temperature) is
136     #     initially done in kelvin and has been scaled by a factor of 10 already,
137     #     in the interest of keeping the values in integer space, a further factor 138   #     of 10 is calculated.
139     #
140     #     scaled C = K * 10 - 27315
141     #     unscaled C = K / 10 - 273.15
142     #
143     #     Args:
144     #         thermals: 1-d ndarray of scaled thermal values in kelvin
145     #         scale: int scale factor used for the thermal values
146     #
147     #     Returns:
148     #         1-d ndarray of thermal values in scaled degrees celsius
149     #     """
150     #     return thermals * scale - 27315
151
152
153             def calculate_variogram(observations):
154             """
155             Calculate the first order variogram/madogram across all bands
156
157     Helper method to make subsequent code clearer
158
159                     Args:
160                     observations: spectral band values
161
162             Returns:
163             1-d ndarray representing the variogram values164       """
165     return np.median(np.abs(np.diff(observations)), axis=1) 166
167
168             def mask_duplicate_values(vector):
169             """
170             Mask out duplicate values.
171
172             Mainly used for removing duplicate observation dates from the dataset.
173             Just because there are duplicate observation dates, doesn't mean that
174             both have valid data.
175
176     Generally this should be applied after other masks.
177
178                     Arg:
179                     vector: 1-d ndarray, ordinal date values
180
181                     Returns:
182                     1-d boolean ndarray
183                     """
184                     mask = np.zeros_like(vector, dtype=np.bool)
185                     mask[np.unique(vector, return_index=True)[1]] = 1
186
187                 return mask
188
189
```

```
190         def mask_value(vector, val):
191         """
192         Build a boolean mask around a certain value in the vector.
193
194                 Args:
195                 vector: 1-d ndarray of values
196                 val: values to mask on
197
198                 Returns:
199                 1-d boolean ndarray
200                 """
201                 return vector == val
202
203
204         def count_value(vector, val):
205         """
206         Count the number of occurrences of a value in the vector.
207
208                 Args:
209                 vector: 1-d ndarray of values
210                 val: value to count
211
212                 Returns:
213                 int
214                 """
215                 return np.sum(mask_value(vector, val))
```

2.5. parameters.py

```
1  ============ 'parameters.py' ============
2
3             ##############################
4             # Default Configuration Options
5             ##############################
6             defaults = {
7             'MEOW_SIZE': 12,
8             'PEEK_SIZE': 6,
9             'DAY_DELTA': 365,
10            'AVG_DAYS_YR': 365.2425,
11            'MIN_YEARS' : 1, #1.33
12
13          # 2 for tri-modal; 2 for bi-modal; 2 for seasonality; 2 for linear
14          'COEFFICIENT_MIN': 4,
15          'COEFFICIENT_MID': 6,
16          'COEFFICIENT_MAX': 8,
17
18          # Value used to determine the minimum number of observations required for a
19          # defined number of coefficients
20          # e.g. COEFFICIENT_MIN * NUM_OBS_FACTOR = 12
21          'NUM_OBS_FACTOR': 3,
22
23          ##############################
24          # Define spectral band indices on input observations array
25          ##############################
```

```
26
27              'BLUE_OR_NDVI_IDX': 0,
28              'GREEN_IDX': 1, 29            # 'RED_IDX': 2,
30              # 'NIR_IDX': 3,
31              # 'SWIR1_IDX': 4,
32              # 'SWIR2_IDX': 5,
33              'SWIR2_IDX': 2,
34
35              # Spectral bands that are utilized for detecting change
36              #'DETECTION_BANDS': [1, 2, 3, 4, 5], # Breakpointbands; tipicamente qt mais bandas, mais breaks estimados
37              # 'DETECTION_BANDS': [0, 1, 5],
38              'DETECTION_BANDS': [0, 1, 2], # Breakpointbands; tipicamente qt mais bandas, mais breaks estimados
39
40              # Spectral bands that are utilized for Tmask filtering
41              # 'TMASK_BANDS': [1, 5],
42              'TMASK_BANDS': [1, 2],
43
44              ##############################
45              # Representative values in the QA band
46              ##############################
47              # 'QA_BITPACKED': True,
48              # original CFMask values
49              #QA_FILL: 255
50              #QA_CLEAR: 0
51              #QA_WATER: 1
52              #QA_SHADOW: 2
53              #QA_SNOW: 3
54              #QA_CLOUD: 4
55              # ARD bitpacked offsets
56              # 'QA_FILL': 0,
57              # 'QA_CLEAR': 1,
58              # 'QA_WATER': 2,
59              # 'QA_SHADOW': 3,
60              # 'QA_SNOW': 4,
61              # 'QA_CLOUD': 5,
62              # 'QA_CIRRUS1': 8,
63              # 'QA_CIRRUS2': 9,
64              # 'QA_OCCLUSION': 10,
65
66              ##############################
67              # Representative values for the curve QA
68              ##############################
69              # 'CURVE_QA': {
70              #     'PERSIST_SNOW': 54,
71              #     'INSUF_CLEAR': 44,
72              #     'START': 14,
73              #     'END': 24},
74
75              ##############################
76              # Threshold values used
77              ##############################
78              'CLEAR_OBSERVATION_THRESHOLD': 3,
79              'CLEAR_PCT_THRESHOLD': 0.25,
80              'SNOW_PCT_THRESHOLD': 0.75,
81              'OUTLIER_THRESHOLD': None, #35.888186879610423, #Tmask (df = detection_bands, prob = 0.999999)
```

```
            'CHANGE_THRESHOLD': None, #15.086272469388987,
            'CHISQUAREPROB': 0.999, #0.99
            'T_CONST': 4.89,

            # Value added to the median green value for filtering purposes
            'MEDIAN_GREEN_FILTER': 400,

            ############################
            # Values related to model fitting
            ############################
            'FITTER_FN': 'ccd.models.lasso.fitted_model',
            'LASSO_MAX_ITER': 1000,
            'ALPHA': 20
            }
```

2.6. procedures.py

```
============ 'procedures.py' ============

    """Functions for providing the over-arching methodology. Tying together the
    individual components that make-up the change detection process. This module
    should really contain any method that could be considered procedural. Methods
    must accept the processing parameters, then use those values for the more
    functional methods that they call. The hope is that this will eventually get converted more and more away from procedural and move more towards the
functional paradigm.

    Any methods determined by the fit_procedure call must accept same 5 arguments,
    in the same order: dates, observations, fitter_fn, quality, proc_params.

The results of this process is a list-of-lists of change models that correspond to observation spectra. A processing mask is also returned, outlining which
observations were utilized and which were not.

    Pre-processing routines are essential to, but distinct from, the core change
    detection algorithm. See the `ccd.qa` for more details related to this
    step.

For more information please refer to the pyccd Algorithm Description Document.

    .. _Algorithm Description Document:
    https://drive.google.com/drive/folders/0BzELHvbrg1pDREJlTF8xOHBZbEU """
    import logging
    import numpy as np

from ccd import qa
            from ccd.change import enough_samples, enough_time,\
            update_processing_mask, stable, determine_num_coefs, calc_residuals, \
            find_closest_doy, change_magnitude, detect_change, detect_outlier, \
            adjustpeek, adjustchgthresh, returnThresholdFromProb
            from ccd.models import results_to_changemodel, tmask
            from ccd.math_utils import adjusted_variogram, euclidean_norm
log = logging.getLogger(__name__)


```

```
41          def standard_procedure(dates, observations, fitter_fn, proc_params):
42          """
43          Runs the core change detection algorithm.
44
45      Step 1: initialize -- Find an initial stable time-frame to build from.
46
47          Step 2: lookback -- The initlize step may have iterated the start of the
48          model past the previous break point. If so then we need too look back at49      previous values to see if they
          can be included within the new
50      initialized model.
51
52          Step 3: catch -- Fit a general model to values that may have been skipped
53          over by the previous steps.
54
55      Step 4: lookforward -- Expand the time-frame until a change is detected.
56
57      Step 5: Iterate.
58
59      Step 6: catch -- End of time series considerations.
60
61                          Args:
62                          dates: list of ordinal day numbers relative to some epoch,
63                          the particular epoch does not matter.
64                          observations: 2-d array of observed spectral values corresponding
65                          to each time.
66                          fitter_fn: a function used to fit observation values and
67                          acquisition dates for each spectra.
68
69        proc_params: dictionary of processing parameters
70
71                          Returns:
72                          list: Change models for each observation of each spectra.
73                          1-d ndarray: processing mask indicating which values were used
74                          for model fitting
75                          """
76                          # TODO do this better
77                          meow_size = proc_params.MEOW_SIZE
78                          defpeek = proc_params.PEEK_SIZE
79
80                              log.debug('Build change models - dates: %s, obs: %s, '
81                              'meow_size: %s, peek_size: %s',
82                              dates.shape[0], observations.shape, meow_size, defpeek)
83
84      processing_mask = np.ones(dates.shape[0], dtype=bool) 85
86                      obs_count = np.sum(processing_mask)
87
88      log.debug('Processing mask initial count: %s', obs_count) 89
90          # Accumulator for models. This is a list of ChangeModel named tuples
91          results = []
92
93                  if obs_count <= meow_size:
94                  return results, processing_mask
95
96                                                                  # TODO Temporary setup on this to just
                                                                    get it going
```

```
97                                                                  peek_size =
                                                                    adjustpeek(dates[processing_mask],
                                                                    defpeek)
98                                                                  proc_params.PEEK_SIZE = peek_size
99                                                                  proc_params.CHANGE_THRESHOLD =
                                                                    returnThresholdFromProb(proc_params.C
                                                                    HISQUAREPROB,
                                                                    len(proc_params.DETECTION_BANDS)
                                                                    )
100                                                                 proc_params.CHANGE_THRESHOLD =
                                                                    adjustchgthresh(peek_size, defpeek,
101                                                                 proc_params.CHANGE_THRESHOLD,
                                                                    proc_params.CHISQUAREPROB, len(
                                                                    proc_params.DETECTION_BANDS))
102
103         log.debug('Peek size: %s', proc_params.PEEK_SIZE)
104         log.debug('Chng thresh: %s', proc_params.CHANGE_THRESHOLD) 105
106         # Compute and store outlier threshold
107         proc_params.OUTLIER_THRESHOLD = returnThresholdFromProb(0.999999, len(proc_params.
            DETECTION_BANDS))
108
109         # Initialize the window which is used for building the models
110         model_window = slice(0, meow_size)
111         previous_end = 0
112
113 # Only capture general curve at the beginning, and not in the middle of 114      # two stable time segments
115             start = True
116
117                                                 # Calculate the variogram/madogram that will be used in
                                                    subsequent
118                                                 # processing steps. See algorithm documentation for further
                                                    information.
119                                                 variogram = adjusted_variogram(dates[processing_mask],
120                                                 observations[:, processing_mask])
121                                                 log.debug('Variogram values: %s', variogram)
122
123             # Only build models as long as sufficient data exists.
124             while model_window.stop <= dates[processing_mask].shape[0] - meow_size:
125             # Step 1: Initialize
126             log.debug('Initialize for change model #: %s', len(results) + 1) 127          if len(results) > 0:
128                 start = False
129
130                                                 # Make things a little more readable by breaking this apart
131                                                 # catch return -> break apart into components
132                                                 initialized = initialize(dates, observations, fitter_fn, model_window,
133                                                 processing_mask, variogram, proc_params)
134
135 model_window, init_models, processing_mask = initialized 136
137 # print('After Initialization - Processing Mask:', processing_mask) 138
139 # Catch for failure 140     if
init_models is None: 141
log.debug('Model initialization failed')
142                     break
143
144                                         # Step 2: Lookback
145                                         if model_window.start > previous_end:
```

```
                                    lb = lookback(dates, observations, model_window, init_models,
                                    previous_end, processing_mask, variogram, proc_params)

                        model_window, processing_mask = lb

    # print('After Lookback - Processing Mask:', processing_mask)
            # Step 3: catch
            # If we have moved > peek_size from the previous break point
            # then we fit a generalized model to those points.
                                    if model_window.start - previous_end > peek_size and start is True:
                                    results.append(catch(dates,
                                    observations,
                                    fitter_fn,
                                    processing_mask,
                                    slice(previous_end, model_window.start),
                                    proc_params))
                                    start = False

    # Handle specific case where if we are at the end of a time series and
    # the peek size is greater than what remains of the data.
                            if model_window.stop + peek_size > dates[processing_mask].shape[0]:
                        break

                                    # Step 4: lookforward
                                    log.debug('Extend change model')
                                    lf = lookforward(dates, observations, model_window, fitter_fn,
                                    processing_mask, variogram, proc_params)

            result, processing_mask, model_window = lf
            results.append(result)

    # print('After Lookforward - Processing Mask:', processing_mask)
    log.debug('Accumulate results, {} so far'.format(len(results)))
            # Step 5: Iterate
            previous_end = model_window.stop
            model_window = slice(model_window.stop, model_window.stop + meow_size)
        # Step 6: Catch
        # We can use previous start here as that value should be equal to
        # model_window.stop due to the constraints on the the previous while
        # loop.
            if previous_end + peek_size < dates[processing_mask].shape[0]:
            model_window = slice(previous_end, dates[processing_mask].shape[0])
            results.append(catch(dates, observations, fitter_fn,
            processing_mask, model_window,
            proc_params=proc_params))

    # print('After Final Catch - Processing Mask:', processing_mask)
             log.debug("change detection complete")

            return results, processing_mask


                    def initialize(dates, observations, fitter_fn, model_window, processing_mask,
                    variogram, proc_params):
                    """
```

```
206                         Determine a good starting point at which to build off of for the207      subsequent process of
                            change detection, both forward and backward.
208
209                     Args:
210                     dates: 1-d ndarray of ordinal day values
211                     observations: 2-d ndarray representing the spectral values
212                     fitter_fn: function used for the regression portion of the algorithm
213                     model_window: start index of time/observation window
214                     processing_mask: 1-d boolean array identifying which values to
215                     consider for processing
216                     variogram: 1-d array of variogram values to compare against for the
217                     normalization factor
218                     proc_params: dictionary of processing parameters
219
220                 Returns:
221                 slice: model window that was deemed to be a stable start
222                 namedtuple: fitted regression models
223                 """
224                 # TODO do this better
225                 meow_size = proc_params.MEOW_SIZE
226                 day_delta = proc_params.DAY_DELTA
227                 detection_bands = proc_params.DETECTION_BANDS
228                 tmask_bands = proc_params.TMASK_BANDS
229                 change_thresh = proc_params.CHANGE_THRESHOLD
230                 tmask_scale = proc_params.T_CONST
231                 avg_days_yr = proc_params.AVG_DAYS_YR
232                 fit_max_iter = proc_params.LASSO_MAX_ITER
233                 alpha = proc_params.ALPHA
234
235         period = dates[processing_mask]
236         spectral_obs = observations[:, processing_mask]
237
238         log.debug('Initial %s', model_window)
239         models = None
240         while model_window.stop + meow_size < period.shape[0]: 241          # Finding a sufficient window of time
            needs to run
242             # each iteration because the starting point
243             # will increment if the model isn't stable, incrementing only
244             # the window stop in lock-step does not guarantee a 1-year+ 245  # time-range.
246                 if not enough_time(period[model_window], day_delta):
247                 model_window = slice(model_window.start, model_window.stop + 1)
248                   continue
249             # stop = find_time_index(dates, model_window, meow_size, day_delta)
250             # model_window = slice(model_window.start, stop)
251             log.debug('Checking window: %s', model_window)
252
253     # Count outliers in the window, if there are too many outliers then 254          # try again.
255                                                 tmask_outliers = tmask.tmask(period[model_window],
256                                                 spectral_obs[:, model_window],
257                                                 variogram, tmask_bands, tmask_scale,
258                                                 avg_days_yr)
259
260                     tmask_count = np.sum(tmask_outliers)
261
262     log.debug('Number of Tmask outliers found: %s', tmask_count) 263
```

```
264     # Subset the data to the observations that currently under scrutiny 265             # and remove the outliers
identified by the tmask.
266     tmask_period = period[model_window][~tmask_outliers] 267
268     # TODO should probably look at a different fit procedure to handle 269             # the following case.
270                 if tmask_count == model_window.stop - model_window.start:
271                 log.debug('Tmask identified all values as outliers') 272
273                                     model_window = slice(model_window.start, model_window.stop + 1)
274                         continue
275
276     # Make sure we still have enough observations and enough time after 277         # the tmask removal.
278     if not enough_time(tmask_period, day_delta) or \ 279             not
enough_samples(tmask_period, meow_size):
280
281                                     log.debug('Insufficient time or observations after Tmask, '
282                                     'extending model window')
283
284     model_window = slice(model_window.start, model_window.stop + 1) 285         continue
286
287     # Update the persistent mask with the values identified by the Tmask 288         if any(tmask_outliers):
289                                                                                     processing_mask =
                                                                                    update_processing_mask(processing_mas
                                                                                    k,
290                                                                                 tmask_outliers,
291                                                                                 model_window)
292
293                                                 # The model window now actually refers to a smaller slice
294                                                 model_window = slice(model_window.start,
295                                                 model_window.stop - tmask_count)
296                                                 # Update the subset
297                                                 period = dates[processing_mask]
298                                                 spectral_obs = observations[:, processing_mask]
299
300                 log.debug('Generating models to check for stability')
301                 models = [fitter_fn(period[model_window], spectrum,
302                 fit_max_iter, avg_days_yr, 4, alpha) 303             for spectrum in spectral_obs[:, model_window]]
304
305     # If a model is not stable, then it is possible that a disturbance 306       # exists somewhere in the
observation window. The window shifts 307             # forward in time, and begins initialization again. 308
if not stable(models, period[model_window], variogram,
309                                             change_thresh, detection_bands):
310
311                                     model_window = slice(model_window.start + 1, model_window.stop + 1)
312
313                         # remove a sobreposição de curvas mas estraga os outros resultados:
314                         # model_window = slice(model_window.stop, model_window.stop + meow_size) 315
316                         log.debug('Unstable model, shift window to: %s', model_window)
317                         models = None
318                           continue
319
320                         else:
321                         log.debug('Stable start found: %s', model_window)
322                          break
323
324                     return model_window, models, processing_mask
325
```

```
326
327                             def lookforward(dates, observations, model_window, fitter_fn, processing_mask,
328                             variogram, proc_params):
329                             """Increase observation window until change is detected or 330       we are out of observations.
331
332                         Args:
333                         dates: list of ordinal day numbers relative to some epoch,
334                         the particular epoch does not matter.
335                         observations: spectral values, list of spectra -> values
336                         model_window: span of indices that is represented in the current
337                         process
338                         fitter_fn: function used to model observations
339                         processing_mask: 1-d boolean array identifying which values to
340                         consider for processing
341                         variogram: 1-d array of variogram values to compare against for the
342                         normalization factor
343                         proc_params: dictionary of processing parameters
344
345                     Returns:
346                     namedtuple: representation of the time segment
347                     1-d bool ndarray: processing mask that may have been modified
348                     slice: model window
349                     """
350                     # TODO do this better
351                     peek_size = proc_params.PEEK_SIZE
352                     coef_min = proc_params.COEFFICIENT_MIN
353                     coef_mid = proc_params.COEFFICIENT_MID
354                     coef_max = proc_params.COEFFICIENT_MAX
355                     num_obs_fact = proc_params.NUM_OBS_FACTOR
356                     detection_bands = proc_params.DETECTION_BANDS
357                     change_thresh = proc_params.CHANGE_THRESHOLD
358                     outlier_thresh = proc_params.OUTLIER_THRESHOLD
359                     avg_days_yr = proc_params.AVG_DAYS_YR
360                     fit_max_iter = proc_params.LASSO_MAX_ITER
361                     alpha=proc_params.ALPHA
362                     min_years=proc_params.MIN_YEARS
363
364             # Step 4: lookforward.
365             # The second step is to update a model until observations that do not 366             # fit the model are found.
367         log.debug('lookforward initial model window: %s', model_window) 368
369         # The fit_window pertains to which locations are used in the model 370          # regression, while the
model_window identifies the locations in which 371             # fitted models apply to. They are not always the
same.
372                 fit_window = model_window
373
374             # Initialized for a check at the first iteration.
375             models = None
376
377         # Simple value to determine if change has occured or not. Change may not 378             # have occurred if we
reach the end of the time series.
379                 change = 0
380
381             # Initial subset of the data
382             period = dates[processing_mask]
383             spectral_obs = observations[:, processing_mask] 384
385             # Used for comparison purposes
```

```
386             fit_span = period[model_window.stop - 1] - period[model_window.start] 387
388             #print('antes do while model_window.stop:',model_window.stop)
389             #print('antes do while peek size:', peek_size)
390             #print('antes do while period shape:', period.shape[0]) 391
392                     # stop is always exclusive
393
394                                                     # while model_window.stop + peek_size <= period.shape[0]:
395                                                     while model_window.stop < period.shape[0]:
396                                                     #print('model_window.stop:',model_window.stop)
397                                                     #print('peek size:', peek_size)
398                                                     #print('period shape:', period.shape[0])
399                                                     num_coefs = determine_num_coefs(period[model_window],
                                                        coef_min,
400                                                     coef_mid, coef_max, num_obs_fact)
401
402     peek_window = slice(model_window.stop, model_window.stop + peek_size) 403
404                     # Used for comparison against fit_span
405                     model_span = period[model_window.stop - 1] - period[model_window.start] 406
407     log.debug('Detecting change for %s', peek_window) 408
409                     # If we have less than 24 observations covered by the model_window
410                     # or it the first iteration, then we always fit a new window 411        # If the number of
                        observations that the current fitted models 412     # expand past a threshold, then we need to fit
                        new ones.
413                             if not models or model_window.stop - model_window.start < 24 or model_span >= min_years *
                                fit_span:
414                             fit_span = period[model_window.stop - 1] - period[
415                             model_window.start]
416
417                                                     fit_window = model_window
418                                                     log.debug('Retrain models')
419                                                     models = [fitter_fn(period[fit_window], spectrum,
420                                                     fit_max_iter, avg_days_yr, num_coefs, alpha)
421                                                     for spectrum in spectral_obs[:, fit_window]]
422
423                                                             residuals =
                                                                np.array([calc_residuals(period[peek_window],
424                                                             spectral_obs[idx, peek_window],
425                                                             models[idx], avg_days_yr)
426                                                             for idx in range(observations.shape[0])])
427
428                         if model_window.stop - model_window.start <= 24:
429                         comp_rmse = [models[idx].rmse for idx in detection_bands]
430
431                     # More than 24 points
432                     else:
433                     # We want to use the closest residual values to the peek_window 434          # values based on
                        seasonality.
435
436     # closest_indexes = find_closest_doy(period, peek_window.stop - 1, 437          #
fit_window, 24) 438
439                                                     closest_indexes = find_closest_doy(period,
                                                        min(peek_window.stop - 1, len( dates) - 1),
440                                                     fit_window, 24)
441
442     # Calculate an RMSE for the seasonal residual values, using 8 443      # as the degrees of freedom.
```

```
                                    comp_rmse = [euclidean_norm(models[idx].residual[closest_indexes]) / 4
                                    for idx in detection_bands]

    # Calculate the change magnitude values for each observation in the # peek_window.
                                            magnitude = change_magnitude(residuals[detection_bands, :],
                                            variogram[detection_bands],
                                            comp_rmse)

            if detect_change(magnitude, change_thresh):
            log.debug('Change detected at: %s', peek_window.start)
                # Change was detected, return to parent method
                change = 1
                 break
            elif detect_outlier(magnitude[0], outlier_thresh):
            log.debug('Outlier detected at: %s', peek_window.start)
                                                        # Keep track of any outliers so they will
                                                        be excluded from future
                                                        # processing steps
                                                        processing_mask =
                                                        update_processing_mask(processing_mas
                                                        k,
                                                        peek_window.start)

                # Because only one value was excluded, we shouldn't need to adjust
                # the model_window.  The location hasn't been used in
                # processing yet. So, the next iteration can use the same windows
                # without issue.
                period = dates[processing_mask]
                spectral_obs = observations[:, processing_mask]
                  continue

    # Check before incrementing the model window, otherwise the reporting    # can get a little messy.

            # if model_window.stop + peek_size > period.shape[0]:
            #     print('Condição de parada ativada: model_window.stop + peek_size > period.shape[0]')
            #     break

    model_window = slice(model_window.start, model_window.stop + 1)
                                            result = results_to_changemodel(fitted_models=models,
                                            start_day=period[model_window.start],
                                            end_day=period[model_window.stop - 1],
                                            break_day=period[peek_window.start],
                                            magnitudes=np.median(residuals, axis=1),
                                            observation_count=(
                                            model_window.stop - model_window.start),
                                            change_probability=change)

                return result, processing_mask, model_window


def lookback(dates, observations, model_window, models, previous_break,     processing_mask,
variogram, proc_params):
        """
        Special case when there is a gap between the start of a time series model      and the previous model break
        point, this can include values that were
```

```
    excluded during the initialization step.

                    Args:
                    dates: list of ordinal days
                    observations: spectral values across bands
                    model_window: current window of values that is being considered
                    models: currently fitted models for the model_window
                    previous_break: index value of the previous break point, or the start
                    of the time series if there wasn't one
                    processing_mask: index values that are currently being masked out from processing
                    variogram: 1-d array of variogram values to compare against for the
                    normalization factor
                    proc_params: dictionary of processing parameters

                Returns:
                slice: window of indices to be used
                array: indices of data that have been flagged as outliers
                """
                # TODO do this better
                peek_size = proc_params.PEEK_SIZE
                detection_bands = proc_params.DETECTION_BANDS
                change_thresh = proc_params.CHANGE_THRESHOLD
                outlier_thresh = proc_params.OUTLIER_THRESHOLD
                avg_days_yr = proc_params.AVG_DAYS_YR
                #alpha = proc_params.ALPHA

            log.debug('Previous break: %s model window: %s', previous_break, model_window)
            period = dates[processing_mask]
            spectral_obs = observations[:, processing_mask]
                while model_window.start > previous_break:
                # Three conditions to see how far we want to look back each iteration.
                # 1. If we have more than 6 previous observations
                # 2. Catch to make sure we don't go past the start of observations
                # 3. Less than 6 observations to look at

    # Important note about python slice objects, start is inclusive and # stop is exclusive,
regardless of direction/step if model_window.start - previous_break > peek_size:
                    peek_window = slice(model_window.start - 1, model_window.start - peek_size , -1)
                    elif model_window.start - peek_size <= 0:
                    peek_window = slice(model_window.start - 1, None, -1) else:
                                peek_window = slice(model_window.start - 1, previous_break - 1, -1)

                        log.debug('Considering index: %s using peek window: %s',
                        peek_window.start, peek_window)

                                                    residuals =
                                                    np.array([calc_residuals(period[peek_window],
                                                    spectral_obs[idx, peek_window],
                                                    models[idx], avg_days_yr)
                                                    for idx in range(observations.shape[0])])

    # log.debug('Residuals for peek window: %s', residuals)
                            comp_rmse = [models[idx].rmse for idx in detection_bands]

    log.debug('RMSE values for comparison: %s', comp_rmse)
                                                magnitude = change_magnitude(residuals[detection_bands, :],
```

```
562                                                 variogram[detection_bands],
563                                                 comp_rmse)
564
565                         if detect_change(magnitude, change_thresh):
566                         log.debug('Change detected for index: %s', peek_window.start)
567                         # change was detected, return to parent method
568                          break
569                                                                 elif detect_outlier(magnitude[0],
                                                                    outlier_thresh):
570                                                                 log.debug('Outlier detected for index:
                                                                    %s', peek_window.start)
571                                                                 processing_mask =
                                                                    update_processing_mask(processing_mas
                                                                    k,
572                                                                 peek_window.start)
573
574                         period = dates[processing_mask]
575                         spectral_obs = observations[:, processing_mask]
576
577     # Because this location was used in determining the model_window 578          # passed in, we must now
account for removing it.
579                                     model_window = slice(model_window.start - 1, model_window.stop - 1)
580                          continue
581
582                 log.debug('Including index: %s', peek_window.start)
583                 model_window = slice(peek_window.start, model_window.stop) 584
585                  return model_window, processing_mask
586
587
588         def catch(dates, observations, fitter_fn, processing_mask, model_window, proc_params):
589         """
590         Handle special cases where general models just need to be fitted and return
591         their results.
592
593                         Args:
594                         dates: list of ordinal day numbers relative to some epoch,
595                         the particular epoch does not matter.
596                         observations: spectral values, list of spectra -> values
597                         model_window: span of indices that is represented in the current
598                         process
599                         fitter_fn: function used to model observations
600                         processing_mask: 1-d boolean array identifying which values to
601                         consider for processing
602
603                 Returns:
604                 namedtuple representing the time segment
605
606         """
607         log.debug('Fitting catch model')
608
609         # if you want to change the `catch` implementation, modify it here
610         period = dates[processing_mask]
611         spectral_obs = observations[:, processing_mask]
612         model_period = period[model_window]
613         model_spectral = spectral_obs[:, model_window]
614
```

```
615             # Find indices where observations are equal to 65535
616             # invalid_indices = np.where(observations == 65535) 617
618             # # Ensure that invalid indices are within the valid range of dates
619             # invalid_indices = invalid_indices[0][(invalid_indices[0] >= 0) &
                (invalid_indices[0] < len(dates))]
620
621             # Create a boolean mask for invalid indices
622             # invalid_mask = np.zeros_like(processing_mask, dtype=bool) 623         #
                invalid_mask[invalid_indices] = True
624
625             # Update processing_mask based on invalid_mask
626             # processing_mask &= ~invalid_mask
627
628                     if np.all(processing_mask == False):
629                     return None
630
631             # TODO do this better
632             avg_days_yr = proc_params.AVG_DAYS_YR
633             fit_max_iter = proc_params.LASSO_MAX_ITER
634             num_coef = proc_params.COEFFICIENT_MIN
635             alpha = proc_params.ALPHA
636
637
638             log.debug('Catching observations: %s', model_window)
639             period = dates[processing_mask]
640             spectral_obs = observations[:,processing_mask] 641
642             # Subset the data based on the model window
643             model_period = period[model_window]
644             model_spectral = spectral_obs[:, model_window]
645
646                 #print(alpha,'alpha')
647
648                         models = [fitter_fn(model_period, spectrum, fit_max_iter, avg_days_yr,num_coef, alpha)
649                         for spectrum in model_spectral]
650
651     if model_window.stop >= period.shape[0]: 652
break_day = period[-1] 653          else:
654                         break_day = period[model_window.stop]
655
656             #print("Model Window Start:", model_window.start)
657             #print("Model Window Stop:", model_window.stop)
658             #print("Period Shape:", period.shape[0])
659
660
661             result = results_to_changemodel(
662             fitted_models=models,
663             start_day=period[model_window.start],
664             end_day=period[model_window.stop-1],
665             break_day=break_day,
666             magnitudes=np.zeros(shape=(6,)),
667             observation_count=(model_window.stop - model_window.start),
668             change_probability=0
669             )
670
                670         return result
```

2.7. version.py

```
========== 'version.py' ==========

    """ Module specifically to hold algorithm version information.  The reason this
    exists is the version information is needed in both setup.py for install and
    also in ccd/__init__.py when generating results.  If these values were
    defined in ccd/__init__.py then install would fail because there are other
    dependencies imported in ccd/__init__.py that are not present until after
    install. Do not import anything into this module."""
    __name = 'lcmap-pyccd'

    # While we sometimes may need to change the code, this may not actually change
    # the core algorithm. So, the core algorithm needs it's own version
    # that actually gets reported with results, and a release version for pypi 14   # and system integration purposes.
    __algorithm_version__ = '2018.10.17'
    __local_version__ = ''

    # __algorithm__ = ':'.join([__name, __algorithm_version__, __local_version__])
    __algorithm__ = ':'.join([__name, __algorithm_version__])
    __version__ = __algorithm_version__
    # __version__ = '.'.join([__algorithm_version__, __local_version__])
```

3. Models

3.1. __init__.py

```
========== '__init__.py' ==========

 from collections import namedtuple
 # TODO: establish standardize object for handling models used for general
 # regression purposes. This will truly make the code much more modular.

 # Because scipy models don't hold information on residuals or rmse, we should 8   # carry them forward with the
models themselves, so we don't have to 9   # recalculate them all the time 10   # TODO: give better names to avoid
model.model.predict nonsense
 FittedModel = namedtuple('FittedModel', ['fitted_model', 'residual', 'rmse']) 12
    def results_to_changemodel(fitted_models, start_day, end_day, break_day,
    magnitudes, observation_count, change_probability): 15        """
            Helper method to consolidate results into a concise, self-documenting data
            structure.

            This also converts any specific package types used during processing to
            standard Python types to help with downstream processing.

    {start_day: int, 23
end_day: int,
                        break_day: int,
                        observation_count: int,
                        change_probability: float,
                        ndvi:  {magnitude: float,
```

```
                    rmse: float,
                    coefficients: (float, float, ...),30            intercept: float},
    etc...

            Returns:
            dict

                        """
                        spectral_models = []
                        for ix, model in enumerate(fitted_models):
                        spectral = {'rmse': float(model.rmse),
                        'coefficients': tuple(float(c) for c in
    model.fitted_model.coef_), 42       'intercept': float(model.fitted_model.intercept_),
                        'magnitude': float(magnitudes[ix])}
                        spectral_models.append(spectral) 45
                    return {'start_day': int(start_day),
                    'end_day': int(end_day),
                    'break_day': int(break_day),
                    'observation_count': int(observation_count),
                    'change_probability': float(change_probability),
                    'ndvi': spectral_models[0],
                    'green': spectral_models[1],
53  # 'red': spectral_models[2], 54    # 'nir': spectral_models[3], 55    # 'swir1':
spectral_models[4], 56  # 'swir2': spectral_models[5]}
            58                      'swir2': spectral_models[2]}
```

3.2. Lasso.py

```
1  ============ 'lasso.py' ============
2
3      from sklearn import linear_model
4      import numpy as np
5
6      from ccd.models import FittedModel
7      from ccd.math_utils import calc_rmse
8      import logging
9      log = logging.getLogger(__name__)
10
11         def __coefficient_cache_key(observation_dates):
12         return tuple(observation_dates)
13
14         def coefficient_matrix(dates, avg_days_yr, num_coefficients):
15         """
16         Fourier transform function to be used for the matrix of inputs for
17         model fitting
18
19             Args:
20             dates: list of ordinal dates
21             num_coefficients: how many coefficients to use to build the matrix
22
23             Returns:
24             Populated numpy array with coefficient values
25             """
26             w = 2 * np.pi / avg_days_yr
27
28     matrix = np.zeros(shape=(len(dates), 7), order='F') 29
```

```
30  # lookup optimizations 31          # Before optimization - 12.53% of total runtime 32
    # After optimization  - 10.57% of total runtime
33          cos = np.cos
34          sin = np.sin
35
36          w12 = w * dates
37          matrix[:, 0] = dates
38          matrix[:, 1] = cos(w12)
39          matrix[:, 2] = sin(w12)
40
41                  if num_coefficients >= 6:
42                  w34 = 2 * w12
43                  matrix[:, 3] = cos(w34)
44                  matrix[:, 4] = sin(w34)
45
46                  if num_coefficients >= 8:
47                  w56 = 3 * w12
48                  matrix[:, 5] = cos(w56)
49                  matrix[:, 6] = sin(w56)
50
51              return matrix
52
53          def fitted_model(dates, spectra_obs, max_iter, avg_days_yr, num_coefficients, alpha):
54          """Create a fully fitted lasso model.
55
56                          Args:
57                          dates: list or ordinal observation dates
58                          spectra_obs: list of values corresponding to the observation dates for
59                          a single spectral band
60                          num_coefficients: how many coefficients to use for the fit
61                          max_iter: maximum number of iterations that the coefficients
62                          undergo to find the convergence point.
63
64                  Returns:
65                  sklearn.linear_model.Lasso().fit(observation_dates, observations)
66
67          Example:
68          fitted_model(dates, obs).predict(...)69       """
70                              coef_matrix = coefficient_matrix(dates, avg_days_yr, num_coefficients)
71
72
73          model = linear_model.Lasso(alpha, max_iter=max_iter) if alpha != 0 else linear_model.LinearRegression()
74          model.fit(coef_matrix, spectra_obs)
75
76              # model = lasso.fit(coef_matrix, spectra_obs)
77          predictions = model.predict(coef_matrix)
78          rmse, residuals = calc_rmse(spectra_obs, predictions, num_pm=num_coefficients) 79
80      return FittedModel(fitted_model=model, rmse=rmse, residual=residuals) 81
82      def predict(model, dates, avg_days_yr):
83      coef_matrix = coefficient_matrix(dates, avg_days_yr, 8) 84
        84          return model.fitted_model.predict(coef_matrix) 86
```

3.3. robust_fit.py

```
============ 'robust_fit.py' ============

    """
    Perform an iteratively re-weighted least squares 'robust regression'. Basically
    a clone of `statsmodels.robust.robust_linear_model.RLM` without all the lovely,
    but costly, creature comforts.

            Reference:
            http://statsmodels.sourceforge.net/stable/rlm.html
            http://cran.r-project.org/web/packages/robustreg/index.html
            http://cran.r-project.org/doc/contrib/Fox-Companion/appendix-robust-regression.pdf 12
    Run this file to test performance gains. Implementation is ~3x faster than
    statsmodels and can reach ~4x faster if Numba is available to accelerate.

"""
# Don't alias to ``np`` until fix is implemented
    # https://github.com/numba/numba/issues/1559
    import numpy
    import sklearn
    import scipy

# from yatsm.accel import try_jit

EPS = numpy.finfo('float').eps


# Weight scaling methods
# @try_jit(nopython=True) 30   def
bisquare(resid, c=4.685):
            """
            Returns weighting for each residual using bisquare weight function

                    Args:
                    resid (np.ndarray): residuals to be weighted
                    c (float): tuning constant for Tukey's Biweight (default: 4.685)

                    Returns:
                    weight (ndarray): weights for residuals

            Reference:
            http://statsmodels.sourceforge.net/stable/generated/statsmodels.robust.norms.T ukeyBiweight.html
            """
                # Weight where abs(resid) < c; otherwise 0
                        return (numpy.abs(resid) < c) * (1 - (resid / c) ** 2) ** 2


# @try_jit(nopython=True) 49   def
mad(x, c=0.6745):
            """
            Returns Median-Absolute-Deviation (MAD) of some data

                    Args:
                    resid (np.ndarray): Observations (e.g., residuals)
                    c (float): scale factor to get to ~standard normal (default: 0.6745)
                    (i.e. 1 / 0.75iCDF ~= 1.4826 = 1 / 0.6745)57
                    Returns:
```

```
59                  float: MAD 'robust' standard deivation  estimate
60
61                  Reference:
62                  http://en.wikipedia.org/wiki/Median_absolute_deviation
63                  """
64              # Return median absolute deviation adjusted sigma
65          rs = numpy.sort(numpy.abs(x))
66          return numpy.median(rs[4:]) / c
67
68  #   return numpy.median(numpy.fabs(x)) / c
69
70
71  # UTILITY FUNCTIONS 72  # @try_jit(nopython=True)
73          def _check_converge(x0, x, tol=1e-8):
74          return not numpy.any(numpy.fabs(x0 - x > tol))
75
76
77  # Broadcast on sw prevents nopython 78  # TODO: check implementation
https://github.com/numba/numba/pull/1542 79  # @try_jit()
80          def _weight_fit(X, y, w):
81          """
82          Apply a weighted OLS fit to data
83
84                  Args:
85                  X (ndarray): independent variables
86                  y (ndarray): dependent variable
87                  w (ndarray): observation weights
88
89                  Returns:
90                  tuple: coefficients and residual vector
91
92          """
93          sw = numpy.sqrt(w)
94
95          Xw = X * sw[:, None]
96          yw = y * sw
97
98      beta, _, _, _ = numpy.linalg.lstsq(Xw, yw, rcond=None) 99
100                     resid = y - numpy.dot(X, beta)
101
102                 return beta, resid
103
104
105 # Robust regression
106     class RLM(sklearn.base.BaseEstimator):
107     """ Robust Linear Model using Iterative Reweighted Least Squares (RIRLS) 108
109         Perform robust fitting regression via iteratively reweighted least squares
110         according to weight function and tuning parameter.
111
112         Basically a clone from `statsmodels` that should be much faster and follows
113         the scikit-learn __init__/fit/predict paradigm.
114
115                 Args:
116                 scale_est (callable): function for scaling residuals
117                 tune (float): tuning constant for scale estimate
```

```
118                 maxiter (int, optional): maximum number of iterations (default: 50)
119                 tol (float, optional): convergence tolerance of estimate120             (default: 1e-8)
121                     scale_est (callable): estimate used to scale the weights
122                     (default: `mad` for median absolute deviation)
123                     scale_constant (float): normalization constant (default: 0.6745)
124                     update_scale (bool, optional): update scale estimate for weights
125                     across iterations (default: True)
126                     M (callable): function for scaling residuals
127                     tune (float): tuning constant for scale estimate
128
129                     Attributes:
130                     coef_ (np.ndarray): 1D array of model coefficients
131                     intercept_ (float): intercept
132                     weights (np.ndarray): 1D array of weights for each observation from a
133                     robust iteratively reweighted least squares
134
135     """
136
137     def __init__(self, M=bisquare, tune=4.685, 138         scale_est=mad,
scale_constant=0.6745, 139          update_scale=True, maxiter=50, tol=1e-8):
140                 self.M = M
141                 self.tune = tune
142                 self.scale_est = scale_est
143                 self.scale_constant = scale_constant
144                 self.update_scale = update_scale
145                 self.maxiter = maxiter
146                 self.tol = tol
147
148     self.coef_ = None 149       self.intercept_ = 0.0
150
151                 def fit(self, X, y):
152                 """ Fit a model predicting y from X design matrix
153
154                     Args:
155                     X (np.ndarray): 2D (n_obs x n_features) design matrix
156                     y (np.ndarray): 1D independent variable
157
158                         Returns:
159                         object: return `self` with model results stored for method
160                         chaining
161
162                 """
163                 self.coef_, resid = _weight_fit(X, y, numpy.ones_like(y)) 164         self.scale =
                    self.scale_est(resid, c=self.scale_constant)
165
166
167                     Q, R = scipy.linalg.qr(X)
168                         E = X.dot(numpy.linalg.inv(R[0:X.shape[1],0:X.shape[1]]))
169     const_h= numpy.ones(X.shape[0])*0.9999 170
171                 h = numpy.minimum(const_h,numpy.sum(E*E,axis=1))
172                 adjfactor = numpy.divide(1,numpy.sqrt(1-h))
173     # self.coef_ = numpy.linalg.lstsq(R,(Q.T.dot(y)))[0] 174          # self.coef_, resid =
_weight_fit(X, y, numpy.ones_like(y)) 175          # U,s,v = numpy.linalg.svd(X) 176          #
print(self.coef_)
177
178                     if self.scale < EPS:
```

```
179                     return self
180
181                 iteration = 1
182                 converged = 0
183                 while not converged and iteration < self.maxiter:
184                 _coef = self.coef_.copy() 185     resid = y-X.dot(_coef)
186                         resid = resid * adjfactor
187                     # print resid
188
189                         if self.update_scale:
190                         self.scale = max(EPS*numpy.std(y),
191   self.scale_est(resid, c=self.scale_constant)) 192        # print self.scale
193 # print iteration,numpy.sort(numpy.abs(resid)/self.scale_constant) 194
195                     self.weights = self.M(resid / self.scale, c=self.tune)
196                     self.coef_, resid = _weight_fit(X, y, self.weights)
197                         # print 'w: ', self.weights
198
199                     iteration += 1
200                     converged = _check_converge(self.coef_, _coef, tol=self.tol)
201               # print resid
202                  return self
203
204             def predict(self, X):
205             """ Predict yhat using model
206
207             Args:
208             X (np.ndarray): 2D (n_obs x n_features) design matrix209
210                     Returns:
211                     np.ndarray: 1D yhat prediction
212
213             """
214             return numpy.dot(X[:,1:], self.coef_[1:]) + X[:,0]*self.coef_[0]
215                     # return numpy.dot(X, self.coef_) + self.intercept_
216
217                         def __str__(self):
218                         return (("%s:\n"
219                         " * Coefficients: %s\n"
220                         " * Intercept = %.5f\n") %
221                         (self.__class__.__name__,
222                         numpy.array_str(self.coef_, precision=4),
                        223                     self.intercept_))
```

3.4. Tmask.py

```
1  ============ 'tmask.py' ============
2
3     import logging
4     import numpy as np
5
6  from ccd.models import robust_fit 7
8  log = logging.getLogger(__name__) 9   10
11            def tmask_coefficient_matrix(dates, avg_days_yr):
12            """Coefficient matrix that is used for Tmask modeling
13
```

```
                Args:
                dates: list of ordinal julian dates

                Returns:
                Populated numpy array with coefficient values
                """
                annual_cycle = 2*np.pi/avg_days_yr
                observation_cycle = annual_cycle / np.ceil((dates[-1] - dates[0]) / avg_days_yr) 22
        matrix = np.ones(shape=(dates.shape[0], 5), order='F')
        matrix[:, 0] = np.cos(annual_cycle * dates)
        matrix[:, 1] = np.sin(annual_cycle * dates)
        matrix[:, 2] = np.cos(observation_cycle * dates)
        matrix[:, 3] = np.sin(observation_cycle * dates) 28
29          return matrix

        def tmask(dates, observations, variogram, bands, t_const, avg_days_yr):
        """Produce an index for filtering outliers.

                        Arguments:
                        dates: ordinal date values associated to each n-moment in the
                        observations
                        observations: spectral values, assumed to be shaped as
                        (n-bands, n-moments)
                        bands: list of band indices used for outlier detection, by default
                        bands 2 and 5.
                        t_const: constant used to scale a variogram value for thresholding on
                        whether a value is an outlier or not

        Return: indexed array, excluding outlier observations.
        """
        # variogram = calculate_variogram(observations)
        # Time and expected values using a four-part matrix of coefficients.
48          # regression = lm.LinearRegression()
49              regression = robust_fit.RLM(maxiter=5)

    tmask_matrix = tmask_coefficient_matrix(dates, avg_days_yr) 52
    # Accumulator for outliers. This starts off as a list of False values 54    # because we don't assume
anything is an outlier.
        _, sample_count = observations.shape
        outliers = np.zeros(sample_count, dtype=bool)

    # For each band, determine if the delta between predeicted and actual 59          # values exceeds the
threshold. If it does, then it is an outlier.
                for band_ix in bands:
                fit = regression.fit(tmask_matrix, observations[band_ix])
                predicted = fit.predict(tmask_matrix)
                outliers += np.abs(predicted - observations[band_ix]) > variogram[band_ix] * t_const

        # Keep all observations that aren't outliers.
        return outliers
                67                                  # return dates[~outliers], observations[:, ~outliers]
```

4. Notebooks

4.1. Avaliacao_exatidao_pyccd

```
1  ============ 'avaliacao_exatidao_pyccd.py' ============
3  # -*- coding: utf-8 -*-
4  """Avaliacao_exatidao_pyccd.ipynb
5
6  Automatically generated by Colaboratory.
7
8  Original file is located at
   9                    https://colab.research.google.com/drive/1jB6l22lRGW6JVRfBt8MXfpuwRAWuKpdl
10
11 # Imports e configurações iniciais 12  """
13 import sys, os
14
15     #ee.Authenticate() # quando este passo nao foi feito obtive erro no geemap.shp_to_ee
16     #ee.Initialize(project='ee-isadgt') #new behaviour verified since December/2023 - need to include the google cloud
       project name on the Initialize method.
17
18     import os #, glob
19     from datetime import datetime #, ee
20     #import rasterio
21     import pandas as pd
22     import numpy as np
23     import geopandas as gpd
24     #import rtree, pygeos, shapely
25     #import haversine as hs # Novo #not used apparently
26     #from haversine.haversine import Unit #not used apparently
27
28     import warnings
29     warnings.filterwarnings('ignore')
30
31 """# Variáveis e inputs
32
33 Caminhos 34  """ 35  """ 36  #caminho para o csv (leitura) 37  csv_path = str(csv_file_ccd) #
r"/content/drive/MyDrive/colab_DGT/csv_1_20240118122930.csv"
38
39 #caminho para gravar o csv pre-processado 40  #csv_preprocessed_path =
r"/content/drive/MyDrive/colab_DGT/csv_teste_preprocessed.csv" 41  csv_preprocessed_path =
str(csv_file_ccd.with_suffix(''))+'pre_proc.csv' 42  43  #caminho para a base de dados de validação 44  #path_adjusted_bdr =
'DGT_buffers_validacao_TNE_2018_2021/Ajustado_Novas_Regras_Alteracao/BDR_CCDC_TNE_Adju sted.shp' 45
path_adjusted_bdr = base_path / 'BDR_300_artigo' / 'BDR_CCDC_TNE_Adjusted.shp'
46
47 """ 48  """Datas"""
49
50     # # datas do filtro das datas da análise (DGT 300)
51     # ############ Não alterar #################
52     # dt_ini = '2018-09-12' # data inicial
53     # dt_end = '2021-09-30' # data final
54
55 # """Outros"""
56
57     # # Margem de tolerância entre a quebra do Modelo e do Analista
58     # theta = 60 # +/- theta dias de diferenca 59
60     # # bandar a filtrar com base na magnitude
```

```
61      # bandFilter = None #não implementado ainda - não mexer
62
63  """# Validação resultados CSV
64
65  ## função pré-processamento 66   """
67              import csv
68              def inferDelimiter(pathDF):
69              with open(pathDF, 'r') as csvfile:
70              dialect = csv.Sniffer().sniff(csvfile.readline())
71              return dialect.delimiter
72
73  def convertDate(data):
74  """Obtem ano, mês e dia a partir de data no formato YYYY-MM-DD""" 75   data = data.split('-')
76          y = int(data[0])
77          m = int(data[1])
78          d = int(data[2])
79          return y,m,d
80
81  def filterDate(pathDF, dataI, dataF,bandFilter, mag = None):
82      """ 83      Reduz o número de linhas do data frame de entrada, removendo as linhas fora do período de análise e
84              para o limite estabelecido de magnitude máxima.
85              Entrada:
86      pathDF: caminho do Data Frame do CCDC 87          dataI: String com a data inicial na forma = 'AAAA-MM-DD' (e.g. a
data inicial dos analistas nos pontos DGT 300) 88          dataF: String com a data final na forma = 'AAAA-MM-DD' (e.g. a data
final dos analistas nos pontos DGT 300)
89                  bandFilter: String com a banda para a qual se deseja filtrar os dados. A esta banda é aplicado o criterio do
                    mag.
90                  mag: Número com o limite da magnitude, e.g 0 só serão utilizadas as linhas com magnitudo menor ou igual a
                    zero
91                  Saída:
92      Data Frame filtrado 93      """
94                  # Data Frame CCDC
95                  if pathDF.endswith('.csv'):
96                  delimiter = inferDelimiter(pathDF)
97                  df = pd.read_csv(pathDF, delimiter = delimiter) 98          if pathDF.endswith('.pkl'):
99                      df = pd.read_pickle(pathDF)
100
101                     for dtCol in df.columns:
102                     if 'tBreak' in dtCol or 'tEnd' in dtCol or 'tStart' in dtCol:
103                     mask = df.loc[:, dtCol] == 0
104                     df[dtCol] = pd.to_datetime(df[dtCol], unit = 'ms') 105          df.loc[mask, dtCol] = np.nan 106   elif
                        'End_S' in dtCol:
107                     df[dtCol] = pd.to_datetime(df[dtCol]) # Esta coluna inicialmente esta em formato texto
108                     df.rename(columns={ 'Unnamed: 0':'IDCCDC'}, inplace=True) 109
110                 if mag != None:
111                 # caso haja magnitude limite, colocar tudo como NAT que seja acima deste limite
112                 df.loc[df[bandFilter] > mag, 'tBreak'] = pd.to_datetime(np.nan) 113          df = df.copy() 114          else:
115                     df = df.copy()
116
117             # filtro das datas
118             yi, mi, diai = convertDate(dataI)
119             fltInicial = datetime(yi, mi, diai)
120             yf, mf, diaf = convertDate(dataF)
121             fltFinal = datetime(yf, mf, diaf)
122
123             # 1 Adiciona a coluna com a menor data de start do fit
```

```
124             df['startMin'] = df.groupby(['coord_ccdc'])['tStart'].transform('min') 125
126             # 2 Adiciona o número de breaks existentes num grupo de IDCCDC, independente de fltInicial e fltFinal
127             df['numBreak'] = np.ceil(df.groupby(['coord_ccdc'])['changeProb'].transform('sum' ))
128
129     # Colocar Nat nas probabilidades fracionadas
130     df.loc[((df.changeProb > 0) & (df.changeProb < 1)), 'tBreak'] = pd.to_datetime(np. nan) 131
132             # 3 Verifica se se os breaks estão dentro do período de análise e transforma em NaT todos os que não estão
133             df['breaks_in_tmask'] = (~df.tBreak.isnull()).astype(int)
134             df.loc[(df['tBreak'] <= fltInicial) | (df['tBreak'] >= fltFinal), 'breaks_in_tmask'] = 0
135             df.loc[(df['tBreak'] <= fltInicial) | (df['tBreak'] >= fltFinal), 'tBreak'] = np. nan
136
137             # Mascaras necessárias
138             # a) Verifica os breaks NaT para as linhas com mais de 1 break
139             mask = pd.Series(np.zeros(len(df),dtype=bool),index = df.index) 140       mask.loc[(df.tBreak.isnull()) &
                (df.numBreak > 1)]= True #cond3 141
142             # b) Verifica nas linhas de 1 break e sejam nulos qual é aquele que tem o início da série,
143             #pois caso esteja fora da data de análise deve ser eliminado
144             nmask = pd.Series(np.zeros(len(df),dtype=bool),index = df.index)
145             nmask.loc[(df.tBreak.isnull()) & (df.numBreak == 1) & (df.breaks_in_tmask == 0) & (df.tStart == df.startMin)]=
                True
146
147             # Aplica as mascaras acima e gera um novo DF
148             subset_Filtro = df[((mask == False) & (nmask == False))].copy() 149
150             # c) Calcula quantos linhas há por IDCCDC e caso ainda existam 2 significa que o break está dentro do período de
                análise e o fit final, sem break
151             # deve ser eliminado
152             smask = pd.Series(np.zeros(len(subset_Filtro),dtype=bool),index = subset_Filtro. index)
153             smask.loc[(subset_Filtro.groupby(['coord_ccdc'])['IDCCDC'].transform('count') == 2
) & (subset_Filtro.changeProb == 0) & (df.numBreak == 1)] = True 154          subset_Filtro =
subset_Filtro[(smask == False)].copy()
155
156             # d) Para os IDCCDC que apresentam linhas com probabilidade fracionada, mantem esta linha, no caso de todas
                estarem fora do período de análise
157             pmask = pd.Series(np.zeros(len(df),dtype=bool),index = df.index)
158             pmask.loc[~((df.changeProb > 0) & (df.changeProb < 1) & (df.tBreak.isnull()) & (df
.groupby(['coord_ccdc'])['tBreak'].transform('count') == 0))]=True
159             subset_Filtro = pd.concat([subset_Filtro,df[pmask == False]])
#subset_Filtro.append(df[pmask == False])
160
161             # e) Para os IDCCDC que tem mais de um break e todos estao fora do periodo e devemos manter o fit final
162             fmask = pd.Series(np.zeros(len(df),dtype=bool),index = df.index)
163             fmask.loc[((df.changeProb == 0) & (df.numBreak > 1) & (df.tBreak.isnull()) & (df.
                groupby(['coord_ccdc'])['tBreak'].transform('count') == 0))]=True
164             subset_Filtro = pd.concat([subset_Filtro,df[fmask]]) #subset_Filtro.append(df[fmask])
165
166
167                 return subset_Filtro
168
169         def spatialJoin(pathPoligonosDGT, dfCCDC):
170         """
171         Realizar o spatial join entre o dataframe do CCDC e os poligonos com alteracoes identificadas pela DGT
172         Entrada:
173         - pathPoligonosDGT: String com o caminho completo dos poligonos desenhados pela DGT174      -
            pathDataFrameCCDC: Data Frame filtrado do CCDC
175     Saida:
```

```
    """
        # 1) ABRIR OS ARQUIVOS
        ## Poligonos DGT
        gdfVal = gpd.read_file(pathPoligonosDGT)
        gdfVal.to_crs(crs = 'EPSG:3763', inplace = True) # Originalmente eles estao em WGS84 29N converte para ETRS
        ## Pontos ISA

        # 2) CONVERTER O DF PARA GEO DF
        gdfCCDC = gpd.GeoDataFrame(dfCCDC, geometry = gpd.points_from_xy(dfCCDC.longitude, dfCCDC.latitude),
        crs = 4326 )

        ## criar a bordadura
        ###idBord = identity.copy() # cria uma copia do identity gerado acima
        idBord = gdfVal.copy()
        idBord['geometry'] = idBord.geometry.buffer(-10) # reduz a geometria em 10 metros
        idBord.drop(list(idBord.columns)[:-1], axis = 1, inplace = True) #remove todas as colunas menos a da geometria
        idBord['bordadura'] = 1 # cria uma nova coluna para poder identificar a borda dura 192   ## novo identity para termos
        a area da borda dura 193
        ###identity = gpd.overlay(identity, idBord, how='identity')
        identity = gpd.overlay(gdfVal, idBord, how = 'identity')

        # Como o poligono inicial nao tinha a coluna de bordadura, há feições onde
        # temos 1 e Nulos, com a linha abaixo invertemos o campo onde era Nullo passa a True
        # e onde era 1 passa para False, ou 1 e 0
        identity.bordadura = identity.bordadura.isnull()
        # Convertemos o resultado para WGS84
        identity.to_crs(crs = 'EPSG: 4326', inplace = True) 203
#


## As datas da DGT estao no formato (20200103) e precisam ser convertidas 208   for dataCol in ['data_0',
'data_1', 'data_2', 'data_3']: 209       # primeiro converter para datetime
                maskZero = pd.Series(np.zeros(len(identity),dtype=bool))
                erro = identity[dataCol].isnull()
                identity.loc[erro, dataCol] = 0
                # converter tudo para inteiros e onde for 0 indicar 1970
                identity[dataCol] = identity[dataCol].astype(int)
                maskZero = identity.loc[:, dataCol] == 0
                identity.loc[maskZero, dataCol] = 19700101
                # converter para datetime
                identity[dataCol] = pd.to_datetime(identity[dataCol], format = '%Y%m%d')
                identity.loc[maskZero, dataCol] = np.nan

        # 4) SPATIAL JOIN ENTRE OS CENTROIDES DO CCDC COM OS BUFFERS DE 200 METROS
        subset = gpd.sjoin(gdfCCDC, identity, how='inner')
        subset.reset_index(inplace = True)
        subset['buffer_ID'] = subset.buffer_ID.astype('int') 225
        """
        Descobrir quais linhas precisam ser duplicadas.
        Pressupondo que não é possível ter informação da 'data_3' sem existir a 'data_1'229     é possível filtrar e verificar a
        negação de quais dados são nulos e depois somar
   o reultado.
   0 = False False: não há data_1 e nem data_3 232     1 = True  False: existe data_1 e não data_3 233     2 = True  True:
existem data_1 e Data_3
    """
        cond = ~subset.filter(items=['data_1', 'data_3']).isnull()
```

```
236         subset['analistas'] = cond.sum(axis=1)
237         subset.loc[subset['analistas'] == 0, 'exists_event'] = False # Analista nao identificou nada
238         subset.loc[subset['analistas'] > 0, 'exists_event'] = True # Analista identificou alteracao
239
240 """
241    CRIA UM DF TEMPORARIO PARA COPIAR AS LINHAS ONDE EXISTEM A 'DATA_3' E INSERE ESTA DATA NO
CAMPO 'DATA1_Z' 242    DEPOIS ADICIONA ISTO AO DATA FRAME ORIGINAL 243    """
244         subset['data1_z'] = ''
245         # criar coluna para as datas anteriores
246         # subset['data0_z'] = ''
247         subset['nome'] = '' # teste para nomear os analistas
248         subset['tipo'] = ''
249         subset['classeAnterior'] = ''
250         subset['classeAtual'] = ''
251         dfTemp = pd.DataFrame(columns = subset.columns) 252   for row in subset.itertuples():
253     # verifica se há duas datas e duplica a linha 254         if row.analistas
== 2:
255                     dfTemp = pd.concat([dfTemp, subset[subset.index==row.Index]],ignore_index=
False)#dfTemp.append(subset[subset.index == row.Index], ignore_index=False)
256                     dfTemp.data1_z = dfTemp.data_3
257                     # capturar o valor da data_2
258                     # defTemp.data0_z = dfTemp.data_2
259                     dfTemp.nome = 'B' # teste para nomear os analistas
260                     dfTemp.tipo = dfTemp.tipo_2
261                     dfTemp.classeAtual = dfTemp.classe_3
262                     dfTemp.classeAnterior = dfTemp.classe_2
263
264         subset.data1_z = subset.data_1
265         # capturar o valor da data_0
266         # subset.data0_z = subset.data_0
267         subset.nome = 'A' # teste para nomear os analistas
268         subset.tipo = subset.tipo_1
269         subset.classeAtual = subset.classe_1
270         subset.classeAnterior = subset.classe_0 271
272 subset = pd.concat([subset, dfTemp],ignore_index=False)#subset.append(dfTemp, ignore_index=False)
273
274         # Contagem do numero de breaks
275         subset['Valid_breaks'] = np.ceil(subset.groupby(['coord_ccdc', 'nome'])['changeProb'
].transform('sum'))
276
277         # COLUNA DO DELTA MIN
278         subset['delta_min'] = (subset.data1_z - subset.tBreak).dt.days
279         subset.drop(['data_1', 'data_3', 'tipo_1', 'tipo_2','classe_0', 'classe_1', 'classe_2', 'classe_3'], axis = 1, inplace = True)
          280
281                     # verificar quais colunas tem magnitude de indices
282                     mags = [ t for t in subset.columns if 'magnitude' in t and not 'B' in t]
283                     ordem = [ 'coord_ccdc','buffer_ID', 'IDCCDC', 'altera', 'changeProb'] + mags + [ 'tBreak', 'data1_z',
284                     'bordadura', 'classe2018', 'classe2019', 'classe2020','classe2021', 'classeAnterior','tipo',
285                     'classeAtual', 'analistas', 'nome', 'exists_event', 'Valid_breaks' ,
'delta_min', 'geometry']
286
287
288 return subset[ordem], subset
289
290
291         def preprocessCsvS2(csv_s2):
```

```
292         csv_s2 = csv_s2.copy()
293         from ast import literal_eval
294         #do some processing on the csv
295         # Selecionar as colunas a explodir e as dos coeficientes
296         tabExplode = []
297         tabCoefs = []
298         for c in csv_s2.columns:
299         if 'coefs' in c or 'magnitude' in c or 'rmse' in c: 300             tabExplode.append(c) 301             if 'coefs' in c:
302                 tabCoefs.append(c)
303                 tabExplode = tabExplode + ['changeProb', 'tBreak', 'tEnd', 'tStart'] 304
305 #convert from string of list to list 306   for col in
tabExplode:
307                 try:
308                 csv_s2[col] = csv_s2[col].apply(literal_eval)
309                 except: #sometimes CCDC returns 'Infinity' or 'NaN' as a rmse value, which results in literal_eval not working
310                 #csv_s2[col] = csv_s2[col].apply(lambda x: x.replace('Infinity','9999999'))
311                 #csv_s2[col] = csv_s2[col].apply(lambda x: x.replace('NaN','-9999999'))
312                 csv_s2[col] = csv_s2[col].apply(literal_eval)
313                 #convert lat long separated by comma to separated by point
314                 #csv_s2['Lat'] = csv_s2['Lat'].apply(lambda x: x.replace(",",".")) 315   #csv_s2['Lon'] =
                    csv_s2['Lon'].apply(lambda x: x.replace(",",".")) 316
317         #explode
318         csv_s2 = csv_s2.explode(tabExplode)
319
320         csv_s2['End_S'] = '2023-09-29'
321         csv_s2['coord_ccdc'] = list(zip(csv_s2.Lat, csv_s2.Lon))
322         csv_s2['Dist_Point'] = -1#"
323         csv_s2['Point_Val'] = -1#"
324
325 #convert date columns from float to int 326   for col in
['tBreak', 'tEnd', 'tStart']:
327     csv_s2[col] = csv_s2[col].astype('int64') 328
329 csv_s2.rename(columns={'Lat':'latitude','Lon':'longitude'}, inplace=True)
330
331 return csv_s2
332
333 """## Carregar csv e fazer pré-processamento"""
334
335     # #abrir csv a partir do google drive
336     # csv_s2 = pd.read_csv(csv_path)
337
338     # #correr pre-processamento
339     # csv_s2 = preprocessCsvS2(csv_s2)
340
341     # #guardar resultado como csv no google drive
342     # if os.path.exists(csv_preprocessed_path):
343     #   os.remove(csv_preprocessed_path)
344     # csv_s2.to_csv(csv_preprocessed_path)
345
346 # """## Filtrar datas
347
348     # Limitar análise ao período considerado pelos analistas DGT
349     # """
350
351     # #correr filtro de datas
352     # ccdcFiltro = filterDate(csv_preprocessed_path,dt_ini, dt_end, bandFilter) 353
```

```
# """## Spatial join

    # Faz join dos pontos do csv com a informação de referencia da DGT (300 buffers). É associada aos pontos a informação da validação - data de alteração, tipo, classes, etc.
    # """

    # #executa o join
    # ccdcVal, ccdcVal_T = spatialJoin(os.path.join(PATH,path_adjusted_bdr), ccdcFiltro)

# """## Validação

    # Faz a validação da deteção - compara resultado do modelo (ccd) com dados de referência DGT
    # """

# função de validação do data frame 368   def valPol(df, theta):
        """
        Esta função recebe o geodataframe gerado no spatialJoin() e contabiliza as métricas de positivos e negativos.
        A Saída é a matriz com os cálculos e um dicinário com as métricas contabilizadas.

    """

        # transforma a coluna de delta min para valor absoluto e cria uma nova coluna com o mínimo delta min por ponto
        df.reset_index(inplace = True)
        print('chegou aqui')
        original_delta_min = df['delta_min'].copy()
        df['delta_min'] = abs(df['delta_min'].fillna(99999)) # substitui os nullos para evitar que sejam os minimos
        df['Min_delta_min'] = df.groupby(['coord_ccdc', 'nome'])['delta_min'].transform( 'min') # calcula o valor minimo por ponto
        df['delta_min'] = abs(original_delta_min) # retorna o valor absoluto da coluna original
        df['Min_delta_min'] = df['Min_delta_min'].replace(99999,np.nan) # substitui os 99999 por nullos

bf = df.copy()

        bf['Valid_breaks'] = bf.groupby(['coord_ccdc', 'nome']).transform('count')[['tBreak' ]] # verifica os breaks validos por pontos
        # SE O TBREAK FOR OBJETO ELE JAMAIS SERA NULO, CONVERTER PARA DATA.
        bf.tBreak = pd.to_datetime(bf.tBreak)
        bf.tStart = pd.to_datetime(bf.tStart)
        bf.tEnd = pd.to_datetime(bf.tEnd)
        bf.analistas = bf.analistas.astype(int)
        bf.exists_event = bf.exists_event.astype(int)
        bf.buffer_ID = bf.buffer_ID.astype(int)
        bf.IDCCDC = bf.IDCCDC.astype(int)

        ## ALGUMAS MASCARAS INICIAIS NECESSARIAS
        # mascara dos breaks a mais que analistas ainda em reformulacao 398
        # PARA O CASO DE TER SOMENTE UM BREAK FP E DOIS ANALISTAS PARA NAO TER DUPLICACAO
        mask = pd.Series(np.zeros(len(bf),dtype=bool), index= bf.index)
        mask.loc[(bf.analistas == 2) & (bf.Valid_breaks < bf.analistas) ] = True #& (bf.delta_min > theta) 402
bf.loc[mask, 'Min_delta_min'] = bf.loc[mask].groupby(['coord_ccdc'])['delta_min']. transform('min')

        # Contabilizar
        # colocar todos os VP (delta_min <=31)
```

```
        #VP
        bf.loc[( (bf.delta_min <= theta) & (~bf.tBreak.isnull()) & (bf.analistas > 0) ), 'VP'] = 1
        # #FP
        # # sem a condição da magnitude ou (changeProb ==1) serao selecionados os que devem ser negativos
        # bf.loc[( (bf.analistas == 0) & (bf.ndvi_magnitude != 0) & (~bf.tBreak.isnull())), 'FP' ] = 1 #FP puro
        # bf.loc[( (bf.delta_min > theta) & (bf.ndvi_magnitude != 0) & ( (bf.delta_min == bf.Min_delta_min) & (~bf.Min_delta_min.isnull()) ) )  , 'FP' ] = 1
        # bf.loc[( (bf.delta_min > theta) & (bf.ndvi_magnitude != 0) & (bf.analistas == 1)  ) & (~bf.tBreak.isnull()), 'FP' ] = 1
        #FP
        # sem a condição da magnitude ou (changeProb ==1) serao selecionados os que devem ser negativos
        bf.loc[( (bf.analistas == 0) & (~bf.tBreak.isnull())), 'FP' ] = 1 #FP puro 417   bf.loc[( (bf.delta_min > theta) & ( (bf.delta_min == bf.Min_delta_min) & (~bf. Min_delta_min.isnull()) ) ) , 'FP' ] = 1
        bf.loc[( (bf.delta_min > theta) & (bf.analistas == 1) ) & (~bf.tBreak.isnull()), 'FP' ] = 1
        #FN
        bf.loc[( (bf.analistas > 0) & (bf.tBreak.isnull()) ), 'FN' ] = 1 # FN puro
        # falsos negativos que precisam ser contabilizado para os FPs
        bf.loc[(bf.analistas == 1) & (bf.Valid_breaks == 1) & (bf.FP == 1), 'FN'] = 1 # parece funcionar
        bf.loc[(bf.analistas == 2) & (bf.Valid_breaks == 3) & (bf.FP == 1) , 'FN'] = 1

        #VN
        bf.loc[( (bf.analistas == 0) & (bf.tBreak.isnull()) ), 'VN' ] = 1 427
        # converter os NaN para 0
        bf[['VP', 'FP', 'FN', 'VN']] = bf[['VP', 'FP', 'FN', 'VN']].fillna(0) 430
        # verificar os breaks que nao foram classificados
        # para isso gero uma coluna total onde somo todas as metricas, as linhas onde ha 0 nao foram classificadas
        bf['total'] = bf.VP + bf.FP +bf.FN + bf.VN
        mask = pd.Series(np.zeros(len(bf),dtype=bool), index= bf.index) #mascara
        # agrupar por coordenada e t break, assim as somente os breaks que nao foram validados para nenhum analista terao valor 0
        mask.loc[(bf.groupby(['coord_ccdc','tBreak'])['total'].transform('sum')==0) & (bf.
analistas == 2) & (bf.Valid_breaks > bf.analistas)] = True
        # neste grupo selecionado devo procurar aquele que tem menor distancia para um analista e classificar como FP
        mask2 = bf[mask].groupby(['coord_ccdc'])['delta_min'].transform('min') == bf. delta_min[mask]
        # agora classificar os candidatos que atendem as duas mascaras
        bf.loc[(mask & mask2), ['FP']] = 1

        # Ajuste FN
        # se for na célula anterior isso contará para o total e a mascara anterior não será feita em alguns pontos onde deve ser feita
        bf.loc[((bf.FP ==1) & (bf.analistas == 1) & (bf.delta_min == bf.Min_delta_min) & (bf .Valid_breaks == 2))      , 'FN' ] = 1
        bf.loc[((bf.FP ==1) & (bf.analistas == 1) & (bf.delta_min == bf.Min_delta_min) & (bf .Valid_breaks == 3))      , 'FN' ] = 1
        bf.loc[(bf.analistas == 2) & (bf.Valid_breaks == 1) & (bf.VP == 0), 'FN'] = 1 447   bf.loc[(bf.analistas == 2) & (bf.Valid_breaks == 2) & (bf.FP == 1), 'FN'] = 1 448   #return bf
        # Bloco para corrigir o problema de quando as duas datas DGT estão mais próximas do mesmo break
        # listar as coordenadas que tem o problema com mesmo break classificado
        listCoord = list(bf.coord_ccdc[(bf.groupby(['coord_ccdc','tBreak'])['total'].
transform('sum') == 0) & (bf.analistas == 2) & (bf.Valid_breaks == 2)]) 452   #return listCoord
        # dividir o data frame em dois para poder limpar as linhas com problema
        bf_filter = bf.loc[~bf.coord_ccdc.isin(listCoord)].copy()
        # limpeza
        bf_remove_lines = bf.loc[bf.coord_ccdc.isin(listCoord)].copy()
        # zerar todas as métricas para poder recalcular
        bf_remove_lines.loc[:, ['VP','VN','FP', 'FN']] = 0
        #return bf_remove_lines
```

```
460         bf_removed = bf_remove_lines.groupby(['buffer_ID','IDCCDC']).apply(testeRemove).copy () # função de remoção
            461   #return bf_removed 462   try:
463       bf_removed = bf_removed.drop(columns=['buffer_ID','IDCCDC']).reset_index() # evitar problema de indece
dup. 464   except:
465             pass
466             # Agora teremos somente duas linhas por ponto que são obrigatóriamente FP ou VP 467   #VP
468         bf_removed.loc[( (bf_removed.delta_min <= theta) ), 'VP'] = 1
469         #FP, FN
470         bf_removed.loc[( (bf_removed.delta_min > theta) ), ['FP', 'FN']] = 1
471         # unir os dois dfs novamente
472         bf_final = pd.concat([bf_filter, bf_removed])#bf_filter.append(bf_removed) 473
474         # remover aqueles que nao possuem metrica
475         bf_final = bf_final[(bf_final.VP > 0) | (bf_final.FP > 0) | (bf_final.FN > 0) | ( bf_final.VN > 0) ].copy()
476         # remover aqueles que apresentam as classes especificas
477         bf_final = bf_final[~(bf_final.tipo.isin(['Agricultura','Agua']))].copy()
478
479
480         # verificar quais colunas tem magnitude de indices
481         mags = [ t for t in bf_final.columns if 'magnitude' in t and not 'B' in t]
482         # colunas para retornar um DF mais limpo
483         c = ['buffer_ID', 'IDCCDC', 'coord_ccdc', 'changeProb'] + mags + ['tBreak',
484         'data1_z', 'analistas', 'nome', 'exists_event', 'Valid_breaks', 485         'delta_min', 'Min_delta_min', 'VP', 'FP', 'FN', 'VN']
            #geometry 486   # também poderá retornar o DF todo classificado, em processo.
487   return bf_final[c], bf_final
488
489
490   # função para realizar a limpeza de linhas indesejadas 491   def
testeRemove(groupedby):
492                 min_delta_min = groupedby['Min_delta_min'].min()
493                 #remove rows only if there is more than 1 row per point, the number of analyst dates is not zero and
                    min_delta_min is greater than zero.
494                 if len(groupedby) > 1 and groupedby.analistas.min() > 0 and min_delta_min >= 0:
495                 Bj, Ai = groupedby.loc[groupedby['delta_min']==min_delta_min][['tBreak', 'data1_z']].values[0]
496                 #remove rows that contain Ai or Bj (other than the row with the min_delta_min) 497      mask =
                    ((groupedby['tBreak'] == Bj) | (groupedby['data1_z'] == Ai)) & (groupedby ['delta_min']!=min_delta_min)
498                       groupedby = groupedby[~mask]
499
500   return groupedby
501
502       # #faz a validação da deteção
503       # DF_FINAL, DF_FINAL_T = valPol(ccdcVal_T, theta) #funcoes.valPol 504
505   # """**Resultados da validação**"""
506
507       # #delimita análise apenas para pontos referentes a transições entre Pinheiro Bravo e
Eucalipto para Superfície sem vegetação, herbáceas e matos
508       # #elimina também pontos da bordadura
509       # df_aux = DF_FINAL_T.copy()
510       # df_aux = df_aux.loc[(df_aux.altera=="Sem Alteracao")|((df_aux.altera=="Com
            Alteracao")&(df_aux.classeAnterior.isin(['Pinheiro
bravo','Eucalipto']))&(df_aux.classeAtual.isin(['Superficie sem vegetacao
                                                escura','Superficie sem vegetacao clara','Vegetacao herbacea espontanea','Matos'])))]
511       # df_aux = df_aux.loc[df_aux.bordadura==0]
512
513       # #imprime f1-score, erro e omissão e erro de comissão
514       # cm = df_aux.FP.sum()/(df_aux.FP.sum()+df_aux.VP.sum())
515       # om = df_aux.FN.sum()/(df_aux.FN.sum()+df_aux.VP.sum())
```

```
516     # f1 = 2*(1-om)*(1-cm)/(2-om-cm)
517     # print('F1-score = {}%'.format(round(100*f1,2)))
518     # print('Omission error = {}%'.format(round(100*om,2)))
519     # print('Commission error = {}%'.format(round(100*cm,2))) 520
521     #Benchmark - Resultados com CCDC do Google Earth Engine
522     ## Parâmetros utilizados
523     ## cloud mask - s2cloudless
524     ## minYears = 1
525     ## chi-square = 0.999
526     ## lambda = 200 (values on 10,000 scale) 527   ## n_obs = 6
528     ## ccdc breakpoint bands = NDVI, B3, B12
529     ## ccdc tmask bands = B3, B12
530
531     ## F1-score = 82.07%
532     ## Omission error = 15.04%
533     ## Commission error = 20.63%
534
535     # params_ccdc = {
536     #     'bandas_breakpoint':['ndvi', 'B3', 'B12'], #bandas efetivamente utilizadas pelo CCDC para identificar breakpoints
537     #     'bandas_tmask':['B3', 'B12'], #bandas utilizadas pelo CCDC como tmask (deteção de nuvens)
538     #     'minObs':6, #The number of observations required to flag a change
539     #     'chiSquare':0.999, #The chi-square probability threshold for change detection in the range of [0, 1]
540     #     'minYears':1,#1.33, #Factors of minimum number of years to apply new fitting 541   #     'dateForm':2, #date format.
        Use 2 for unix time in milliseconds
542     #     'Lambda':200, #lambda para NDVI normalizado * 10000
543     #     'maxIter':25000 #maximum number of runs for regression convergence
544     #     }
545
546     # params_ImgCol = {
547     #     'nameImage':"COPERNICUS/S2_SR_HARMONIZED",
#https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2_SR_HARMONIZ ED#description
548     #     'date_start':'2016-01-01',
549     #     'date_end':'2021-12-31',
550     #     'indices':['ndvi'], #indices a serem adicionados
551     #     'cloudFilter':'s2cloudless', #algoritmo de filtragem das nuvens/sombras ('SCL' ou 's2cloudless')
552
553     #     'bandas':['ndvi', 'B2', 'B3','B4','B8', 'B11', 'B12'],  #Bandas a serem selecionadas para a coleção que entra no CCDC
554     #     'banda':'ndvi' #indicar que é um parametro para os graficos apenas. # banda para qual desejamos a informacao do
        CCDC --- OBS: algumas funcoes estao feitas apenas para o ndvi. Ver esse parametro banda com mais cuidado
555     #     }
556
557     # save_name="DF_VAL_DETECAO_{}_{}.parquet".format(str(theta),funcoes.fromParamsReturnNam
        e(params_ImgCol, params_ccdc, 9999, 9999, 210)).replace('LON999900000E_LAT999900000N_','')
558     # #filename_apos_ccdc_analista_val_detecao = os.path.join(PATH, Outputs,
CCDC_Data_Frame, DFInfoCompleta, save_name)
559     # filename_apos_ccdc_analista_val_detecao = os.path.join(PATH, 'Outputs',
'CCDC_Data_Frame', 'DFInfoCompleta_new', save_name)
560
561 # df_ccdc_ee = pd.read_parquet(filename_apos_ccdc_analista_val_detecao) 562
563     # DF_FINAL_T['lat_8dg'] = DF_FINAL_T.latitude.round(8)
564     # DF_FINAL_T['lon_8dg'] = DF_FINAL_T.longitude.round(8)
565
566     # #pontos em comum c/ a amostra de 20mil
567     # #mask_pts = DF_FINAL_T.coord_ccdc.values #nao funcinou bem - talvez por conta das casas decimais
568     # mask_pts_lat = DF_FINAL_T.lat_8dg.values
569     # mask_pts_lon = DF_FINAL_T.lon_8dg.values
```

```
570
571         # df_ccdc_ee['lat_8dg'] = df_ccdc_ee.latitude.round(8)
572         # df_ccdc_ee['lon_8dg'] = df_ccdc_ee.longitude.round(8)
573         # sel_cccdc_ee = df_ccdc_ee.loc[(df_ccdc_ee.lat_8dg.isin(mask_pts_lat))&(df_ccdc_ee.lon_8dg.isin(mask_p
            ts_lon))].copy()
574
575         # #mask back - because some points from the new csv file were discarded for some reason (likely LC class mask, due to
            positional approximation margin)
576         # #this means we are going to compare only the same set of points on both datasets
577         # mask_pts_lat_ee = sel_cccdc_ee.lat_8dg.values
578         # mask_pts_lon_ee = sel_cccdc_ee.lon_8dg.values
579
580   # DF_FINAL_T_masked =
DF_FINAL_T.loc[(DF_FINAL_T.lat_8dg.isin(mask_pts_lat_ee))&(DF_FINAL_T.lon_8dg.isin(mas k_pts_lon_ee))].copy()
581
582         # #delimita análise apenas para pontos referentes a transições entre Pinheiro Bravo e
Eucalipto para Superfície sem vegetação, herbáceas e matos
583         # #elimina também pontos da bordadura
584         # df_aux = DF_FINAL_T_masked.copy()
585         # df_aux = df_aux.loc[(df_aux.altera=="Sem Alteracao")|((df_aux.altera=="Com
            Alteracao")&(df_aux.classeAnterior.isin(['Pinheiro
bravo','Eucalipto']))&(df_aux.classeAtual.isin(['Superficie sem vegetacao
escura','Superficie sem vegetacao clara','Vegetacao herbacea espontanea','Matos'])))]
586         # df_aux = df_aux.loc[df_aux.bordadura==0]
587
588         # #imprime f1-score, erro e omissão e erro de comissão
589         # cm = df_aux.FP.sum()/(df_aux.FP.sum()+df_aux.VP.sum())
590         # om = df_aux.FN.sum()/(df_aux.FN.sum()+df_aux.VP.sum())
591         # f1 = 2*(1-om)*(1-cm)/(2-om-cm)
592         # print('Results with pyccd for the 20k sample')
593         # print('F1-score = {}%'.format(round(100*f1,2)))
594         # print('Omission error = {}%'.format(round(100*om,2)))
595         # print('Commission error = {}%'.format(round(100*cm,2)))
596
597         # #delimita análise apenas para pontos referentes a transições entre Pinheiro Bravo e
Eucalipto para Superfície sem vegetação, herbáceas e matos
598         # #elimina também pontos da bordadura
599         # df_aux = sel_cccdc_ee.copy()
600         # df_aux = df_aux.loc[(df_aux.altera=="Sem Alteracao")|((df_aux.altera=="Com
            Alteracao")&(df_aux.classeAnterior.isin(['Pinheiro
bravo','Eucalipto']))&(df_aux.classeAtual.isin(['Superficie sem vegetacao
escura','Superficie sem vegetacao clara','Vegetacao herbacea espontanea','Matos'])))]
601         # df_aux = df_aux.loc[df_aux.bordadura==0]
602
603         # #imprime f1-score, erro e omissão e erro de comissão
604         # cm = df_aux.FP.sum()/(df_aux.FP.sum()+df_aux.VP.sum())
605         # om = df_aux.FN.sum()/(df_aux.FN.sum()+df_aux.VP.sum())
606         # f1 = 2*(1-om)*(1-cm)/(2-om-cm)
607         # print('Results with gee for the 20k sample')
608         # print('F1-score = {}%'.format(round(100*f1,2)))
609         # print('Omission error = {}%'.format(round(100*om,2)))
610         # print('Commission error = {}%'.format(round(100*cm,2)))
611
612         # df_aux_gee = sel_cccdc_ee.copy()
613         # df_aux_gee = df_aux_gee.loc[(df_aux_gee.altera=="Sem Alteracao")|((df_aux_gee.altera=="Com
Alteracao")&(df_aux_gee.classeAnterior.isin(['Pinheiro
```

```
bravo','Eucalipto']))&(df_aux_gee.classeAtual.isin(['Superficie sem vegetacao escura','Superficie sem vegetacao clara','Vegetacao herbacea espontanea','Matos'])))]
614       # df_aux_gee = df_aux_gee.loc[df_aux_gee.bordadura==0]
615
616       # df_aux = DF_FINAL_T_masked.copy()
617       # df_aux = df_aux.loc[(df_aux.altera=="Sem Alteracao")|((df_aux.altera=="Com
          Alteracao")&(df_aux.classeAnterior.isin(['Pinheiro
bravo','Eucalipto']))&(df_aux.classeAtual.isin(['Superficie sem vegetacao
escura','Superficie sem vegetacao clara','Vegetacao herbacea espontanea','Matos'])))]
618       # df_aux = df_aux.loc[df_aux.bordadura==0]
619
620   # test = df_aux_gee.set_index(['latitude','longitude']).join(df_aux[['latitude','longitude','VP
','FP','VN','FN']].set_index(['latitude','longitude']),rsuffix='_pyccd')
621
622   # test = test[['VP','FP','VN','FN','VP_pyccd','FP_pyccd','VN_pyccd','FN_pyccd']].reset_index()
623
624   # test.loc[(test.VN==1)&(test.FP_pyccd==1)].to_csv('points_VNGEE_FPpyccd.csv')
```

4.2. Processing.py

```
1  ============ 'processing.py' ============
2
3      import xarray as xr
4      import rioxarray
5      import numpy as np
6      from datetime import datetime, timezone, timedelta 7   import pandas as pd
8  from notebooks.read_files import convertPointToCrs 9   import ccd

10             from rasterio.features import geometry_window
11             from shapely.geometry import Point
12             import rasterio
13             import geopandas as gpd
14             import matplotlib.pyplot as plt
15             import matplotlib.dates as mdates
16             import os
17             #%%
18             def processar_centros_pixeis(shapefile_path, raster_path):
19             # Carregar o shapefile
20             poligonos = gpd.read_file(shapefile_path)
21             caminho_raster = raster_path
22
23             # Lista para armazenar os centros dos pixels para cada geometria
24             todos_centros_pixeis = []
25             poligonos = poligonos[poligonos.is_valid] 26
27                     for index, row in poligonos.iterrows():

28
29                 # Obter a geometria do polígono
30                 geometry = row['geometry']
31
32                         # Carregar o raster
```

```
                    with rasterio.open(caminho_raster) as src:
                    window = geometry_window(src, [geometry])

                        transform = src.window_transform(window)

                    # Obter o tamanho do pixel
                    x_res = transform.a
                    y_res = transform.e

                    # Calcular o deslocamento do centro do pixel
                    x_offset = x_res / 2.0
                    y_offset = y_res / 2.0

                        pixel_centers = []

    # Calcular o centro do pixel para cada pixel na janela for y in range(window.height):
                            for x in range(window.width):
                            # Calcular as coordenadas do centro do pixel
                            pixel_center_x = transform.c + (x * x_res) + x_offset
                            pixel_center_y = transform.f + (y * y_res) + y_offset

    # Verificar se o ponto do centro do pixel está dentro do polígono if Point(pixel_center_x, pixel_center_y).within(geometry): # Armazenar as coordenadas do centro do pixel na lista
        pixel_centers.append((pixel_center_x, pixel_center_y))

                # Adicionar os centros dos pixels desta geometria à lista geral
                todos_centros_pixeis.append(pixel_centers)

    pontos_shapely = [Point(centro) for sublist in todos_centros_pixeis for centro in sublist]

    # Criar um GeoDataFrame a partir da lista de pontos gdf_centros_pixeis =
    gpd.GeoDataFrame(geometry=pontos_shapely)

    return gdf_centros_pixeis

    #%%
    def getTimeSeriesForPoints(tif_names, tif_dates_ord, bandas_desejadas, dados_geoespaciais_metros):

        time_var = xr.Variable('time',tif_dates_ord)
        # Load in and concatenate all individual GeoTIFFs
        tifs_xr = [rioxarray.open_rasterio(i, chunks={'x':10924, 'y':10900}) for i in tif_names]
        geotiffs_da = xr.concat(tifs_xr, dim=time_var).sel(band=bandas_desejadas)
        # COORDENADAS X E Y DOS 10 000 PONTOS ESCOLHIDOS
        points_x_int = xr.DataArray(np.round(dados_geoespaciais_metros.geometry.x.values). astype('int'),
        dims=['location'])
        points_y_int = xr.DataArray(np.round(dados_geoespaciais_metros.geometry.y.values). astype('int'),
        dims=['location'])
```

```
80
81          selection = geotiffs_da.sel(x=points_x_int, y=points_y_int, band=bandas_desejadas)
82          dates = selection.time
83          xs = selection.x
84          ys = selection.y
85          sel_values = selection.values
86
87          # with open('C:/Users/Public/Documents/sel_values.npy','rb') as f:
88          #     sel_values = np.load(f)
89
90          return sel_values, dates, xs, ys
91          #%%
92          def runDetectionForPoint(args, plot_flag=False): # se plot_flag =  False não faz gráficos se True faz
            gráficos
93          i,sel_values, dates, xs, ys, NODATA_VALUE, FOLDER_OUTPUTS, img_collection = args 94
95                  ponto = sel_values[:,:,i]

96
97                  ponto_desejado=xs[i],ys[i]

98
99                          ponto_with_dates = np.column_stack((dates, ponto[:, 0], ponto[:, 1:]))

100
101         mask = (ponto_with_dates != NODATA_VALUE).all(axis=1)
102         ponto_with_dates_filtered = ponto_with_dates[mask].transpose() 103
104     dates, blues, greens, reds, nirs, swir1s, swir2s = ponto_with_dates_filtered 105

106         # Calcular o NDVI
107         ndvis = np.where((nirs + reds) > 0, 10000 * (nirs - reds) / (nirs + reds),
            NODATA_VALUE)

108
109     ponto_with_dates_filtered[1]=ndvis 110

111     ponto_with_dates_filtered1=ponto_with_dates_filtered.transpose() 112

113     ponto_with_dates_filtered2 = ponto_with_dates_filtered1[~np.any(
ponto_with_dates_filtered1 == NODATA_VALUE, axis=1)] 114

115     ponto_with_dates_filtered3=ponto_with_dates_filtered2.transpose() 116

117     dates, ndvis, greens, reds, nirs, swir1s, swir2s = ponto_with_dates_filtered3 118

119         # results = ccd.detect(dates, ndvis, greens, reds, nirs, swir1s, swir2s)
120         results = ccd.detect(dates, ndvis, greens, swir2s) 121
122
123         predicted_values = []
124         prediction_dates = []
125         break_dates = []
126         start_dates = []
127         end_dates=[]
128         coeficientes=[]
```

```
129         prob=[]
130
131             for num, result in enumerate(results['change_models']):
132             days = np.arange(result['start_day'], result['end_day'] + 1)
133             prediction_dates.append(days)
134             break_dates.append(result['break_day'])
135             start_dates.append(result['start_day'])
136             end_dates.append(result['end_day'])
137             prob.append(result['change_probability']) 138
139             intercept = result['ndvi']['intercept']
140             coef = result['ndvi']['coefficients']
141             coeficientes.append(coef)
142
143             coef_str = f"({coef[0]:.2f}, {coef[1]:.2f}, {coef[2]:.2f}, {coef[3]:.2f}, { coef[4]:.2f}, {coef[5]:.2f},
                {coef[6]:.2f})"
144
145                                         predicted_values.append(intercept + coef[0] * days +
146                                         coef[1]*np.cos(days*1*2*np.pi/365.25) + coef[2]*np.sin
                                            (days*1*2*np.pi/365.25) +
147                                         coef[3]*np.cos(days*2*2*np.pi/365.25) + coef[4]*np.sin
                                            (days*2*2*np.pi/365.25) +
148                                         coef[5]*np.cos(days*3*2*np.pi/365.25) + coef[6]*np.sin
                                            (days*3*2*np.pi/365.25))
149
150         ndvi_magnitudes = [predicted_values[num][-1] - predicted_values[num + 1][0] for num in
            range(len(predicted_values) - 1)]

151
152         # Se não houver mais segmentos a seguir adiciona NODATA_VALUE se só existir um segmento adiciona
            0
153         ndvi_magnitudes.append(65535 if ndvi_magnitudes and any(ndvi_magnitudes) else 0) 154
155         datas = [datetime.fromordinal(data) for data in break_dates]
156         break_dates_epoch = [int(data.replace(tzinfo=timezone.utc).timestamp() * 1000) for data in datas]
157
158         datas = [datetime.fromordinal(data) for data in start_dates]
159         start_dates_epoch = [int(data.replace(tzinfo=timezone.utc).timestamp() * 1000) for data in datas]
160
161         datas = [datetime.fromordinal(data) for data in end_dates]
162         end_dates_epoch = [int(data.replace(tzinfo=timezone.utc).timestamp() * 1000) for data in datas]
163
164     ponto_desejado_wgs = convertPointToCrs(ponto_desejado, 32629, 4326) 165

166     ponto_desejado_wgs_x, ponto_desejado_wgs_y = ponto_desejado_wgs 167

168             dados = [
169             {'tBreak': break_dates_epoch,'tEnd': end_dates_epoch,'tStart':
                start_dates_epoch,'changeProb':prob, 'Lat': ponto_desejado_wgs_y,'Lon':
                ponto_desejado_wgs_x, 'ndvi_magnitude' : ndvi_magnitudes}

170             ]
```

```
171
172                 df = pd.DataFrame(dados)
173
174             # Reorganizar colunas
175             ordem_colunas = ['tBreak', 'tEnd', 'tStart', 'changeProb', 'Lat', 'Lon', 'ndvi_magnitude']
176             df=df[ordem_colunas]
177
178     # Se plot_flag = True faz gráficos 179     if
plot_flag:
180                 # BANDA QUE QUEREMOS PLOTAR NO GRÁFICO
181                 variavel_grafico = ndvis
182
183                 mask = np.array(results['processing_mask'], dtype='bool')
184                 date_objects1 = [datetime.fromordinal(int(ordinal)) for ordinal in dates] 185
186                 plt.style.use('ggplot')
187                 fg = plt.figure(figsize=(14, 4), dpi=90)
188
189     limite_inicial = datetime.strptime('2018-01-01', '%Y-%m-%d') 190         limite_final =
datetime.strptime('2021-12-31', '%Y-%m-%d') 191
192                 a1 = fg.add_subplot(1, 1, 1, xlim=(limite_inicial, limite_final))
193                 plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%Y')) 194
                        plt.gca().xaxis.set_major_locator(mdates.DayLocator()) 195   196
                        a1.xaxis.set_major_locator(mdates.YearLocator(1))
197     a1.xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%Y')) 198
199                         colors = ['orange', 'purple', 'brown']
200
201                         # Predicted curves
202                         for idx, (_preddate, _predvalue, _coef) in enumerate(zip(prediction_dates, predicted_values,
                        coeficientes)):
203                         # Converter números ordinais de volta para objetos de data
204                         _preddate = [datetime.fromordinal(int(ordinal)) for ordinal in _preddate]
205                         color = colors[idx % len(colors)]
206                         coef_str = f"({', '.join([f'{c:.2f}' for c in _coef])})"
207                         label = f'Predicted values {idx + 1} (Coefs: {coef_str})'
208                         a1.plot(_preddate, _predvalue, color, linewidth=1, label=label)
209
210                 a1.plot(np.array(date_objects1)[mask], np.array(variavel_grafico)[mask], 'g+', label='Observed
                values') # Observed values
211                 a1.plot(np.array(date_objects1)[~mask], np.array(variavel_grafico)[~mask],
                'g+') # Observed values masked out 212
213     ticks = [min(date_objects1) + timedelta(days=i*365) for i in range(10) if min( date_objects1) +
timedelta(days=i*365) <= datetime(2021, 12, 31)] 214             plt.xticks(ticks)
215                 plt.title('Lat:' + str(round(ponto_desejado_wgs_x, 5)) + ' Lon:' + str(round(
                ponto_desejado_wgs_y, 5)))
216
217                 a1.plot([], [], color='r', linestyle='--', label='Start dates')
```

```
                a1.plot([], [], color='brown', linestyle='--', label='End Dates')
                a1.plot([], [], color='b', linestyle='--', label='Break dates')
                # a1.plot([], [], color='black', linestyle='--', label='DGT Dates') 221
                    for b in break_dates:
                    b_date = datetime.fromordinal(b)
                    a1.axvline(b_date, color='b', linestyle='--')
                    a1.text(mdates.date2num(b_date)+1, a1.get_ylim()[1], b_date.strftime( '%d-%m-%Y'),
                    rotation=90, ha='right',weight='bold', va='top', color='b', size=8)

    # Linhas verticais para datas de início (color='r') 228      for s in start_dates:
                    s_date = datetime.fromordinal(s)
                    a1.axvline(s_date, color='r', linestyle='--')
                                a1.text(mdates.date2num(s_date) + 1, a1.get_ylim()[0], s_date.strftime(
                                    '%d-%m-%Y'), rotation=90, ha='right',weight='bold', va='bottom', color='r'
                    ,size=8)


    for e in end_dates:
    e_date = datetime.fromordinal(e)
    a1.axvline(e_date, color='brown', linestyle='--')
    a1.text(mdates.date2num(e_date) + 1, a1.get_ylim()[0], e_date.strftime( '%d-%m-%Y'), rotation=90,
ha='right',weight='bold', va='bottom', color=
                    'brown',size=8,alpha=0.6)


                reference_start_date = datetime.strptime('2018-09-12', '%Y-%m-%d')
                reference_end_date = datetime.strptime('2021-09-30', '%Y-%m-%d')
                a1.axvspan(reference_start_date, reference_end_date, facecolor='pink', alpha=
                0.3,label='Período de Referência')

                    plt.ylabel('NDVI')

                plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.1), fancybox=True, shadow=True, ncol=3)
                    plt.tight_layout()
                caminho_graficos=os.path.join(FOLDER_OUTPUTS / 'plots' / f'{img_collection
                }_ccdc_ponto_{i}_{start_dates[0]}_{end_dates[-1]}.png')
                plt.savefig(caminho_graficos)
                   plt.close()

            return df
```

4.3. read_files.py

```
========== 'read_files.py' ==========

    from pyproj import Transformer
    import os
    import re
```

```
    from datetime import datetime
#%%
        def get_most_recent_file(directory, exclude_string=None):
        try:
                                # Get a list of all files in the directory
                files = [f for f in os.listdir(directory) if os.path.isfile(os.path.join( directory, f))]

                            # If there are no files, return None
                        if not files:
                        return None

    # Filter files based on the exclude_string                 if exclude_string:
                                        files = [f for f in files if exclude_string not in f]

    # Get the full path for each file and its corresponding modification time        file_times = [(os.path.join(directory, file), os.path.getmtime(os.path.join( directory, file))) for file in files]

                                # Find the file with the maximum modification time
    most_recent_file = max(file_times, key=lambda x: x[1])
                        return most_recent_file[0]

                except Exception as e:
                print(f"An error occurred: {e}")
                return None
#%%
        def convertPointToCrs(point, source_crs, target_crs):
        """
        Converts a point from a source crs to a target crs.

                Args:
                point: point (shapely.geometry.poin.Point) as extracted from a gdf.
                source_crs: original crs of the input point. Use int (e.g. 4326) or string (e.g. 'EPSG:4326')
                target_crs: new crs the the point should bear. Use int (e.g. 32629) or string
                (e.g. 'EPSG:32629')
                Returns:
                point with new crs
                """
                transformer = Transformer.from_crs(source_crs, target_crs, always_xy=True)
                    #create a transformer for the conversion
            x, y = point

                # transform coordinates to new crs
                new_x, new_y = transformer.transform(x, y)

        return new_x, new_y
        #%%
        def read_tif_files_theia(S2_tile,tiles):
        # DGT
        DGT=False
    # outro         # Theia_T29TNE_20171007-112058

                            list_files=[]
                            for i in range(2017, 2022):
                            if DGT:
                            if i == 2017:
                            base_folder = fr"\\192.168.10.35\\Imag_sentinel2\\Theia_S2process\\" +
```

```
                                S2_tile
64                              else:
65                              base_folder = fr"\\192.168.10.35\\Imag_sentinel2\\Theia_S2process_" + str(i + 1) + "\\" + S2_tile
66                              tiff_pattern = fr"{base_folder}\\S2*.tif"
67                              else:
68                              base_folder=tiles
69                                 #print('base_folder',base_folder)
70                                         tiff_pattern=re.compile('^Theia_T29TNE_' + re.escape(str(i)) + '.*tif$')
71
72                          tiff_files1=[]
73                          for root, dirs, files in os.walk(base_folder):
74                          for file in files:
75                          if tiff_pattern.match(file): 76   tiff_files1.append(file)
77
78                           # Ordena os arquivos pela data
79                      tiff_files = sorted(tiff_files1)
80                      list_files.extend(tiff_files)
81
82
83                      if DGT:
84                      dates = []
85                      date_pattern = re.compile(r"S2A_L2A_(\d{8})-\d{6}_"+S2_tile+".tif") 86   date_pattern2 =
                        re.compile(r"S2B_L2A_(\d{8})-\d{6}_"+S2_tile+".tif") 87             for tiff_file in tiff_files:
88      match = date_pattern.search(tiff_file) 89      match1 =
date_pattern2.search(tiff_file)
90                          if match:
91                          date = match.group(1) 92              dates.append(date) 93       if
                            match1:
94                                  date = match1.group(1)
95                                  dates.append(date) 96          else:
97                      L=len('Theia_T29TNE_')
98                      dates= [x[L:(L+8)] for x in list_files]
99
100     date_objects = [datetime.strptime(date, '%Y%m%d').date() for date in dates] 101          return list_files,
date_objects
102  #%%
103     def read_tif_files_gee(S2_tile,tiles):
104     list_files=[] 105              DGT=False 106              if DGT:
107                             for i in range(2017, 2022):
108                             if i == 2017:
109                             base_folder = fr"\\192.168.10.35\\Imag_sentinel2\\Theia_S2process\\" +
                                S2_tile
110                             else:
111                             base_folder = fr"\\192.168.10.35\\Imag_sentinel2\\Theia_S2process_" + str(i + 1) + "\\" + S2_tile
                                112          tiff_pattern = fr"{base_folder}\\S2*.tif" 113              else:
114                         base_folder=tiles
115                             #print('base_folder',base_folder)
116     tiff_pattern=re.compile('^S2SR_image_.*tif$') 117
118                     tiff_files1=[]
119                     for root, dirs, files in os.walk(base_folder):
120                     for file in files:
121                     if tiff_pattern.match(file): 122   tiff_files1.append(file)
123
124                      # Ordena os arquivos pela data
125             tiff_files = sorted(tiff_files1) #, key=extract_date)
126             list_files.extend(tiff_files)
```

```
127
128                 if DGT:
129                 dates = []
130                 date_pattern = re.compile(r"S2A_L2A_(\d{8})-\d{6}_"+S2_tile+".tif") 131             date_pattern2 =
                    re.compile(r"S2B_L2A_(\d{8})-\d{6}_"+S2_tile+".tif") 132          for tiff_file in tiff_files:
133     match = date_pattern.search(tiff_file) 134    match1 =
date_pattern2.search(tiff_file)
135                             if match:
136                             date = match.group(1)
137                             dates.append(date) 138        if match1:
139                             date = match1.group(1)
140                             dates.append(date) 141        else:
142                 L=len('S2SR_image_')
143                 dates= [x[L:(L+13)] for x in list_files]
144
145         date_objects = [datetime.utcfromtimestamp(int(date)/1000).date() for date in dates ]
146         return list_files, date_objects
147 #%%
148     def readPoints(caminho_arquivo, n_samples=None, random_state_value=42):
149     dados_geoespaciais_metros = caminho_arquivo # seria melhor ler csv; apenas coordenadas interessam 150
                if n_samples:
151                 dados_geoespaciais_metros = dados_geoespaciais_metros.sample(n_samples,
                    random_state=random_state_value).copy()
152
153                  return dados_geoespaciais_metros
```

4.4. Teste_xarray_ccd.py

```
1  ============ 'teste_xarray_ccd.py' ============
2
3      import os
4      user_profile = os.environ['USERPROFILE']
5      import logging
6
7      directory_path = os.path.join(user_profile, 'Desktop', 'CCD_yml_win')
8      os.chdir(directory_path)
9      import geopandas as gpd
10     import pandas as pd
11     import os
12     import sys
13     from pathlib import Path
14     # chamar python a partir da pasta 'CCD'
15     module_path= Path(__name__ ).parent.absolute() / 'S2CHANGE' / 'scripts' /
       'pyccd_theia' #  / 'CCD' / 'S2CHANGE' / 'scripts' /
16     base_path= Path(__name__ ).parent.absolute() # dir do script; # dir referência (acima): 'DGT-S2CHANGE_2023' 17   if
       module_path not in sys.path:
18             sys.path.append(str(module_path))
19             import ccd
20             from notebooks.avaliacao_exatidao_pyccd import filterDate, spatialJoin, preprocessCsvS2, valPol
21             from notebooks.read_files import read_tif_files_theia, read_tif_files_gee, get_most_recent_file, readPoints
22             from notebooks.processing import getTimeSeriesForPoints, runDetectionForPoint, processar_centros_pixeis
23             from notebooks.utils import fromParamsReturnName
24             from tqdm import tqdm
25             from concurrent.futures import ProcessPoolExecutor
```

```
        import warnings
        warnings.filterwarnings('ignore')
        import time
        #%%
        # Início da medição do tempo
        start_time = time.time()
        #%%
        public_documents = Path('C:/Users/Public/Documents/') 34
    samples = public_documents / 'inputs_pontos'
    pontos_input = 'pontos_300_buffers_1_metros.gpkg'
    caminho_arquivo = samples / pontos_input
38
 FOLDER_THEIA = public_documents / 'imagens_Theia' # Caminho dados THEIA 40
FOLDER_GEE = public_documents / 'imagens_GEE' # Caminho dados GEE 41
 FOLDER_BDR = public_documents / 'BDR_300_artigo' / 'BDR_CCDC_TNE_Adjusted.shp' # Caminho para a base de
    dados de validação 43
 FOLDER_OUTPUTS = public_documents / 'output_BDR300'
 S2_tile = 'T29TNE'
 var = 'THEIA' # choose variable: THEIA or GEE 47
        if var == 'THEIA':
        tiles = FOLDER_THEIA / S2_tile
        else:
        tiles = FOLDER_GEE / S2_tile

 img_collection = tiles.parts[-2]

 N=10000

 random_state_value = 42

 bandas_desejadas = [1, 2, 3, 7, 9, 10]
60
 alpha = ccd.parameters.defaults['ALPHA'] # Looks for alpha in the parameters.py file 62   ccd_params =
ccd.parameters.defaults 63
 NODATA_VALUE= 65535
65
    # Parametros da validacao
    # datas do filtro das datas da análise (DGT 300)
    ########### Não alterar ################
    dt_ini = '2018-09-12' # data inicial
 dt_end = '2021-09-30' # data final
    # Margem de tolerância entre a quebra do Modelo e do Analista
    theta = 60 # +/- theta dias de diferenca 73   # bandar a filtrar com base na magnitude
        bandFilter = None #não implementado ainda - não mexer
        #%%
        def main():
        #abre geopackage com pontos
        # print('A abrir o geopackage com pontos...')
        raster_path = tiles / 'Theia_T29TNE_20170813-112433.tif' 80   81          print('Processar centros dos pontos de
        cada geometria para corresponder aos centros dos pixeis dos rasters...') 82
83              start_time = time.time()
84
    gdf_centros_pixeis = processar_centros_pixeis(FOLDER_BDR, raster_path) 86
        # Fim da execução do código
```

```
        end_time = time.time()

        # Calcula o tempo decorrido em segundos
        execution_time_seconds = end_time - start_time

        # Converte o tempo decorrido para minutos
        execution_time_minutes = execution_time_seconds / 60
    print("Processar centros dos pixeis:", execution_time_minutes, "minutos")
dados_geoespaciais_metros = readPoints(gdf_centros_pixeis, N, random_state_value)
                #recolhe nome dos tifs e respetivas datas
    print('A recolher nome e data dos tifs...')
        if var=='THEIA':
        tif_names, tif_dates = read_tif_files_theia(S2_tile,tiles)       else:
    tif_names, tif_dates = read_tif_files_gee(S2_tile,tiles)
        #add full path to tif names
        tif_names = [os.path.join(tiles,i) for i in tif_names]
        #convert dates to ordinal
        tif_dates_ord = [d.toordinal() for d in tif_dates]
            print(f'Processando dados {var}... ({tiles})')
        start_time = time.time()
        #abre tifs com xarray e armazena informacao
                print('A abrir tifs com xarray e carregar série temporal...')
        sel_values, dates, xs, ys = getTimeSeriesForPoints(tif_names, tif_dates_ord, bandas_desejadas,
        dados_geoespaciais_metros)

        # Fim da execução do código
        end_time = time.time()

        # Calcula o tempo decorrido em segundos
        execution_time_seconds = end_time - start_time

        # Converte o tempo decorrido para minutos
        execution_time_minutes = execution_time_seconds / 60
    print(f'Ler dados {var}:", execution_time_minutes, "minutos")
                #executa o ccd em paralelo por ponto
            print('A executar o ccd nos pontos...')
        dfs = []
        with ProcessPoolExecutor(max_workers=os.cpu_count()) as executor:
        tqdm_bar = tqdm(total=sel_values.shape[2])
    arg_list = [(i,sel_values, dates, xs, ys, NODATA_VALUE, FOLDER_OUTPUTS, img_collection) for i in
range(sel_values.shape[2])]

                    for result_df in executor.map(runDetectionForPoint, arg_list):
                    dfs.append(result_df)
                    tqdm_bar.update(1)   tqdm_bar.close()       if dfs:
                df_final = pd.concat(dfs, ignore_index=True)
                # df_final.to_csv('teste_csv_parallel.csv', index=False)
                filename = fromParamsReturnName(img_collection, ccd_params, (S2_tile,tiles), N , random_state_value)
                df_final.to_csv(FOLDER_OUTPUTS / 'tabular' / '{}.csv'.format(filename), index= False)
                #%%
```

```
148                 def runValidation():
149                     print('A correr validação dos resultados do ccd...')
150     filename = fromParamsReturnName(img_collection, ccd_params, (S2_tile,tiles), N, random_state_value) 151
152             csv_s2 = pd.read_csv(FOLDER_OUTPUTS / 'tabular' / '{}.csv'.format(filename))
153             #correr pre-processamento
154             csv_s2 = preprocessCsvS2(csv_s2)
155             csv_preprocessed_path = '{}_pre_proc.csv'.format(filename)
156             csv_s2.to_csv(csv_preprocessed_path)
159             """## Filtrar datas
160             Limitar análise ao período considerado pelos analistas DGT
161             """
162             #correr filtro de datas
163             ccdcFiltro = filterDate(csv_preprocessed_path, dt_ini, dt_end, bandFilter)
164             """## Spatial join
165             Faz join dos pontos do csv com a informação de referencia da DGT (300 buffers). É associada aos pontos a
                informação da validação - data de alteração, tipo, classes, etc.
166             """
167             gdfVal = gpd.read_file(FOLDER_BDR)
168             gdfVal.to_crs(crs = 'EPSG:3763', inplace = True)
169             #executa o join
170             ccdcVal, ccdcVal_T = spatialJoin(FOLDER_BDR, ccdcFiltro)
171             """## Validação
172             Faz a validação da deteção - compara resultado do modelo (ccd) com dados de referência DGT
173             """
174             #faz a validação da deteção
175             DF_FINAL, DF_FINAL_T = valPol(ccdcVal_T, theta) #funcoes.valPol
176             """**Resultados da validação**"""
177             #delimita análise apenas para pontos referentes a transições entre Pinheiro Bravo e Eucalipto para Superfície sem
                vegetação, herbáceas e matos
178             #elimina também pontos da bordadura
179             df_aux = DF_FINAL_T.copy()
180             df_aux = df_aux.loc[(df_aux.altera=="Sem Alteracao")|((df_aux.altera=="Com
                Alteracao")&(df_aux.classeAnterior.isin(['Pinheiro bravo','Eucalipto']))&(df_aux.
                classeAtual.isin(['Superficie sem vegetacao escura','Superficie sem vegetacao
                clara','Vegetacao herbacea espontanea','Matos'])))]
181             df_aux = df_aux.loc[df_aux.bordadura==0]
182             #imprime f1-score, erro e omissão e erro de comissão
183             cm = df_aux.FP.sum()/(df_aux.FP.sum()+df_aux.VP.sum())
184             om = df_aux.FN.sum()/(df_aux.FN.sum()+df_aux.VP.sum())
185             f1 = 2*(1-om)*(1-cm)/(2-om-cm)
186             print(f'Alpha: {alpha}') 187                 print('F1-score = {}%'.format(round(100*f1,2))) 188             print('Omission
                error = {}%'.format(round(100*om,2))) 189     print('Commission error = {}%'.format(round(100*cm,2))) 190
191                                 DF_FINAL_T.to_csv(FOLDER_OUTPUTS / 'tabular' / f'VAL_{filename}.csv', index=False)


193             if __name__ == '__main__':
194             main()
195             runValidation()
```

4.5. utils.py

```
1  ============ 'utils.py' ============
2
```

```
from notebooks.read_files import read_tif_files_theia, read_tif_files_gee

def fromParamsReturnName(col_name, ccd_params, tifs_info, n_sample, random_state_value ):
    """
    Returns file name based on execution parameters:
    col_name: image collection name (Theia, GEE).
    ccd_params: parameters found on ccd parameters.py file.
    tifs_info: path and tile name of tifs in image collection (in the for of (S2_tiles, tiles)).
    n_sample: number of samples used when sampling input points.
    random_state_value: seed used for sampling input points.

    NOTE: currently not implemented to return names of bands used for change detection and tmask.
    """

    #get chi
    chi = str(ccd_params['CHISQUAREPROB'])
        chi = chi[chi.find('.')+1:]
    #get minYears
    minYears = str(ccd_params['MIN_YEARS']).replace('.','')
    #get num obs
    n_obs = str(ccd_params['PEEK_SIZE'])
    #get lambda
        lam = str(ccd_params['ALPHA'])
    #get max iter
    maxIter = str(ccd_params['LASSO_MAX_ITER'])
    #get detection bands
    ### TODO (because using ndvi is currently a temporary solution - replacing blue)
    #get tmask bands
    ### TODO

        #get start and end dates
        S2_tile, tiles = tifs_info

            # Extract the dataset type from col_name
            suffix = col_name.split('_')[-1]

    # Load dataset and retrieve dates
    if suffix == 'Theia':
        _ , dates = read_tif_files_theia(S2_tile,tiles)
    else:
            _ , dates = read_tif_files_gee(S2_tile,tiles)

    start_date = min(dates).strftime("%Y%m%d")
    end_date = max(dates).strftime("%Y%m%d")

        name = "{0}-NDVI_XX{1}YM{2}NOBS{3}LDA{4}ITER{5}_START{6}_END{7}_N{8}_RS{9}".format(col_name, chi, minYears, n_obs, lam, maxIter, start_date, end_date, n_sample, random_state_value)

        return name
```