









# flodym: A Python package for dynamic material flow analysis

Jakob Dürrwächter<sup>1</sup><sup>✉</sup>, Merlin Hosak<sup>1</sup>, Bennet Weiss<sup>1</sup><sup>1</sup>, and Falko Ueckerdt<sup>1</sup>

<sup>1</sup> Potsdam Institute for Climate Impact Research, Energy Transition Lab, Potsdam, Germany   
Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: 

Submitted: 15 July 2025

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

## Summary

Dynamic material flow analysis (MFA) is one of the core methods in the field of industrial ecology. It systematically tracks the time-dependent mass flows of materials within a system (such as a country's society) throughout different stages of their life cycle, and accounts for their accumulation in stocks, such as the in-use stock of materials contained in products, assets or infrastructure at a given point in time. Such analyses are used for resource management, for assessing environmental impacts, and investigating circular economy measures - assisting in policy advice, urban and regional planning, or sustainable product design.

*flodym* (Flexible Open Dynamic Material Systems Model) is a library of objects and functions needed to build dynamic MFA. It implements the `FlodymArray` class, which internally manages operations of one or several such multi-dimensional arrays. Objects representing flows, stocks, and parameters all inherit from this class. Stocks include lifetime models for dynamic stock modelling, i.e. for calculating the relation of material flows entering a stock and the mass and age structure of that stock over time. The whole MFA system is realized with an abstract parent class including sanity checks for the system, that users can implement a subclass of. *flodym* includes functionality for efficient read-in and export via pandas ([McKinney, 2010](#)), ([The pandas development team, 2020](#)), as well as visualization routines.

*flodym* is based on the concepts of the Open Dynamic Material Systems Model (ODYM) ([Pauliuk & Heeren, 2020](#)). It can be seen as a re-implementation with vastly expanded functionality and the aim of improved structuring. As a result, *flodym* enables users to effortlessly deploy a customized, flexible MFA - built for maintainability and future expansion.

## Statement of need

MFA is a widespread method in industrial ecology and related fields, which makes general and accessible MFA tools vital for a large audience in academia and beyond.

While there are several existing open source MFA software packages, ODYM ([Pauliuk & Heeren, 2020](#)) is, to the knowledge of the authors, the only general and adaptable open-source MFA library, allowing users to write their own MFA with the full range of options that custom code offers. ODYM is therefore widely used in the industrial ecology community and beyond. One of ODYM's strengths is that it builds on an abstraction of the principles and structures of MFAs, such as:

- formalizing a system definition and establishing mass conservation checks
- formalizing dynamic stock models, as described later in Lauinger et al. ([2021](#))
- translating the abstract concepts of processes, stocks, flows, and parameters into a general library, without prescribing any details about the MFA structure, such as its

41 dimensions.

42 *flodym* is based on the concepts of ODYM such that its structure, scope and strengths are  
43 similar to ODYM. However, there are also aspects in which *flodym* aims to fill gaps and add  
44 value, setting it apart from the original:

- 45 ■ ODYM stores dimensionality information in its array objects, but does not harvest the  
46 full potential of this information. *flodym* uses dimensionality information for complete  
47 internal dimension management in operations of multi-dimensional arrays. For example,  
48 the ODYM-based code

49 `waste = np.einsum('trp,pw->trw', end_of_life_products, waste_share)`

50 reduces to

51 `waste[...] = end_of_life_products * waste_share`

52 using *FlodymArray* objects. This allows to write simpler code and reduces errors. For  
53 example, dimensions of the same size could simply be switched in the `einsum` statement,  
54 which yields wrong results but goes unnoticed by the code. More importantly, it makes  
55 the code flexible (hence the name *flodym*) for adaptation and extension. Since the  
56 dimensions of each object are not explicitly given for every array operation, but only  
57 once in the array definition, dimensions can be added, removed or re-ordered later with  
58 minimal changes to the source code.

- 59 ■ Slicing is eased in a similar way. If, for example, only the values of the waste array for  
60 the C (carbon) entry of the `element` dimension are needed, the ODYM syntax

61 `waste.Values[:,0,:,:]`

62 simplifies to

63 `waste['C']`

64 Again, this allows for adding or removing other dimensions later, or changing the position  
65 of the C entry in the `element` dimension, without having to change the code. Apart  
66 from these functionalities, which are built on Python's magic methods, *FlodymArrays*  
67 feature a large range of built-in conventional methods for dimension manipulation, such  
68 as `sum_over`, `cast_to` or `get_shares_over`.

- 69 ■ Data read-in and initialization in ODYM prescribes a strict format based on Excel files.  
70 There is no data export functionality. In *flodym*, data read-in and export are based on  
71 pandas, opening them to a wide range of formats. Users can either use pre-built *flodym*  
72 read-in functions, or write their own, and generate objects from data frames. On data  
73 read-in, *flodym* performs checks on the data, detecting errors early on. Data read-in is  
74 performance-optimized especially for sparse arrays, since the full array size is only used  
75 after converting the input pandas data frame to a numpy array. Data is type-checked  
76 through the use of pydantic (Colvin et al., 2025), adding robustness to the code.

- 77 ■ ODYM contains the possibility of data export to a non-Python Sankey plotting tool, but  
78 no other visualization tools. In *flodym*, general visualization routines are implemented for  
79 pyplot (Hunter, 2007) and plotly (Plotly Technologies Inc., 2015) visualization, including  
80 plotting of multi-dimensional arrays, and Sankey plots of the MFA system.

- 81 ■ In ODYM, the class for dynamic stock models does not allow for dimensions apart from  
82 time. It also does not contain integrated methods for all required computation steps.  
83 Moreover, the stock objects which are used in the MFA system do not contain inflow,  
84 outflow, and stock, but only one of the three, distinguished by a `Type` attribute. To  
85 transfer the results of the dynamic stock model into the MFA, one has to loop over  
86 all non-time dimensions, run several sub-methods of the scalar dynamic stock model,  
87 and transfer the results into the MFA arrays. This is somewhat cumbersome and a  
88 performance bottleneck. In *flodym*, the treatment of material stocks is simplified and

integrated with the rest of the MFA. This is realized through Stock objects containing FlodymArray objects for inflow, outflow and stock arrays, as well as a lifetime model and compute functions. Both stock and lifetime model are multi-dimensional and part of the mfa system class, such that the interaction with them is seamless and the performance gains of numpy array operations are leveraged.

- *flodym* features various smaller functional extensions compared to ODYM. For example, stock models can handle non-evenly-spaced time step vectors, or sub-year lifetimes.
- ODYM features several great application examples, but only a partial API reference, and the API does not always follow PEP 8 naming conventions. The whole *flodym* code incorporates principles of software development (such as PEP 8 formatting, or GitHub actions for tests and documentation building) and clean code, easing future collaboration and extension. The code is extensively documented, including docstrings, type hints, an API reference, how-tos and examples.

Other existing open MFA packages such as OMAT (Villalba & Hoekman, 2018) or STAN (Cencic & Rechberger, 2008) are different in scope: They are no libraries, but rather comprehensive tools, which eases their use, but limits the flexibility for using them in non-standard ways like *flodym* allows. The same applies to the pymfa (Thiébaud et al., 2019) and PMFA (Kawecki-Wenger, n.d.) packages, which are moreover focused on probabilistic MFA as an extension or special case of MFA.

## Applications

So far, *flodym* is used in-house for the REMIND-MFA (Dürrwächter et al., 2025) and the external TRANSIENCE EU MFA (Saurat et al., 2025). Further external applications are currently in early development stage.

## Acknowledgements

Thank you to Stefan Pauliuk and other contributors to ODYM (Pauliuk & Heeren, 2020), which forms the conceptual basis for *flodym*.

We gratefully acknowledge funding from the TRANSIENCE project, grant number 101137606, funded by the European Commission within the Horizon Europe Research and Innovation Programme, from the Kopernikus-Projekt Ariadne through the German Federal Ministry of Education and Research (grant no. 03SFK5A0-2), and from the PRISMA project funded by the European Commission within the Horizon Europe Research and Innovation Programme under grant agreement No. 101081604 (PRISMA).

## References

- Cencic, O., & Rechberger, H. (2008). Material flow analysis with software STAN. In *Environmental informatics and industrial ecology*. Shaker Verlag.
- Colvin, S., Jolibois, E., Ramezani, H., Garcia Badaracco, A., Dorsey, T., Montague, D., Matveenko, S., Trylesinski, M., Runkle, S., Hewitt, D., Hall, A., & Plot, V. (2025). *Pydantic* (Version v2.11.7). <https://docs.pydantic.dev/latest/>
- Dürrwächter, J., Hosak, M., Weiß, B., Zhang, Q., & Ueckerdt, F. (2025). *REMIND-MFA*. <https://github.com/pik-piam/remind-mfa>
- 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>

- 131 Kawecki-Wenger, D. (n.d.). *PMFA: Base functions for performing a probabilistic MFA using r*.  
132 <https://github.com/empa-tsl/PMFA>
- 133 Lauinger, D., Billy, R. G., Vásquez, F., & Müller, D. B. (2021). A general framework for stock  
134 dynamics of populations and built and natural environments. *Journal of Industrial Ecology*,  
135 25(5), 1136–1146. <https://doi.org/10.1111/jiec.13117>
- 136 McKinney, Wes. (2010). Data Structures for Statistical Computing in Python. In Stéfan van  
137 der Walt & Jarrod Millman (Eds.), *Proceedings of the 9th Python in Science Conference*  
138 (pp. 56–61). <https://doi.org/10.25080/Majora-92bf1922-00a>
- 139 Pauliuk, S., & Heeren, N. (2020). ODOM—an open software framework for studying dynamic  
140 material systems: Principles, implementation, and data structures. *Journal of Industrial*  
141 *Ecology*, 24(3), 446–458. <https://doi.org/https://doi.org/10.1111/jiec.12952>
- 142 Plotly Technologies Inc. (2015). *Collaborative data science*. Plotly Technologies Inc. <https://plot.ly>
- 144 Saurat, M., Lotz, T. M., Bußmann, S., & Holtz, G. (2025). *TRANSIENCE-EU-MFA*. <https://transience-eu-mfa.readthedocs.io>
- 146 The pandas development team. (2020). *Pandas-dev/pandas: pandas* (latest). Zenodo.  
147 <https://doi.org/10.5281/zenodo.3509134>
- 148 Thiébaud, E., Alexandru, C., Badat, R., Kohler, D., & Hilty, L. (2019). *pymfa2: Ein*  
149 *werkzeug zur analyse von materialflüssen in python 2.7* (Version 2.1) [Computer software].  
150 <https://bitbucket.org/Xeelk/pymfa2/src/master/>
- 151 Villalba, G., & Hoekman, P. (2018). Using web-based technology to bring hands-on urban  
152 material flow analysis to the classroom. *Journal of Industrial Ecology*, 22(2), 434–442.  
153 <https://doi.org/https://doi.org/10.1111/jiec.12553>