O código utilizado no trabalho foi criado na linguagem `Python3` utilizando as bibliotecas `Numpy`, `Argparse` e `os`. O Algoritmo é apresentado a seguir:

```python
from argparse import ArgumentParser, FileType
import os
import numpy as np


def get_args():
    parser = ArgumentParser(description='Program to calculate \
                                         the period  of a star \
                                         using the Conditional \
                                         Entropy method')

    parser.add_argument('-i', type=str, default=None,
                        help='Location or file of stellar data')

    parser.add_argument('-o', type=str, default=None,
                        help='Location for output files')

    parser.add_argument('-minp', '--min-period', type=float,
                        default= 0.1,
                        help='Minimun Period to search'
                             '(default = 0.1)')

    parser.add_argument('-maxp', '--max-period', type=float,
                        default= 32.0,
                        help='Maximum Period to search'
                             '(default = 32.0)')

    parser.add_argument('-precision', type=float,
                        default=0.0001,
                        help='Step between periods'
                             '(default = 0.0001)')

    parser.add_argument('-p_bins', '--phase-bins', type=int,
                        default=10,
                        help='Quantity of phase bins'
                             '(default = 10)')

    parser.add_argument('-m_bins', '--mag-bins', type=int,
                        default=5,
                        help='Quantity of magnitude bins'
                             '(default = 5)')
```

```
    args = parser.parse_args()

    return args


def get_files(data):
    '''
    Get the correct path of file or files inside a directory
    '''
    if os.path.isfile(data):
        return [data]

    elif os.path.isdir(os.path.abspath(data)):
        path = os.path.abspath(data)
        os.chdir(path)
        list_of_files = os.listdir(path)
        return sorted(list_of_files)


def out_dir(out):
    '''
    check if the output path is relative or not
    and return the absolute path
    '''
    if out == None:
        return os.getcwd()
    elif os.path.isdir(out):
        return os.path.abspath(out)


def rephase(data, period, col=0, copy=True):
    '''
    transform the time of observations to the phase space
    '''
    rephased = np.ma.array(data, copy=copy)
    rephased[:, col] = get_phase(rephased[:, col], period)
    return rephased


def get_phase(time, period):
    '''
    divide the time of observations by the period
    '''
    return (time / period) % 1
```

```python
def normalization(data):
    '''
    Normalize the magnitude of the star
    '''
    norm = np.ma.copy(data)
    norm[:,1] = (norm[:,1] - np.min(norm[:,1])) \
        / (np.max(norm[:,1]) - np.min(norm[:,1]))

    return norm


def periods_array(min_period, max_period, step):
    '''
    Creates a period array from min_period to max_period with a
    step equals to the third argument
    '''
    period = np.arange(min_period, max_period+step, step)
    return period


def cond_entropy(period, data, p_bins=10, m_bins=5):
    '''
    Compute the conditional entropy for the
    normalized observations
    '''
    if period <= 0:
        return np.PINF
    r = rephase(data, period)
    bins, *_ = np.histogram2d(r[:,0], r[:,1], [p_bins, m_bins],
                              [[0,1], [0,1]])
    size = r.shape[0]
    if size > 0:
        divided_bins = bins / size
        arg_positive = divided_bins > 0
        column_sums = np.sum(divided_bins, axis=1)
        column_sums = np.repeat(np.reshape(column_sums,
                              (p_bins,1)), m_bins, axis=1)

        select_divided_bins = divided_bins[arg_positive]
        select_column_sums  = column_sums[arg_positive]

        A = np.empty((p_bins, m_bins), dtype=float)
        A[ arg_positive] = select_divided_bins \
```

```
                        * np.log(select_column_sums \
                        / select_divided_bins)
        A[~arg_positive] = 0

        return np.sum(A)
    else:
        return np.PINF


def main():
    args = get_args()

    files = get_files(args.i)
    out = out_dir(args.o)
    ce = []
    periods = periods_array(args.min_period, args.max_period,
                            args.precision)

    for star in files:
        ent_data = []
        ce_period = []
        data = np.ma.array(np.loadtxt(star), mask=None,
                           dtype=float)
        norm = normalization(data)

        for p in periods:
            ent_data.append(cond_entropy(p, norm,
                                         args.phase_bins,
                                         args.mag_bins))
            ce_period.append(p)

        np.savetxt(os.path.join(out,
                 'entropies_'+os.path.basename(star)+'.txt'),
                 np.dstack((ce_period, ent_data))[0],
                 fmt='%s')
        right_period = star, ce_period[np.argmin(ent_data)]
        ce.append(right_period)
        np.savetxt(os.path.join(out, 'results.dat'),
                   ce, fmt='%s')

if __name__ == "__main__":
    exit(main())
```

Algoritmo 1: Esquema básico do algoritmo em português estruturado

Entrada: Tempo e Magnitude
Saída: Período P que minimiza a entropia

Início
  Leitura dos dados de entrada como vetores;
  Cria um vetor com n períodos sendo P = $(p_1\ , p_2, \cdots, p_n)$;
  Normalização da magnitude;
  para cada p_i *em* P **faça**
    Transformar o tempo para o espaço de fase;
    Faz as repartições e contabiliza os pontos;
    Calcula a entropia de Shannon condicional;
    Armazena a entropia calculada para o período p_i
  fim
  Achar o valor mínimo de entropia: E_{min} = min(Entropia)
  Achar o período que minimiza a entropia: $P_{E_{min}}$=P[min(entropia)]
Fim
retorna $P_{E_{min}}$