**Heart Rate Processor in Python (HRPY)**

**Figure 1.** *HRPY program mascot. Harpy Eagle free clip art image from Designlooter.com.*

*Graphic Illustration By Analia Treviño-Flitto*

Table of Contents

**Uses3**

**Capabilities 3**

**Libraries and Operating Systems 3**

**Directions4**

**Troubleshooting6**

**Process Flow7**

\*\* *For additional information not please see the Technical Report at* [*https://github.com/mars-group-2021/test\_code*](https://github.com/mars-group-2021/test_code)

**Uses**

This program was developed specifically for Nationwide Children's Hospital (NCH) by Maura Franz as part of the UMGC Master of Science in Biotechnology, Bioinformatics Specialization Capstone in Spring 2021. It was built to convert ECG output files (.csv and .hdr) with the respective data to MIT format (.hea and .dat). Find the full code at <https://github.com/mars-group-2021/mars-hrpy>

**Capabilities**

1. Fills in missing data in ECG files
2. Extracts information from the .csv and .hdr files
3. Converts the data found in the .csv file to MIT format
4. Outputs:
   1. Modified .csv file with gaps filled in
      1. The program fills in gaps by using the average of the last known value and the next known value
   2. Header (.hea) and binary data file (.dat)
5. The output files should be readable by the DOS mystery converter

**Libraries and operating systems**

**Table 1**

*Python libraries incorporated into HRPY.*

|  |  |  |  |
| --- | --- | --- | --- |
| Library | Version | Content | Website |
| Pandas | 1.2.1 | Data analysis and manipulation tool | https://pandas.pydata.org/ |
| wfdb | 3.3.0 | Interface between WFDB library and Python | https://physionet.org/content/wfdb-python/3.3.0/ |

**Table 2**

*Versions of Python and Operating Systems Used.*

|  |  |
| --- | --- |
| Python Version | Operating System / Platform |
| 3.9.1 | macOS v11.2.3 Big Sur |
| 3.9.2 | macOS v11.2.3 Big Sur |
| 3.9.2 | Windows 10 Home 10.0.18363 Build 18363 |

**Directions**

1. Ensure the WFDB library is installed.
   1. Go to the computer’s command line prompt and type in:
      1. >pip install wfdb
2. In the command prompt, navigate to the folder where the program is located.
3. Run the program by typing in the following:
   1. >python hrpy\_v1.py <hdr\_filename.hdr> <csv\_filename.csv>
      1. NOTE: if the .hdr and .csv files are not located in the same directory as the program, the file path will need to be included.
   2. There is no need to concatenate the .hdr and .csv files before running the program.
   3. The program will parse out information from the files and does not require further user input.
   4. IDs from the hdr file are used as the signal IDs.
4. The program will deliver the following outputs:
   1. Modified .csv file
      1. Any data gaps will be filled in using the average of the last valid value and next valid value.
      2. This file is for your own information and does not need to be kept
      3. This will be saved in the same directory as the Python program and will be named as follows:
         1. csv\_filename\_new.csv
         2. For example if your original file name was “PT843.csv” the modified file would be “PT843\_new.csv”
   2. Header file (.hea)
      1. This will be saved in the same directory as the Python program and will be named as follows:
         1. csv\_filename\_new.hea
      2. The record line:
         1. Name of the input file, the number of signals, the sampling frequency (number of samples per second), number of samples per signal, the base time and the base date.
      3. Signal specific lines:
         1. The output file name, the storage format of the data, the ADC gain, the baseline ADC units, the units of the signal, the ADC resolution, the ADC zero, initial value, checksum, block size, and description (in our case, it is the column header/ID number).
         2. Refer to the Definitions table (D1) for information on these.
   3. Data file (.dat)
      1. This will be saved in the same directory as the python program and will be named as follows:
         1. csv\_filename\_new.dat
      2. This is binary and will not be viewable

**Troubleshooting**

1. Ensure the WFDB library is installed and up-to-date.
   1. The program requires at least version 3.3.0 to be installed.
      1. >pip install wfbd==3.3.0
2. Consider updating Python to version 3.9.x and use pip3 for package installation
   1. Using pip3 will ensure that all packages will be installed in Python3 (using pip will use the first Python version listed in your system PATH variable).
3. Ensure the .csv file is is the format shown in Figure D4

**Figure 1**

*Appropriate input .csv file format*



*Note*: Date and time are in the upper left cell. All data is contained in columns to the right of the date- time cell.

**Process Flow**

***Current Functions & Variable List ( in Order of Process Flow )***

hdrparse( file\_name )

* Purpose: Extracts information from the .hdr file
* Input: .hdr filename
* Return: Lists for IDs, units and frequency
* Process:
  + Reads each line and splits it into a list using "," as a delineator
  + The first item is the ID. The "id: " is stripped from the string and the resulting ID is placed in the ids list. "ID" is added to each item for readability in the **csv2mit** function.
  + The second item is the label. “label: ” is stripped from the string and the resulting label is placed in the labels list.
  + The third item is the unit. “unit: ” is stripped from the string and the resulting unit is placed in the units list.
  + The fourth item is the sampling period. “period: ” is stripped from the string and the resulting sampling period is placed in the frequency list.
* Variables:
  + list( ids ): stores the IDs as strings.
  + list( labels ): stores the labels.
  + list( units ): stores the units.
  + list( frequency ): stores the sampling period.

namecsv( file\_name )

* Purpose: create new name for the modified .csv file
* Input: original .csv filename
* Return: name for the new .csv file
* Process:
  + Finds the position of the “.” In the original filename and removes all characters from that position to the end of the string
  + Adds “\_new.csv” to the end of the shortened string
* Variables:
  + str( file\_name ): input name of file to be modified
  + int( position ): position of the “.” in file\_name
  + int( strip\_pos ): number of characters to be removed from file\_name

findtimedate( file\_name )

* Purpose: extracts the start date and time from the .csv file
* Input: .csv filename
* Return: start time and date
* Process:
  + Finds the line in the .csv that begins with four numbers and stores it as a string ( timeline )
  + The string variable ( timeline ) is split into a list using commas as the split point
  + The first item in the list is the date and time cell
  + This is turned into a list of the date and time as the first item.
  + The date and time are then split into a list where the first item is the date and the second is the time.
    - These are stored as string variables
  + The date has to be rearranged to be read by the **csv2mit** function.
    - The datetime package worked in shell, but not in cmd, so it is rearranged the long way.
    - The date is made into a list with the “-“ as the split point.
    - The parts are rearranged and then joined in the correct order (DD/MM/YYYY) with a “/” as a separator.
* Variables:
  + str( file\_name ): name of csv file
  + file( csv ): opened csv file
  + list( firstline ): list of the first line
  + str( starttimestring ): first part of the firstline list
  + list( starttimeparts ): date and time as first item in the list
  + str( starttime ): date and time
  + list( time\_parts ): list of date and time
  + str( time ): sampling start time
  + str( initialdate ): date extracted from the .csv
  + list( dateparts ): each part of the date is an item in this list
  + str( y ): year from date
  + str ( m ): month from date
  + str( d ): day from date
  + str( s ): “/” character
  + list( dateparts ): the date is rearranged in the proper order (d,m,y)
  + str( ewdate ): correct format of the date for converting to MIT format

repaircolumn( column )

* Purpose: replaces empty cells with average of last value and next value
* Input: column (in the form of a list) from the .csv file (the program uses Pandas dataframe to read and store these)
* Return: repaired column
* Process:
  + Position and currentvalue are initialized to 0
  + The program looks through each value of the list.
    - If the value is a whitespace character:
      * The next valid value is found using the **findnextvalue()** function.
      * The average of the last valid value ( currentvalue ) and the next valid value is calculated and appended to the new column.
    - If the value is of type float, it is appended to the new column
    - If the value is neither of those, it is stripped of any whitespace, converted into a float value and appended to the new column.
      * Some of the values in the .csv file are strings with whitespace, but need to be floats in order for the final **csv2mit()** function to work.
* Variables:
  + list( newcolumn ): column with the revised values
  + int( position ): position of the current value being read. This is initialized to 0 and increased at each read.
  + float( currentvalue ): this is initialized to 0. If the value is a whitespace, it is used to find the value to be added to the list, otherwise it is defined as the value being read and is appended to the newcolumn list.
  + float ( nextvalue ): the next valid value as defined by the **findnextvalue()** function.
  + float( newvalue ): the average of currentvalue and nextvalue.

findnextvalue( column, position, currentvalue )

* Purpose: finds the next value in the list that is not a white space.
* Input: list of values, current position in the list, last valid value ( currentvalue )
* Return: next valid value
* Process:
  + If the current position in the list is greater than the length of the column, the value is left as the current value
    - This stops errors from occurring when the last item in the column is a whitespace.
  + Otherwise it cycles through the list to find the next numerical entry, each time checking to make sure it has not exceeded the length of the list.
* Variables:
  + list( column ): the original column of data from the .csv read in as a list
  + int( position ): current position in the column list.
  + float( value ): the value at the current position in the list
  + float( currentvalue ): last valid value in the list (found in **findnextvalue()** function)

samplefrequency( frequency\_list )

* Purpose: find the time between samples
* Input: list of sample periods from the .hdr file.
* Return: time between samples
* Process:
  + The list of sample periods (in ms) is cycled through. The lowest is used for future functions.
* Variables:
  + list( frequency\_list ): list of sample periods from the .hdr file
  + int( sample\_frequency ): the lowest sampling period from the frequency\_list. This is initialized to 2,000,000

time\_correction\_remove( sample\_period, column ):

* Purpose: if the sample period is less than 8 ms, the data columns are passed through this list to remove values not taken at 8 ms intervals
* Input: sample period taken from the .hdr file and a data column from the .csv file
* Return: data column with correct sample frequency
* Process:
  + The correction value ( c\_value ) is found by dividing 8 by the input sample period. The data column is cycled through a while loop where, if the index of each value is divisible by the correction factor, the value is added to a new column.
* Variables:
  + int( sample\_period ): input value. This is parsed out of the .hdr file.
  + list( column ): input value. Data column from the .csv file
  + int( c\_factor ): correction factor. Found by dividing 8 by sample\_period
  + list( new\_column ): stores values that are at the correct time interval
  + int( i ): keeps track of index value

time\_correction\_add( sample\_period, column ):

* Purpose: if the sample period is greater than 8 ms, the data columns are passed through this list to extrapolate values so that data is at 8 ms.
* Input: sample period taken from the .hdr file and a data column from the .csv file
* Return: data column with correct sample frequency
* Process:
  + The correction factor ( c\_factor ) is found by dividing the input sample period by 8.
  + The data column is cycled through two while loops.
    - The length of the column is not exceeded.
    - Number of values to be added is met but not exceeded.
  + The number of values to be added is found by subtracting one from the correction factor.
  + While the number to be added is still greater than 0:
    - The new column ( new\_column ) has values appended to it.
    - The first value from the original column is added first T
      * This is kept as the lower index value ( low\_val ) to be used.
    - The higher index value ( high\_val ) is defined as the second value in the original data column (increased later on)
    - A new value is found by finding the average of the two.
    - The index of the new list is increased.
      * The lower index value ( low\_val ) is now the new value.
    - The index of the original list stays the same until the number of values to be added has been met.
      * This ensures that a graphical representation will show a straight line between the two original values.
    - The new value is added to the new list.
  + Once the number of values to be added has been met between the two samples:
    - The index value of the new list and the first list is increased.
    - The while loop is repeated with the next set of values from the original data set until the index value of the original data set has been reached.
* Variables:
  + int( sample\_period ): input value. Time between samples. Taken from the .hdr file
  + list( column ): input value. Data column from the .csv file
  + list( new\_column ): new column to store the values found in the function
  + int( i ): index value of original data column
  + int( j ): index value of new data column
  + int( y ): number of values to be added
  + float( low\_val ): value with lower index value in original column
  + float( high\_val ): next value up in original data column

float( new\_val ): average of the two neighboring values