# PARMESAN: Meteorological Timeseries and Turbulence Analysis Backed by Symbolic Mathematics

Yann Georg Büchau [1][¶], **Hasan Mashni**[1], **Matteo Bramati**[1], **Vasileios Savvakis** [1], **Ines Schäfer**[1], **Saskia Jung**[1], **Gabriela Miranda-Garcia** [1], **Daniel Hardt** [2], **and Jens Bange** [1]

1 Eberhard Karls Universität Tübingen, Germany **2** Akaflieg Braunschweig e.V., Braunschweig, Germany ¶ Corresponding author

Summary

PARMESAN (the **P**ython **A**tmospheric **R**esearch Package for **ME**teorological Time**S**eries and Turbulence **AN**alysis) is a Python package providing common functionality for atmospheric scientists doing time series or turbulence analysis. Several meteorological quantities such as potential temperature, various humidity measures, gas concentrations, wind speed and direction, turbulence and stability parameters can be calculated. Furthermore, signal processing functionality such as properly normed variance spectra for frequency analysis is available. In contrast to existing packages with similar goals, its routines for physical quantities are derived from symbolic mathematical expressions, enabling inspection, automatic rearrangement, reuse and recombination of the underlying equations. Building on this, PARMESAN's functions as well as their comprehensive parameter documentation are mostly auto-generated, minimizing human error and effort. In addition, sensitivity/error propagation analysis is possible as mathematical operations like derivations can be applied to the underlying equations. Physical consistency in terms of units and value domains are transparently ensured for PARMESAN functions. PARMESAN's approach can be reused to simplify implementation of robust routines in other fields of physics.

Statement of need

The need to assert properly balanced physical units right from within running programs and models has been recognised for a long time now (Chizeck et al., 2009; Cooper & McKeever, 2008). Unit conversion errors in science and engineering have caused costly system failures such as the NASA Mars Climate Orbiter crash in 1999 (NASA, 1999).

Nowadays, the Python ecosystem comprises many packages that ease specific tasks when performing physical calculations: numpy (Harris et al., 2020) and scipy (Virtanen et al., 2020) provide efficient numerical routines, pandas (The pandas development team, 2023) and xarray (Hoyer & Hamman, 2017) provide structures to read, write and aggregate data, pint (Grecco & Chéron, 2023) handles physical units and the uncertainties package (Lebigot, 2023) simplifies linear error propagation. Partly based on those, collections of routines for atmospheric science exist such as metpy (May et al., 2022), iris (Met Office, 2010 - 2023) and aoslib/PyAOS (PyAOS, 2023). However these focus more on gridded, spatial data which is common in modelling and remote sensing and have little functionality for turbulence analysis. Turbulence plays an important role in atmospheric exchange processes, especially in the planetary boundary layer (Stull, 1988). It is a statistical process and thus mostly quantified through high-resolution in-situ measurement techniques (Foken, 2021). metpy and iris can both handle units and require the user to explicitly specify them. Their physical quantities are

calculated using hard-coded expressions. In contrast, the `atmos` package (McGibbon, 2023) has implemented an equation solving system for more flexible reusability and less hard-coding of relationships between quantities. Its development seems to have stalled since 2020, though. None of the above packages have a mechanism for transparently checking that input and output values are within reasonable physical bounds.

PARMESAN addresses the aforementioned gaps by providing functions for meteorological quantities that are backed by symbolic mathematical expressions employing SymPy (Meurer et al., 2017), a powerful computer algebra system written purely in Python. Inputs and outputs are checked for and potentially converted to correct units while asserting that the physical domains are not exceeded. It can rearrange its equations and thus flexibly increase the number of available functions. PARMESAN has already been used successfully in Büchau et al. (2022), Büchau et al. (2023), Wüsteney (2023) and Herrmann (2023) for data analysis of meteorological measurements.

Structure

Functions for physical quantities in PARMESAN are based on symbolic mathematical equations created using SymPy (Meurer et al., 2017). PARMESAN defines a descriptive list of symbols (i.e. variables and constants, Figure 1) and relates them to form the common laws of thermodynamics, parametrisations and definitions used in atmospheric science.
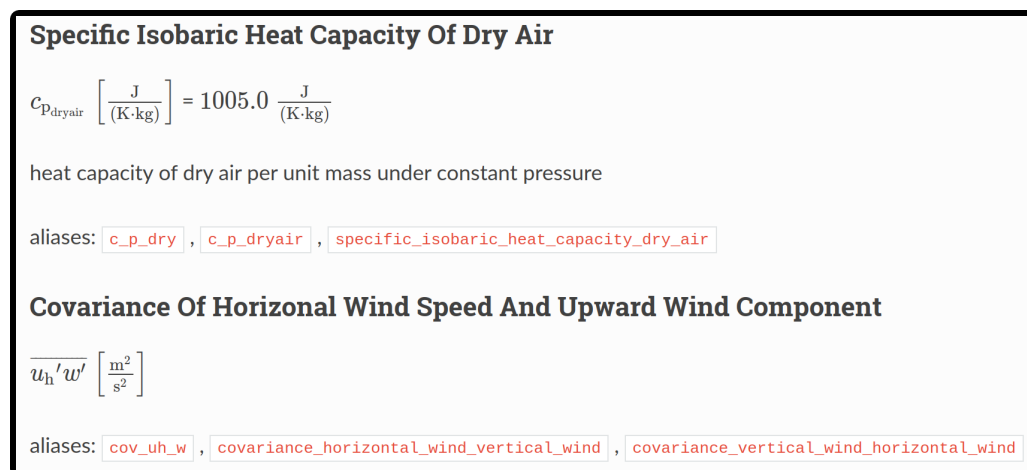

Specific Isobaric Heat Capacity Of Dry Air

$$c_{\mathrm{p_{dryair}}} \left[\frac{\mathrm{J}}{\mathrm{(K \cdot kg)}} \right] = 1005.0 \; \frac{\mathrm{J}}{\mathrm{(K \cdot kg)}}$$

heat capacity of dry air per unit mass under constant pressure

aliases: `c_p_dry` , `c_p_dryair` , `specific_isobaric_heat_capacity_dry_air`

Covariance Of Horizonal Wind Speed And Upward Wind Component

$$\overline{u_{\mathrm{h}}'w'} \left[\frac{\mathrm{m}^2}{\mathrm{s}^2} \right]$$

aliases: `cov_uh_w` , `covariance_horizontal_wind_vertical_wind` , `covariance_vertical_wind_horizontal_wind`

Figure 1: Excerpt of auto-generated symbol list in `parmesan.symbols`. Symbols have metadata such as descriptions, units and default values attached. For readability, they can be referred to with different variable names, which are also available as parameter aliases when calling functions in PARMESAN.

This approach has many advantages over the traditional method of hard-coding mathematical operations between function inputs using language-specific constructs. First of all, information about the mathematical relationship between quantities is not lost, but can instead be queried and reused. SymPy equations can be rearranged and recombined to generate new expressions, enabling the generation of many specific functions from a set of base equations. Additionally, SymPy expressions are translatable into code for numerous programming languages. PARMESAN uses this mechanism to turn its equations into executable Python functions that use the efficient numpy package internally (Harris et al., 2020), so no runtime overhead is introduced and array inputs and outputs are supported. Symbolic expressions are automatically simplified and terms cancelled accordingly, revealing the set of input parameters an equation really depends on. This information is then used to automatically generate extensive documentation for each individual function (Figure 2) - a great benefit for consistency and minimisation of human effort and oversight in the documentation.

Creating new functions in or from PARMESAN thus often requires only very few lines of code. Here is a compacted version of PARMESAN's function for potential temperature specifically for dry air:

```python
from parmesan.symbols import *   # Import all of PARMESAN's symbols
@from_sympy() # decorator turning SymPy expression into code and documentation
def potential_temperature(): # no arguments necessary, added automatically
    return T * (p_ref / p) ** (R_dryair / c_p_dryair)
    # SymPy expression - practically equal to typical Python code
```

In this case, the resulting quantity is derived from the function's name, documentation is generated ([Figure 2]) and the equation is immediately checked for units consistency employing the `pint` package ([Grecco & Chéron, 2023]). Each symbol has metadata attached, such as a physical unit and a domain ([Figure 1]). These are available to the resulting function for assertion, so a PARMESAN function will check and auto-convert input and output units and issue a warning when unphysical values arise such as negative absolute temperatures:

```python
# Implicit Units
potential_temperature(T=300, p=100000) # K and Pa assumed
# 300.0 K

# Explicit Units
from parmesan.units import units # PARMESAN's predefined units
potential_temperature(T=units.Quantity(20,"°C"), p=950 * units.hPa)
# 297.477188635086 K

# Parameter/Symbol Aliases
potential_temperature(temperature=300, pressure=100000)
# 300.0 K

# Arrays
import numpy as np
potential_temperature(T=300, p=np.array([950,980,1010]) * units.hPa)
# Magnitude: [304.42830151978785 301.7364178157801 299.14844787358106]
#     Units: K

# Bounds check
potential_temperature(T=-10, p=1010*units.hPa) # temperature out of bounds
# OutOfBoundsWarning: 1 of 1 input values to potential_temperature for
# argument 'T' are out of bounds defined by 'positive': [-10] at indices [0]
# -9.971614929119369 K

# Units check
potential_temperature(T=300, p=1010*units.degrees) # wrong unit -> error
# DimensionalityError: potential_temperature():
#     p=<Quantity(1010, 'degree')> could not be converted to pascal:
#     Cannot convert from 'degree' (dimensionless) to 'pascal'
#     ([mass] / [length] / [time] ** 2)
```
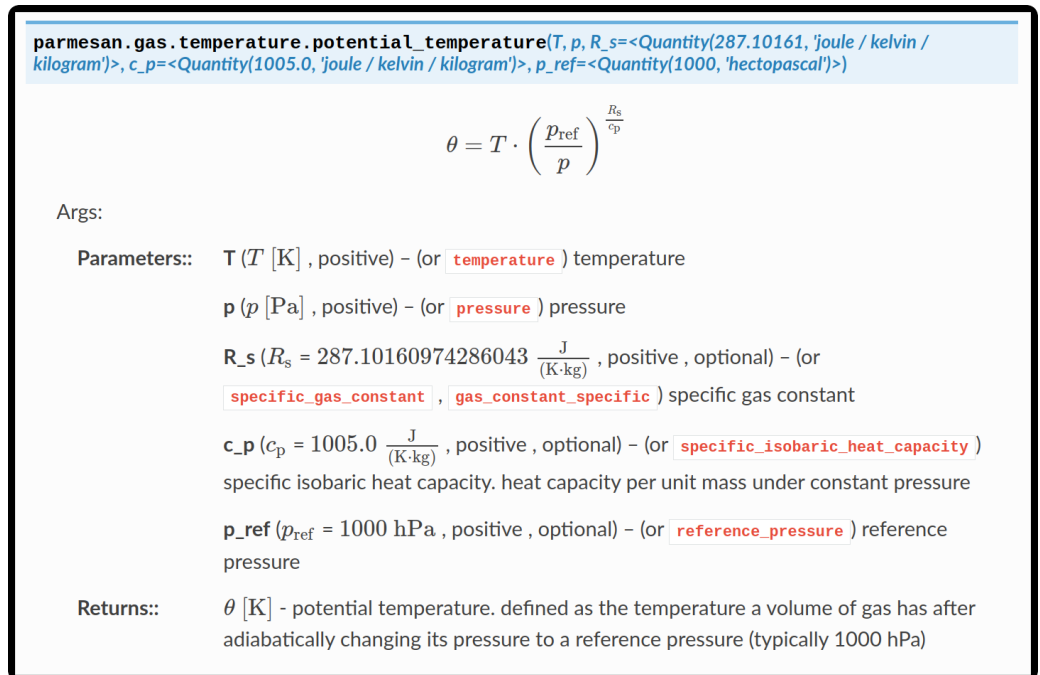
Figure 2: Auto-generated comprehensive parameter documentation and LaTeX-formatted equation for PARMESAN's `potential_temperature()` function to calculate potential temperature from atmospheric pressure and temperature. Parameter aliases, units, defaults and bounds are taken from PARMESAN's symbol library (Figure 1) and used coherently across functions in PARMESAN.

Another benefit of having the underlying symbolic expression for an equation available is the possibility to do sensitivity analysis. PARMESAN can derive the maximum relative error $\Delta y_{\mathrm{max,rel}}$ (Equation 1) for its symbolic functions (Figure 3):

$$\Delta y_{\mathrm{max}}(x_1, \dots, x_{\mathrm{n}}) = \sum_{i=1}^{n} \left| \frac{\partial y}{\partial x_i} \right| \cdot \Delta x_{i_{\mathrm{max}}}$$

$$\Delta y_{\mathrm{max,rel}} = \frac{\Delta y_{\mathrm{max}}}{\overline{y}}$$

(1)

The maximum relative error is a conservative estimation method for the propagation of errors of input quantities x_i to effective error in the output quantity y, assuming the most severe combination of input quantity deviations $\Delta x_{i_{\mathrm{max}}}$. Custom sensitivity analyses can also be implemented based on PARMESAN's equations.

Figure 3: Auto-generated maximum relative error equation (Equation 1) for PARMESAN's `potential_temperature()` function (Figure 2). Symbolic PARMESAN functions automatically have a sensitivity analysis attached to quantify how a change in input parameters affects the output. In this case, the maximum expected relative error of potential temperature [%] is the sum of the maximum relative errors of temperature and pressure [%], with the pressure term scaled by a factor.

PARMESAN can also rearrange its existing equations (Figure 4) for a quantity of interest by its provided `get_function()` function:

```python
from parmesan.symbols import *
# get (or rearrange) functions that calculate mixing ratio
mixing_ratio_functions = list(get_function(result=mixing_ratio)
# get (or rearrange) functions that calculate mixing ratio
# from at least temperature and pressure
mixing_ratio_functions = list(get_function(result=mixing_ratio, inputs=(T, p))
```

The functions found can be called as usual or their underlying equations can be examined by accessing their `.equation` attribute. In a Jupyter notebook (Kluyver et al., 2016) the equations appear as formatted markup similar to what is depicted in Figure 4.

Figure 4: Excerpt of auto-generated humidity equation list in PARMESAN's `humidity` module. As the underlying equations in PARMESAN's functions are available as symbolic expressions, it can provide overviews of all related equations.

Besides physical equations, PARMESAN provides tools often needed when analysing timeseries such as calculating second-order moments, variance spectrum (Figure 5), autocorrelation, structure function (variogram) and running covariance, e.g. for calculating eddy fluxes (Foken, 2021), backed by the `scipy` package (Virtanen et al., 2020) for efficient numerics and `matplotlib` (Hunter, 2007) for visualisation. PARMESAN integrates with the common pandas

data analysis framework ([The pandas development team, 2023](#)) by adding a `.parmesan` accessor to `DataFrame` and `Series` objects to apply PARMESAN functions such as a variance spectrum or autocorrelation directly to them.
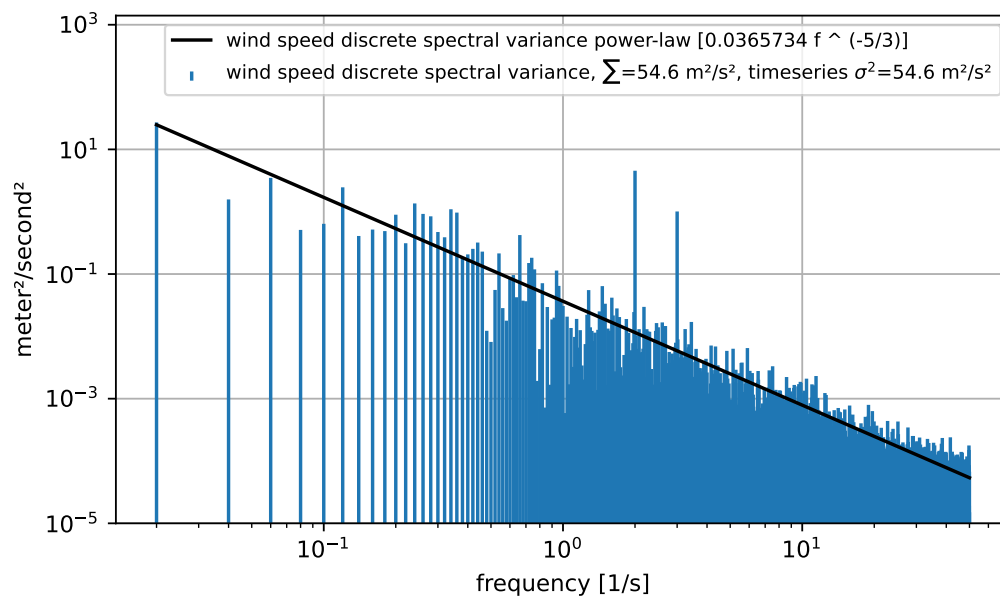

Figure 5: PARMESAN discrete `variance_spectrum()` of an artificial wind timeseries (random walk overlayed with 2Hz and 3Hz sine waves). Note how Parseval's Theorem ([Stull, 1988](#)) is correctly fulfilled as the timeseries variance equals the sum of discrete spectral variances. A Kolmogorov power-law fit ([Ortiz-Suslow & Wang, 2019](#)) was optionally added by PARMESAN.

Acknowledgements

References

Büchau, Y. G., Leven, C., & Bange, J. (2023). *A Portable low-cost Device to Quantify Advective Gas Fluxes from Mofettes into the Lower Atmosphere: First Application to Starzach Mofettes (Germany)*. https://doi.org/10.31223/X5ZX0D

Büchau, Y. G., van Kesteren, B., Platis, A., & Bange, J. (2022). An Autarkic Wireless Sensor Network to Monitor Atmospheric CO2 Concentrations. *Meteorol. Z. (Contrib. Atm. Sci.)*. https://doi.org/10.1127/metz/2022/1125

Chizeck, H. J., Butterworth, E., & Bassingthwaighte, J. B. (2009). Error detection and unit conversion. *IEEE Engineering in Medicine and Biology Magazine*, *28*(3), 50–58. https://doi.org/10.1109/MEMB.2009.932477

Cooper, J., & McKeever, S. (2008). A model-driven approach to automatic conversion of physical units. *Software: Practice and Experience*, *38*(4), 337–359. https://doi.org/10.1002/spe.828

Foken, T. (2021). *Springer Handbook of Atmospheric Measurements*. Springer International Publishing. https://doi.org/10.1007/978-3-030-52171-4

Grecco, H., & Chéron, J. (2023). pint: Operate and Manipulate Physical Quantities in Python. In *GitHub repository*. GitHub. https://github.com/hgrecco/pint

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Herrmann, K. (2023). *Design Reiteration of a Chimney Gas Flowmeter for Natural CO2 Emissions from Mofettes: Differential Pressure Measurement Increases Resolution and Accuracy* [Bachelor's thesis]. University of Tübingen, Dept. of Geosciences. https://doi.org/10.15496/publikation-89337

Hoyer, S., & Hamman, J. (2017). xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.148

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., & team, J. development. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Scmidt (Eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (pp. 87–90). IOS Press. https://doi.org/10.3233/978-1-61499-649-1-87

Lebigot, E. O. (2023). Uncertainties: a Python package for calculations with uncertainties. In *GitHub repository*. GitHub. https://github.com/lebigot/uncertainties/

May, R. M., Goebbert, K. H., Thielen, J. E., Leeman, J. R., Camron, M. D., Bruick, Z., Bruning, E. C., Manser, R. P., Arms, S. C., & Marsh, P. T. (2022). MetPy: A Meteorological Python Library for Data Analysis and Visualization. *Bulletin of the American Meteorological Society*, *103*(10), E2273–E2284. https://doi.org/10.1175/BAMS-D-21-0125.1

McGibbon, J. (2023). aoslib: An atmospheric sciences library for Python. In *GitHub repository*. GitHub. https://github.com/atmos-python/atmos

Met Office. (2010 - 2023). *Iris: A powerful, format-agnostic, and community-driven Python package for analysing and visualising Earth science data* (v3.6 ed.). https://doi.org/10.5281/zenodo.7948293

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., … Scopatz, A. (2017). SymPy: symbolic computing in Python. *PeerJ Computer Science*, *3*, e103. https://doi.org/10.7717/peerj-cs.103

NASA. (1999). *Mars Climate Orbiter Mishap Investigation Board Phase I Report* (Mars Climate Orbiter Mishap Investigation Board, Ed.). National Aeronautics and Space Administration (NASA). https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf

Ortiz-Suslow, D. G., & Wang, Q. (2019). An Evaluation of Kolmogorov's -5/3 Power Law Observed Within the Turbulent Airflow Above the Ocean. *Geophysical Research Letters*, *46*(24), 14901–14911. https://doi.org/10.1029/2019GL085083

PyAOS. (2023). aoslib: A collection of standard atmospheric and oceanic sciences routines. In *GitHub repository*. GitHub. https://github.com/PyAOS/aoslib

Stull, R. B. (1988). *An Introduction to Boundary Layer Meteorology*. Kluwer Academic Publishers. https://doi.org/10.1007/978-94-009-3027-8

The pandas development team. (2023). *pandas-dev/pandas: Pandas* (Version v2.1.0rc0). Zenodo. https://doi.org/10.5281/zenodo.8239932

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wüsteney, S. (2023). *Bodenwassergehalts-Messung in tonigen Böden des Starzacher Mofetten-Gebiets mittels SMT100 Sensoren: Korrektur der Topp-Gleichung* [Bachelor's thesis]. University of Tübingen, Dept. of Geosciences. https://doi.org/10.15496/publikation-89327