**Appendix: Jupyter-based signal processing software**

In the following Appendix, we describe public domain signal processing software operable on any Unix based system (Mac OS, Linux) and a tutorial for streamlining the conversion of TREV impedance measurements into beat-by-beat estimates of contractility. The software, SCOT: Semi-automated Contractility estimates from Ohmic impedance measured with TREV, uses the Jupyter Noteboook and is downloadable at https://github.com/caitgregory/SCOT. Unless otherwise specified, the pipeline uses the Tkinter package to manage all GUI interactions and Matplotlib (Hunter, 2007) to manage plots. It is currently configured to interact with output files from AcqKnowledge (BIOPAC); however, it theoretically could be amended to work with other file formats. Users can fully test or replicate this pipeline by downloading an example data set from the tutorial at <https://github.com/caitgregory/SCOT/blob/main/tutorial.md>. The example data were recorded for 45 minutes during a simultaneous fMRI recording while human participants performed speeded reaches with a joystick. These data were minimally preprocessed during acquisition using AcqKnowledge Software by performing an online lowpass filter (max cutoff = 20 Hz) followed by on-line calculation of *dZ/dt (blood acceleration).* From this we compute the contractility time series offline using the Jupyter software

*Pipeline Processing*

In four largely automated steps, users are able to import the data (Jupyter Notebook, Cell 1), identify beat by beat time intervals (Cells 2 and 3), estimate cardiac contractility at each beat (Cell 4), and remove artifacts related to heart rate and respiratory activity (Cell 5).

Cell 1 of the Jupyter Notebook loads the data via a GUI (Figure A1) using the bioread functions (Vack, 2023). (To replicate the pipeline, users can use the AcqKnowledge file IV\_301\_1.acq.) The resulting menu allows users to specify the appropriate acceleration channel and respiration channels defined during the acquisition. Note that the pipeline imports the stroke acceleration channel (which provides more easily identifiable peaks relative to noise. Here, users can also specify if the acceleration channel or the respiration channel require a FIR low-pass filter. We have preset the cutoff of these filters in the notebook at 22.5 Hz and 0.35 Hz, respectively. The filters use a Hamming window and a length computed by the convention used in freely available packages for processing electrophysiological data (MNE; Gramfort, et al., 2013). Specifically, we construct a filter using the firwin function (SciPy; Virtanen, et al., 2020) with a length of *N*. *N* is computed with 3.3 x 1/*tb*. Here, *tb* is a transition bandwidth which is the minimum value between *f1* and *f2*, where *f1* is the maximum between one quarter of the specified cutoff and 2, and *f2* is the Nyquist frequency minus the specified cutoff. We then apply the filter using the lfilter function (SciPy) and adjust the phase shift by discarding the first *N/2* samples of data and readjusting the time points. The user exits out of the menu which initiates the above steps. Depending on the length of the data this initial import could take a couple of minutes. A notice will appear in the cell output once this step is complete.

[FIGURE A1 HERE]

Cells 2 and 3 identify the time interval between each heartbeat by finding peaks in the acceleration timeseries. As noted above, the acceleration timeseries (*dZ/dt*) is more robust to noise than the contractility timeseries (*d2Z/dt2*), allowing for an easier identification of peaks. The program uses the SciPy findpeaks function which we preset to find peaks spaced at least 0.5 seconds apart (equivalent to a heart rate of 120 BPM). In Cell 2, (Figure A2) users can visually inspect a 20-second portion of the acceleration timeseries at time to identify a minimum threshold for peak amplitude, which they manually input. In addition to the minimum spacing, this peak amplitude threshold acts as an extra automated control against spurious peak identification. On the sample data set, we selected a minimum threshold value of 0.5.

[FIGURE A2 HERE]

Cell 3 (Figure A3) allows users to manually add and remove heart beats via an interactive timeseries of the acceleration waveform. Users can scroll through the data and identify any peaks that the program may have missed or mislabeled. There are three keyboard options that allow the user to edit the predetermined time points of the peaks. Using a two-button mouse, or equivalent keystrokes and clicks for a one-button mouse, a left click will add a peak and a right click will remove a peak. If is there is noise in the signal at any point or the user is unsure where exactly the peak should go, the user can press m + left click. Here, the script performs a moving ensemble average of the two previous and two consecutive peaks to determine the location of the peak of interest.

[FIGURE A3 HERE]

Cell 4 (Figure A4) plots the contractility timeseries (the derivative of acceleration). Note that in cell 3 we found the maximum values of the acceleration timeseries, i.e., the critical values such that *d2Z/dt2* = 0. Given that maximum acceleration is reached after peak contractility, the time points of peaks identified in cells 2 and 3 need to be adjusted backward in time. We accordingly search for maximum contractility amplitude in the time window spanning 250 ms prior to the identified acceleration peak. Users can scroll through the data and if necessary, manually adjust the identified peak amplitude using the same keyboard functions as Cell 3. A left click will add a peak, a right click will remove a peak, and pressing m + left click will provide an estimate of peak amplitude by performing a moving ensemble average of the two previous and two consecutive peaks amplitudes.

[FIGURE A4 HERE]

Lastly, Cell 5 removes the influence of heart rate and respiration from the contractility estimates using the residualizing method described in the methods section above. Briefly, a multiple regression was conducted where contractility is modeled as a function of the heart rate, respiratory amount, and respiratory cycle at each heartbeat. Heart rate is computed from the inter-beat intervals identified in cell 3. Respiratory amount and cycle are identified by first finding each consecutive cosine-like segment in the specified respiration timeseries. *Y*-axis values (i.e., respiration amount) of each segment are demeaned while *x-*axis values (i.e., respiration phase) are normalized between 0 and 2*π*. We then extract the respiratory amount and phase values closest to each heartbeat. Prior to the regression, each regressor is *z*-scored. We output the residuals of the regression model as the contractility estimates with the effects of heart rate and respiration removed. Once completed, data are outputted into a csv file with each row corresponding to a heartbeat, and columns to the time of each heartbeat (relative to the beginning of the recording) and the contractility amplitude. By default, the csv will be named the same as the input AcqKnowledge file with the csv extension; however, users can change this through a GUI.

Figure captions

Figure A1: Cell 1 GUI.

Figure A2: GUI for cell 2. Note the peak threshold is inputted as 0.5. This threshold value helps avoid flutter between acceleration peaks. The participant has a premature ventricular contraction at time 16.5s (causing a reduction of contractility due to reduced ventricular filling). Also note the onset of MRI scanning at 18 seconds. Despite the associated MRI associated noise, acceleration peaks are still visible and robust.

Figure A3: Cell 3 GUI. The acceleration time series plotted over time with detected peaks. The user is able to use the slider along the bottom of the graph to scroll through the data and adjust the peak location as needed.

Figure A4: Cell 4 GUI. The contractility timeseries plotted over time.