# pybreathe: a python package for respiratory airflow rates analysis

Thibaut Coustillet [ORCID] [1,2]

1 Sorbonne Université, CNRS, Inserm, Development, Adaptation and Ageing, Dev2A, F-75005 Paris, France [ROR] **2** Sorbonne Université, CNRS, Inserm, Institut de Biologie Paris-Seine, IBPS, F-75005 Paris, France [ROR]

Summary

Breathing is the vital function that enables air to flow into the lungs during inhalation and out during exhalation. Inhalation delivers (di)oxygen to the tissues, while exhalation flushes out carbon dioxide. In physiology, breathing is widely studied to investigate a whole range of respiratory diseases as well as physical abilities. A consistent and robust analytical framework is essential to cope with the large volume of data and to address the often time-consuming nature of routine analysis procedures. pybreathe is a python package that allows breathing to be formally analysed. It lets users to extract essential features such as the volume inhaled and exhaled, the inspiratory and expiratory times. It also provides the breathing frequency. The package has been designed to be user-friendly, relying solely on a user-supplied discretised air flow rate considered as a time series (instantaneous flow rate).

Statement of need

In 2017, half a billion people worldwide lived with a chronic respiratory disease (Soriano et al., 2020). Concurrently, the role of breathing has garnered growing attention in research focused on both athletic/sport performance (Contreras-Briceño et al., 2024; Harbour et al., 2022) and overall well-being (Fincham et al., 2023). Due to the large amount of respiratory data acquired over the last few years, numerous algorithms have been developed to structure analysis and open up new insights. Such tools complement commercial analysis software used historically (Lusk et al., 2023).

On the one hand, most of the open source software dealing with breathing relies heavily on peak/hollow detection (Bishop et al., 2022; Brammer, 2020; Makowski et al., 2021) to extract signal features such as amplitude and breathing period. However, such detection methods often require human correction, manual curation or advanced algorithms to guarantee the accuracy of the results (Vanegas et al., 2020). Besides, such local extrema detection approaches are more suited for the characterization of instantaneous volume than of instantaneous flow. Although the former can be deduced from the latter (Figure 1), it is generally flow rates, themselves derived from pressure differences that are supplied (Criée et al., 2011). In some cases, however, signals and related algorithms may also originate from chest/abdominal belts (Holm et al., 2024).
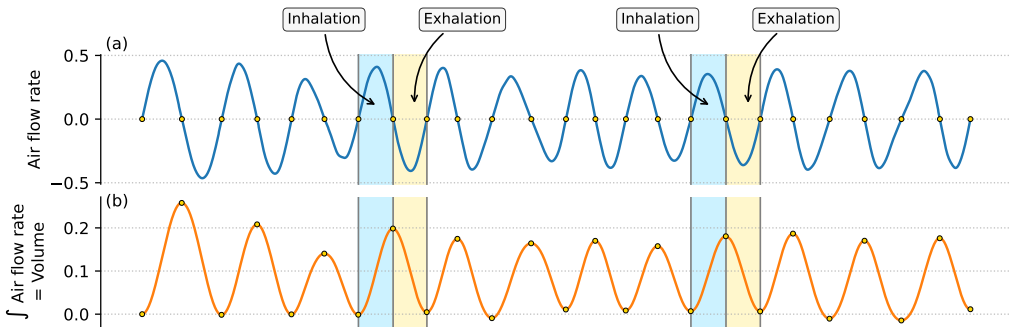
Figure 1: Relationship between instantaneous flow rate (a) and instantaneous volume (b). The volume is obtained by integrating the flow rate over time. Thus, when the flow rate is positive (inhalation; blue areas), the volume increases, whereas when the flow rate is negative (exhalation; yellow areas), the volume decreases.

On the other hand, some advanced algorithms use cutting-edge clustering methods to detect respiratory patterns that go beyond the features mentioned above (Germain et al., 2023). Although this kind of approach is particularly valuable for providing a deeper understanding of respiratory physiology across various experimental conditions, it requires advanced programming skills and knowledge and may be complicated to set up in practice for non-computer users.

In this paper, we sought to implement an easy-to-use framework specially designed to facilitate respiratory analyses derived from instantaneous air flow rates (recorded by plethysmography). pybreathe is a python package with an API designed to be used by both experimenters and developers. Users can acquire their respiratory data using any standard software. A necessary and sufficient condition for using pybreathe is to export the data (instantaneous air flow rate) and discretise it into a classic text format (*e.g.*, .txt, .csv). Most software applications used for measuring respiratory air flow rates include this functionality and can output a two-column file: the first column represents the discretised time vector, while the second provides the corresponding flow values at each time point (Table 1).

Table 1: Example of a two-column table depicting the instantaneous discretised air flow rate required for the use of pybreathe. Instantaneous air flow rate is a time series. User files should have the same configuration. To enable the calculation of volumes in absolute values, the flow rate should also be in absolute values (*e.g.*, mL.s^{-1} or L.s^{-1}). The time vector format should also represent actual timepoints.

time	values
0.0	0.0650
0.004	0.0660
0.008	0.0671
0.012	0.0681
0.016	0.0692
...	...

The main difference with other respiratory analysis algorithms is that pybreathe is based on ventilatory flow and not on volume. To our knowledge, there is no open-source algorithm that simply extracts elementary but essential features from air flow recordings.

The main feature of a respiratory signal is the tidal volume (*i.e.*, the volume passing through the lungs during a single breath). In the case of air flow rates, peak/hollow analysis cannot be applied because the amplitude (*i.e.*, height) depends on the 'speed' at which the air flows in and out: for the same exhaled or inhaled volume, the faster the airflow, the greater the amplitude (Figure 2).
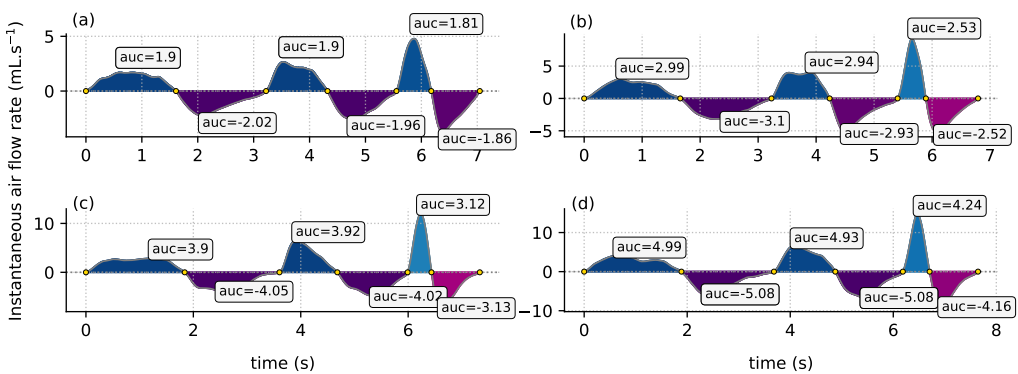
Figure 2: Manual injection/aspiration of different quantities of air into a chamber with a syringe at three different speeds: slow, moderate and fast. (a) 2 mL; (b) 3 mL; (c) 4 mL; (d) 5 mL. Injection corresponds to the positive parts (blue) while aspiration corresponds to the negative parts (purple). AUCs values were obtained with `pybreathe`.

In this situation, to really grasp the tidal volume, we need to get the Area Under the Curve (AUC) instead of the amplitude. We manually injected and aspirated different quantities of air into a chamber at different flow rates (Figure 2) and used `pybreathe` to demonstrate the relevance of calculating AUCs rather than amplitudes (Table 2).

Table 2: Comparison of the integral (Area Under the Curve) and amplitude (height) of several volumes of air manually injected/aspirated into a chamber.

actual volume	speed	positive integral	negative integral	positive amplitude	negative amplitude
≈ 2 mL	slow	1.90	- 2.02	1.70	- 2.26
≈ 2 mL	moderate	1.90	- 1.96	2.66	- 2.54
≈ 2 mL	fast	1.81	- 1.86	4.79	- 3.75
≈ 3 mL	slow	2.99	- 3.10	3.04	- 3.19
≈ 3 mL	moderate	2.94	- 2.93	3.98	- 5.07
≈ 3 mL	fast	2.53	- 2.52	9.24	- 5.2
≈ 4 mL	slow	3.90	- 4.05	2.91	- 3.71
≈ 4 mL	moderate	3.92	- 4.02	6.46	- 4.81
≈ 4 mL	fast	3.12	- 3.13	12.03	- 7.09
≈ 5 mL	slow	4.99	- 5.08	4.21	- 5.76
≈ 5 mL	moderate	4.93	- 5.08	6.58	- 6.70
≈ 5 mL	fast	4.24	- 4.16	14.88	- 9.04

For each quantity of air, the integral faithfully represents the volume injected and aspirated. The amplitude is not representative of the volume injected or aspirated. Interestingly, regardless of the volume of air injected, high injection speeds consistently compromise measurement precision. This issue arises solely because the air is injected manually by an experimenter with a syringe and does not originate from `pybreathe`.

For a given respiratory signal, `pybreathe` detects zero-crossings (Figure 1a) and each positive segment will be either inhalation or exhalation (depending on the configuration of the primary acquisition software) and each negative segment will be the other phase. AUC (integral) of these segments therefore corresponds to the volume inhaled or exhaled. The duration of these

segments (time between two zeros) corresponds to the inspiratory or expiratory time. Breathing frequency is also provided based either on the frequency of peaks or hollows, or using a more sophisticaded spectral analysis.

pybreathe fundamentals

The pybreathe package is accompagnied by a *BreathingFlow* object which is the core of the algorithm. Users should instantiate a *BreathingFlow* object with their own data (*e.g.*, Table 1).

```python
from pybreathe import BreathingFlow

# Loading a discretised respiratory flow rate
my_signal = BreathingFlow.from_file(
  filename="path_to_your_discretised_flow",
  identifier="my data"
)
```

where path_to_your_discretised_flow is a two-column discretised file (*e.g.*, .txt, .csv) representing instantaneous airflow as a function of time (e.g., Table 1).

The package comes with example scripts based on simulated breathing signals that are representative of the data that can be collected experimentally. They explain the milestones involved in carrying out an analysis (feature extraction) and they supply useful documentation. To get started with pybreathe API, users are strongly advised to refer to these scripts using either the sine function (mimicking a respiratory signal) or two artificial breathing signals by instantiating *BreathingFlow* objects using the class methods provided for this purpose:

```python
from pybreathe import BreathingFlow

# Sine function
sine = BreathingFlow.load_sinus()

# Artificial signal #1
example_01 = BreathingFlow.load_breathing_like_signal_01()

# Artificial signal #2
example_02 = BreathingFlow.load_breathing_like_signal_02()
```

Then, the five main methods for extracting features are:

- example_01.get_positive_time() which extracts the average duration of positive segments,
- example_01.get_negative_time() which extracts the average duration of negative segments,
- example_01.get_positive_auc() which extracts the average AUC of positive segments,
- example_01.get_negative_auc() which extracts the average AUC of negative segments,
- example_01.get_frequency(), which extracts the signal frequency.

For optional arguments and other available methods (*e.g.*, visualisation, saving, artefact removal), users should refer to the example scripts.

Proof

To ensure that the package worked correctly and that the methods returned the desired output, we checked the volumes extracted by pybreathe when we injected/aspirated known quantities of air (Figure 2 and Table 2). We observed that the manually tested volumes (2 mL, 3 mL, 4 mL, 5 mL) corresponded well to the integral values (AUCs) for each injection or aspiration.

100 However, because the manual injection/aspiration of air into a chamber can be imprecise due to
101 the experimenter and the precision of the syringe, we also created an artifical respiratory signal
102 corresponding to the sine function (Figure 3). Based on the sine, we checked that the signal
103 features obtained with pybreathe were indeed the same as those obtained *mathematically*.

104 Let f the sinus function.

$$f : \mathbb{R} \longrightarrow \mathbb{R}$$
$$x \longmapsto \sin(x)$$

105 The sin function is 2π-periodic, i.e.,

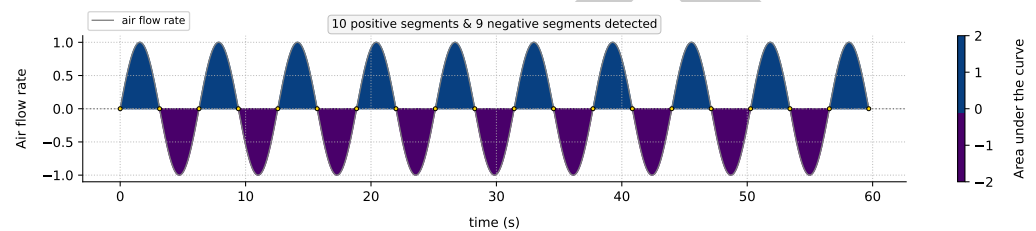$$\forall x \in \mathbb{R}, \quad \sin(x + 2\pi) = \sin(x)$$

Figure 3: Graph of the sine function on the interval $[0, 10\pi]$.

Positive time (~ inspiratory time)

107 Statement: the mean length of the interval where the sine function is positive is exactly equal
108 to π.

Mathematical proof

110 Let $x \in \mathbb{R}$.

111 $\sin(x) \geq 0 \iff x \bmod 2\pi \in [0, \pi]$

112 Thus, the duration (interval length) of all positive segments is $\pi - 0$, which is equal to π.

pybreathe validation

```
>>> from pybreathe import BreathingFlow
>>> sinus = BreathingFlow.load_sinus()
>>> sinus.get_positive_time()
mean = 3.14 ± 9.19e-10 (n = 10).
```

Negative time (~ expiratory time)

115 In the same way, we can demonstrate that the average duration of negative segments is also π,
116 which is also found with pybreathe.

```
>>> from pybreathe import BreathingFlow
>>> sinus = BreathingFlow.load_sinus()
>>> sinus.get_negative_time()
mean = 3.14 ± 7.43e-10 (n = 9).
```

Positive Area Under the Curve (~ inhaled volume)

118 Statement: the mean AUC (integral) of the positive segments of the sine function is exactly 2.

Mathematical proof

Let $x \in \mathbb{R}$.

$$\sin(x) \geq 0 \iff x \bmod 2\pi \in [0, \pi]$$

$$
\begin{aligned}
\int_0^\pi \sin x \; dx &= [-\cos x]_0^\pi \\
&= -\cos(\pi) - (-\cos(0)) \\
&= -\cos(\pi) + \cos(0) \\
&= -(-1) + 1 \\
&= 1 + 1 \\
&= 2
\end{aligned}
$$

Thus, the mean AUC of all positive segments is 2.

pybreathe validation

```
>>> from pybreathe import BreathingFlow
>>> sinus = BreathingFlow.load_sinus()
>>> sinus.get_positive_auc()
mean = 2.0 ± 8.4e-12 (n = 10).
```

Negative Area Under the Curve (~ exhaled volume)

In the same way, we can demonstrate that the average AUC of negative segments is exactly -2, which is also found with pybreathe.

```
>>> from pybreathe import BreathingFlow
>>> sinus = BreathingFlow.load_sinus()
>>> sinus.get_negative_auc()
mean = -2.0 ± 7.68e-12 (n = 9).
```

Acknowledgements

References

Bishop, M., Weinhold, M., Turk, A. Z., Adeck, A., & SheikhBahaei, S. (2022). An open-source tool for automated analysis of breathing behaviors in common marmosets and rodents. *eLife*, *11*, e71647. https://doi.org/10.7554/eLife.71647

Brammer, J. (2020). Biopeaks: A graphical user interface for feature extraction from heart- and breathing biosignals. *Journal of Open Source Software*, *5*(54), 2621. https://doi.org/10.21105/joss.02621

Contreras-Briceño, F., Cancino, J., Espinosa-Ramírez, M., Fernández, G., Johnson, V., & Hurtado, D. E. (2024). Estimation of ventilatory thresholds during exercise using respiratory wearable sensors. *Npj Digital Medicine*, *7*(1). https://doi.org/10.1038/s41746-024-01191-9

Criée, C. P., Sorichter, S., Smith, H. J., Kardos, P., Merget, R., Heise, D., Berdel, D., Köhler, D., Magnussen, H., Marek, W., Mitfessel, H., Rasche, K., Rolke, M., Worth, H., & Jörres, R. A. (2011). Body plethysmography – its principles and clinical use. *Respiratory Medicine*, *105*(7), 959–971. https://doi.org/10.1016/j.rmed.2011.02.006

Fincham, G. W., Strauss, C., Montero-Marin, J., & Cavanagh, K. (2023). Effect of breathwork on stress and mental health: A meta-analysis of randomised-controlled trials. *Scientific Reports*, *13*(1). https://doi.org/10.1038/s41598-022-27247-y

Germain, T., Truong, C., Oudre, L., & Krejci, E. (2023). Unsupervised classification of plethysmography signals with advanced visual representations. *Frontiers in Physiology*, *14*, 1154328. https://doi.org/10.3389/fphys.2023.1154328

Harbour, E., Stöggl, T., Schwameder, H., & Finkenzeller, T. (2022). Breath tools: A synthesis of evidence-based breathing strategies to enhance human running. *Frontiers in Physiology*, *13*. https://doi.org/10.3389/fphys.2022.813243

Holm, B., Borsky, M., Arnardottir, E. S., Serwatko, M., Mallett, J., Islind, A. S., & Óskarsdóttir, M. (2024). BreathFinder: A method for non-invasive isolation of respiratory cycles utilizing the thoracic respiratory inductance plethysmography signal. *Nature and Science of Sleep*, *16*, 1253–1266. https://doi.org/10.2147/NSS.S468431

Lusk, S., Ward, C. S., Chang, A., Twitchell-Heyne, A., Fattig, S., Allen, G., Jankowsky, J. L., & Ray, R. S. (2023). An automated respiratory data pipeline for waveform characteristic analysis. *The Journal of Physiology*, *601*(21), 4767–4806. https://doi.org/10.1113/JP284363

Makowski, D., Pham, T., Lau, Z. J., Brammer, J. C., Lespinasse, F., Pham, H., Schölzel, C., & Chen, S. H. A. (2021). NeuroKit2: A python toolbox for neurophysiological signal processing. *Behavior Research Methods*, *53*(4), 1689–1696. https://doi.org/10.3758/s13428-020-01516-y

Soriano, J. B., Kendrick, P. J., Paulson, K. R., Gupta, V., Abrams, E. M., Adedoyin, R. A., Adhikari, T. B., Advani, S. M., Agrawal, A., Ahmadian, E., Alahdab, F., Aljunid, S. M., Altirkawi, K. A., Alvis-Guzman, N., Anber, N. H., Andrei, C. L., Anjomshoa, M., Ansari, F., Antó, J. M., … Vos, T. (2020). Prevalence and attributable health burden of chronic respiratory diseases, 1990–2017: A systematic analysis for the global burden of disease study 2017. *The Lancet Respiratory Medicine*, *8*(6), 585–596. https://doi.org/10.1016/S2213-2600(20)30105-3

Vanegas, E., Igual, R., & Plaza, I. (2020). Sensing systems for respiration monitoring: A technical systematic review. *Sensors (Basel, Switzerland)*, *20*(18), 5446. https://doi.org/10.3390/s20185446