Cytometry Part A

ISAC CYTOMETRY Journal of Quantitative Cell Science PART A

TECHNICAL NOTE OPEN ACCESS

CytoNormPy Enables a Fast and Scalable Removal of Batch Effects in Cytometry Datasets

Tarik Exner[1,2] | Nicolaj Hackert[1,2] | Luca Leomazzi[3,4] | Sofie Van Gassen[3,4] | Yvan Saeys[3,4] | Hanns-Martin Lorenz[1] | Ricardo Grieshaber-Bouyer[1,5,6,7]

[1]Internal Medicine V, Hematology, Oncology and Rheumatology, Heidelberg University Hospital, Heidelberg, Germany | [2]Institute for Immunology, Heidelberg University Hospital, Germany | [3]Data Mining and Modeling for Biomedicine Group, VIB Center for Inflammation Research, Ghent, Belgium | [4]Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Ghent, Belgium | [5]Deutsches Zentrum Immuntherapie (DZI), Friedrich-Alexander-Universität Erlangen-Nürnberg and Universitätsklinikum Erlangen, Erlangen, Germany | [6]Department of Internal Medicine 5—Hematology and Oncology, Friedrich-Alexander-Universität Erlangen-Nürnberg and University Hospital Erlangen, Erlangen, Germany | [7]Department of Internal Medicine 3—Rheumatology and Immunology, Friedrich-Alexander-Universität Erlangen- Nürnberg and University Hospital Erlangen, Erlangen, Germany

Correspondence: Tarik Exner (tarik.exner@med.uni-heidelberg.de) | Ricardo Grieshaber-Bouyer (ricardo.grieshaber@fau.de)

Received: 2 January 2025 | **Revised:** 15 July 2025 | **Accepted:** 26 July 2025

Funding: This work was supported by European Union Horizon 2021 Research and Innovation Programme (101072891), Fonds Wetenschappelijk Onderzoek (1272823N), Deutsche Forschungsgemeinschaft (GR 5979/2-1, 517717827), German Research Foundation (DFG) (INST 35/1597-1 FUGG), Else Kröner-Fresenius-Stiftung (2022_EKEA.72), IZKF Erlangen (N10), State of Baden-Württemberg, and German Society for Rheumatology (DGRh).

Keywords: batch-correction | cytometry | CytoNorm | Python

ABSTRACT

Cytometry has evolved as a crucial technique in clinical diagnostics, clinical studies, and research. However, batch effects due to technical variation complicate the analysis of cytometry data in clinical and fundamental research settings and have to be accounted for. Here, we present a Python implementation of the widely used CytoNorm algorithm for the removal of batch effects, implementing the complete feature set of the recently published CytoNorm 2.0. Our implementation ran up to 85% faster than its R counterpart while being fully compatible with common single-cell data structures and frameworks of Python. We extend the previous functionality by adding common clustering algorithms and provide key visualizations of the algorithm and its evaluation. The CytoNormPy implementation is freely available on GitHub: https://github.com/TarikExner/CytoNormPy.

1 | Introduction

Cytometry is an essential technique to measure multiple features of single cells. As the scale of cytometry studies grows and longitudinal studies are performed, not all samples can be acquired simultaneously, which introduces batch effects into the data. Therefore, the analysis of cytometry data increasingly necessitates accurate and efficient methods for batch correction.

CytoNorm has recently been described as an algorithm for the removal of batch effects using cluster-specific marker normalization across batches [1, 2]. The algorithm uses reference samples measured in all batches to quantify technical variation between different batches in the dataset (summarized in Figure 1). CytoNorm has been widely adopted in the analysis of cytometry data in fundamental science (e.g., [3, 4]) and is currently the most cited algorithm among peer-reviewed methods for the removal of batch effects in mass cytometry (iMUBAC [5], CytofRUV [6], CytofBatchAdjust [7], and cyCombine [8]).

CytoNorm is currently implemented in R, one of the most widely used programming languages for data analysis, including

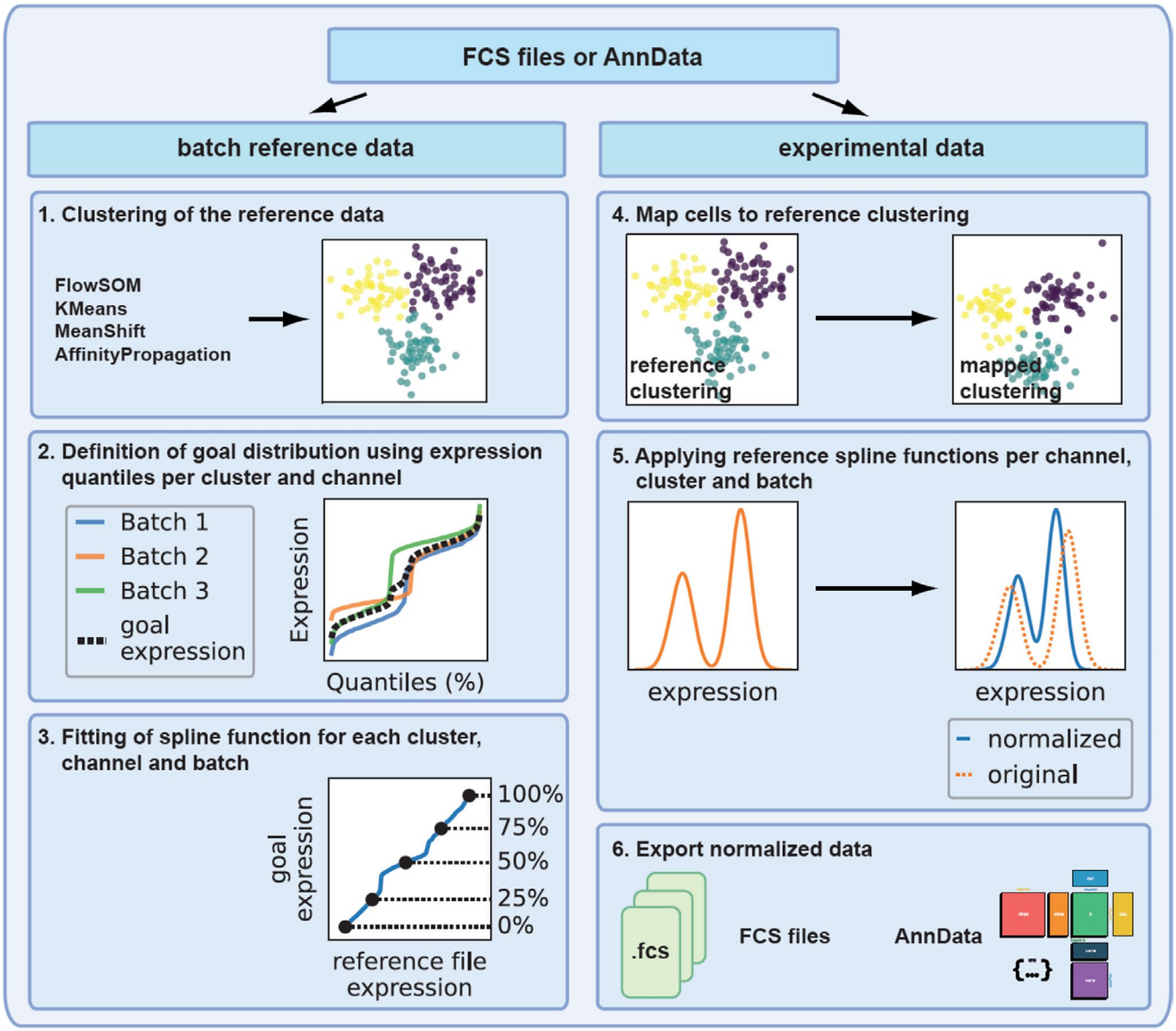

FIGURE 1 | CytoNorm/CytoNormPy algorithm overview. FCS files or anndata objects are divided into reference data (left side) and experimental data (right side) where reference data are comprised of samples that have been measured in all batches. Reference data are subsequently clustered (1) using one of the implemented clustering algorithms. Expression values are quantized, and a desired goal expression is defined, which defaults to the mean of all quantized expression values over the batches (2). A spline function between the expression values and the goal expression is fitted for each channel, cluster, and batch (3). The experimental data are then mapped to the reference cluster space (4) and the corresponding spline functions are used to transform the data (5). Finally, the data are either exported as .fcs files or returned as an anndata object containing the normalized expression values (6). [Color figure can be viewed at wileyonlinelibrary.com]

cytometry. However, Python has become increasingly popular in the analysis of single-cell data, as it offers a generalizable and flexible data structure (*anndata* [9]) and accompanying frameworks like the *scverse* [10] ecosystem, *scanpy* [11] and *pytometry* [11, 12], among others. Further, many popular machine/deep learning frameworks provide Python APIs, and Python often outperforms R in terms of scalability and speed in many applications.

Here, we present *cytonormpy*, a Python implementation of the CytoNorm algorithm. We show that *cytonormpy* runs significantly faster than CytoNorm, yielding near identical results with the original implementation and offering full compatibility with Python single cell analysis data structures, as normalized data within *anndata* objects can be readily used in downstream applications using scikit-learn, TensorFlow and the scVerse. We extend the original capabilities and add suitable clustering algorithms for cytometry data [13–16]. Cytonormpy fully implements the functionality that was recently added to the original CytoNorm algorithm (CytoNorm 2.0), namely normalization without control files, normalization towards another model or goal distribution and normalization of markers not included in the clustering step [2]. Finally, we provide visualization techniques to display key steps of the algorithm and its results.

2 | Materials and Methods

2.1 | Implementation Details

The implementation described here follows the steps implemented in the R package. For the use of .fcs files, the reference files from a sample present in all batches are first read using *flowio* [17], and annotated using user-provided metadata, whereas pre-assembled *anndata* [9] objects can directly be passed into the function. The reference data of batches not containing control files will be assembled by concatenation of all files of the respective batch and subsampling a user-defined number of cells. Next, the data are optionally transformed using one of the implemented transformation methods in flowutils [17], namely *log*, *logicle*, *arcsinh*, and *hyperlog* transformation. Data are subsequently clustered using the FlowSOM [18, 19], KMeans, MeanShift, or AffinityPropagation (scikit-learn [20] implementation) algorithm, where a specific set of markers can be chosen for clustering while still normalizing the full set of markers. A user-defined number of quantiles (default: 99) is then calculated per marker, cluster, and batch, which are saved into a pre-allocated *numpy* nd-array for more efficient data storage and access. Quantized expression values of markers of cells in clusters with less than a user-defined number of cells (default: 50 cells per cluster) are masked out, and the corresponding clusters are excluded from further calculations. The desired marker distribution across batches (goal distribution) is calculated by taking the mean of the expression quantiles over batches. Alternatively, the median can be used, or the expression values of a specific batch can be selected. Next, spline functions are calculated for each marker and cluster using the batch-specific quantile expression values and the respective goal distribution.

The fitting of the spline function follows the R implementation exactly. First, duplicate values within the quantile expression values are removed using a custom implementation of R's *regularize.values* function using the mean as a tie-function. Subsequently, a cubic hermite spline is calculated using the *scipy* [21] implementation. The interpolating tangents are selected using the Fritsch–Carlson method [22] using a custom implementation modeled after the R stats library. The spline function outside of the fitted values is linearly extended using the slope of the first and last datapoint.

The final normalization step of non-reference files is carried out as follows: First, the files are read concurrently using multi-threading. The data are subsequently mapped onto the clusters obtained from the clustering algorithms trained on the reference data. The cluster-specific expression values are finally normalized using the respective spline functions. FCS data were written to the hard drive, whereas *anndata* objects were returned.

For the evaluation of batch-effect removal, we implemented the calculation of Earth Mover's distance (EMD) according to the R implementation of CytoNorm. The EMD is computed before and after normalization as maximum sample-pairwise EMD between batches. As the EMD is a measure of the dissimilarity of signal distribution of the same sample in different batches, the numerical value is expected to be lowered after normalization. The corresponding plot (compare Figure 2D) shows a red dashed line that marks the EMD alteration cutoff, where points above this line represent an improvement, and points below this line mark a decline in batch effect removal after normalization. The EMD can be calculated per cell type and channel or per channel only.

Finally, in order to quantify the conservation of biological signal, the median absolute deviation (MAD) per channel in the original and the normalized samples is computed. Ideally, the signal distribution, and therefore the biological information, is not altered by normalization. We chose two dashed lines as a cutoff that signifies a marked deviation of the signal distribution (compare Figure 2D).

For functions carrying out numerical operations, including the regularize_values function, the Fritsch–Carlson tangent selection and quantile calculation, we make use of the *numba* [23] JIT (just in time) compilation package with eager compilation in nopython mode to take advantage of the increased performance of precompiled code in Python.

2.2 | Dataset Preprocessing and Benchmarking

For benchmarking and reproducibility comparison, we used three datasets in total (compare Table 1; detailed sample-wise descriptions and panel information are provided in the Supporting Information). The Van Gassen's [18] dataset consists of two healthy donors with stimulated and unstimulated data measured over 10 batches (40 samples). For the benchmark, samples of healthy control 1 were used as references, whereas samples of healthy control 2 were normalized. The dataset assembled by Trussart et al. [6] contained samples of healthy controls ($n = 3$) and patients with chronic leukocytic leukemia ($n = 3$) in two batches. The dataset was first pre-processed as described in their methods section and the samples HC1 and CLL2 were used as references. The third dataset (Dana Farber Cancer Institute [8], DFCI) consisted of three batches, of which the third one was used as a reference. The data were split into two panels with 15 overlapping markers and both panels were considered a single batch, following the procedure of the original publication [8]. The samples HD05 and CLL08 were used as references. For the benchmark, only the channels corresponding to antigens measured by antibodies were selected. All data were *arcsinh* transformed using a cofactor of 5. The reference data were clustered without subsampling using the FlowSOM algorithm with a grid size of 5×5 and a total of 10 clusters, and normalization was applied for each cluster as described above. As a control, the same analysis was repeated without clustering in order to remove the effect of inter-language deviation due to small differences in cluster assignment comparing the FlowSOM implementations in R and Python. In order to compare the different implemented clustering algorithms, we repeated the benchmark using the clustering algorithms on a subset of 20,000 cells, as the full dataset was prohibitively large for AffinityPropagation and MeanShift. Although the KMeans algorithm with the same cluster number as FlowSOM was slightly faster than FlowSOM (91%–97% of runtime across datasets), AffinityPropagation and MeanShift resulted in 3–23× increase in runtime, which is however expected as there

is no option to pre-define the number of clusters, which were much higher than the 10 clusters set for FlowSOM and KMeans (data not shown). Execution time was measured using the *time* module in Python (version 3.10) and the Sys library in R (version 4.4.0), respectively. For the benchmark, CytoNorm version 2.0.2 was used.

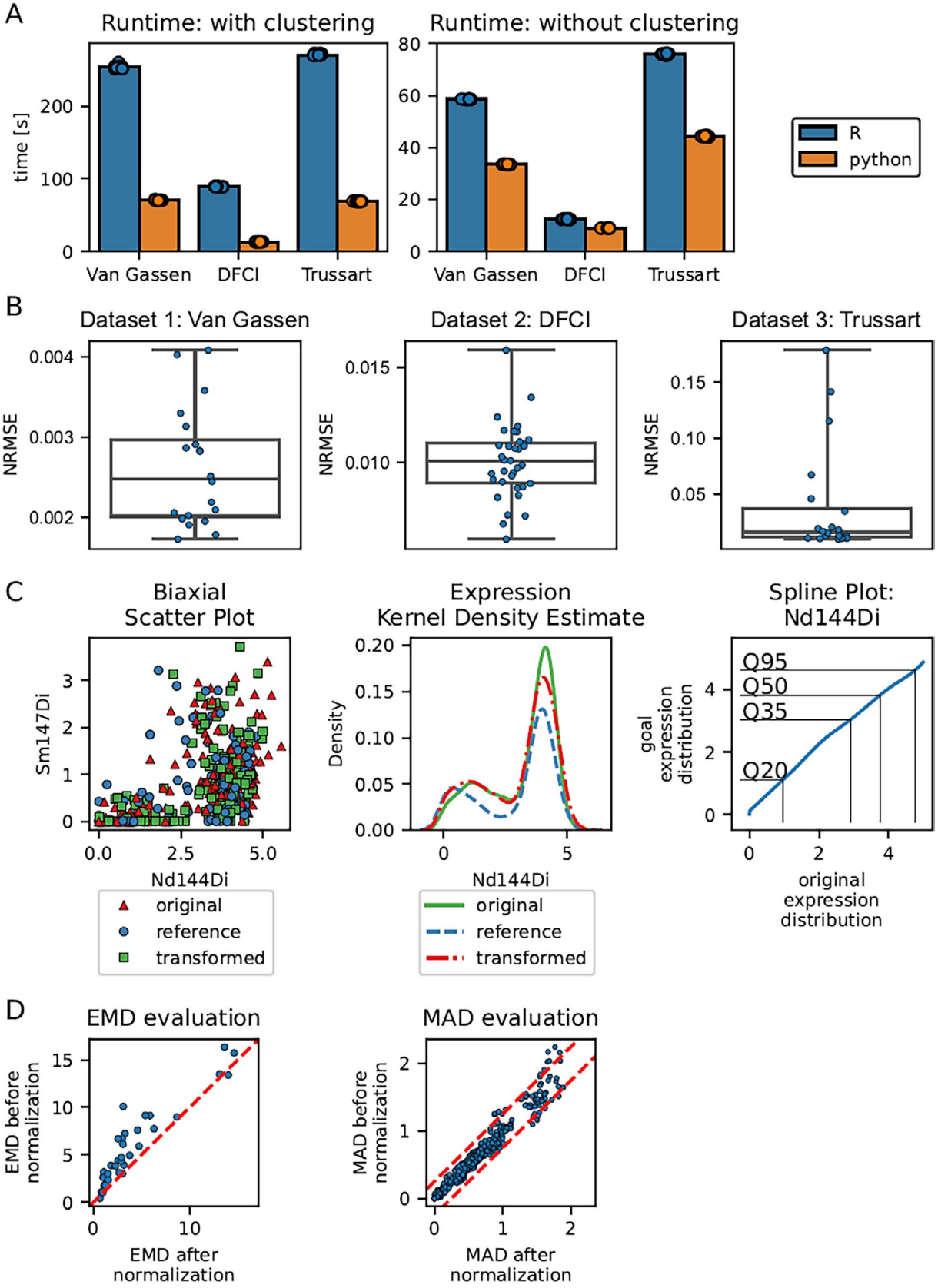

FIGURE 2 | Legend on next page.

FIGURE 2 | Benchmarking and cross-language comparison. (A) Runtime comparison. Runtime was determined on the indicated datasets for 10 runs. Displayed is the median runtime with (left plot) and without the clustering step (right plot) showing a significant improvement of runtime in the python implementation compared to R, especially if the default implementation including the clustering step was used. (B) Deviation of normalized expression values comparing python and R. Data were processed as described above, including the clustering step. The root mean squared error (RMSE) between the normalized values calculated from the R implementation and the python implementation was calculated per file and channel, normalized to the respective channel range and plotted as average over all channels per file (represented by the data points). (C) Visualization of the normalization procedure. CytoNormPy implements a scatter plot (left) and a kernel density estimate (middle) in order to visualize the effect of normalization. A spline plot (right) is used to visualize the fitted spline function where individual, user-defined quantile values can be displayed. For all plots, the Van Gassen dataset was used. (D) Visualization of the evaluation metrics. Left plot: Earth Mover's distances (EMD) as a measure to quantify batch effects are computed before and after normalization as the maximum sample-pairwise EMD between batches. Each data point corresponds to a single channel. The red dashed line marks a deviation of zero and points above this line represent channels where the EMD improved after normalization and therefore signifies a batch effect reduction. Right plot: Median absolute deviations (MAD) are calculated per file and channel as a measure of biological signal conservation before and after normalization, under the premise that normalization should not significantly alter these values. In the scatter plot, the close alignment of most points along the diagonal suggests minimal deviation from the expected post-normalization MAD values. The red dashed lines (here: ±0.25) serve as visual thresholds, demarcating boundaries beyond which changes in MAD values might indicate a notable effect on the biological signal distribution. [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 1 | Datasets used in this study.

Dataset	FlowRepository accession	*n* batches	*n* samples (reference/validation)	*n* channels	*n* cells (reference/validation)
Van Gassen	FR-FCM-Z247	10	40 (20/20)	37	6.193.423 (2.842.208/3.351.215)
Trussart	FR-FCM-Z2L2	2	24 (4/20)	34	8.589.739 (2.095.046/6.494.693)
DFCI	FR-FCM-Z52G	2	39 (4/35)	15	1.910.861 (508.055/1.402.806)

Note: Statistics for the number of batches within the respective dataset (*n* batches) as well as the number of individual files (*n* samples), the channel count (*n* channels), and the number of cells in the reference and validation subsets (*n* cells) are given. All datasets are mass cytometry datasets.

3 | Results and Discussion

3.1 | Highly Enhanced Performance of the Python Implementation

We first measured the computation time of our implementation in direct comparison to the R implementation on an 8-core machine and 32GB RAM using three datasets (Table 1 and Section 2). Using the default implementation including the clustering step, we found that the Python implementation led to a speed increase of 72%–85% (Figure 2A). Upon further inspection of the runtime of the individual steps, we noticed that most of the runtime of cytonormpy was spent in I/O operations and the clustering step (Figure S1A). As the R implementation uses frequent saving to the hard drive to save cluster-specific data, the observed speed difference is best explained by the increased amount of I/O operations, as well as the cluster inference in non-reference data, as it has be shown that the FlowSOM clustering itself has a lower runtime in Python compared to its R counterpart (Figure S1B) [19]. In order to evaluate the performance independent of the intra-algorithm I/O operations and FlowSOM clustering itself, we compared the runtimes while skipping the clustering step. Here, our implementation showed a noticeable performance benefit of 27%–43% compared to the implementation in R. This suggests that implementation choices in the R version, such as repeated disk I/O and non-vectorized loops, are likely responsible for the slower performance. We note that a carefully re-engineered R implementation, for example, using Rcpp and optimized data flow, could likely reduce runtime substantially. However, our focus here is not on benchmarking programming languages, but on providing a fast and well-integrated CytoNorm implementation for Python workflows.

3.2 | CytoNormPy Yields Almost Identical Results Compared to CytoNorm

We next tested the difference of the resulting normalized values comparing the R and Python implementation. We found that the values were highly concordant as judged by the normalized RMSE values per file (Figure 2B). We could trace the numerical differences back to small deviations in cluster assignment of the cells, as skipping the clustering step resulted in identical values for the batch-normalized values between Python and R (Figure S1C).

3.3 | Visualization Techniques

We implemented a dedicated module for data visualization. In addition to a normal bi-axial scatter plot, we provide tools to visualize the normalization impact using kernel density estimates.

A spline plot was implemented that plots the expression values and the corresponding goal distribution as a line plot, labeling user-defined quantiles for a clear display of the underlying fitted function (Figure 2C). For the visualization of the evaluation metrics EMD and the MAD, scatter plots are provided that plot the respective metrics before and after normalization (Figure 2D and Section 2). At last, we added the recently added waterfall plot to visualize the sample distribution across clusters [2]. The resulting plots are fully customizable using the widely used matplotlib and seaborn libraries.

4 | Conclusion

We present a Python implementation of the CytoNorm algorithm, which has been widely used to remove batch effects by reference-driven data normalization. While being fully compatible with existing frameworks such as *scanpy* or *pytometry*, we show that our implementation outperforms the R implementation significantly in terms of runtime while yielding highly comparable results. Finally, dedicated visualization capabilities enable a comprehensive diagnostic understanding of the algorithm by users and allow quantification of the effect of batch normalization and conservation of biological signal.

Acknowledgments

T.E. was supported by an MD/PhD fellowship from the Medical Faculty of Heidelberg. L.L. has received funding from the European Union's Horizon 2021 Research and Innovation Programme under the Marie Skłodowska-Curie (grant agreement no. 101072891). S.V.G. is supported by an FWO Postdoctoral Research Grant (1272823N, Research Foundation—Flanders). This work was supported by grants to R.G.-B. from Deutsche Forschungsgemeinschaft (DFG, GR 5979/2-1, 517717827), Else Kröner-Fresenius-Stiftung (2022_EKEA.72), the IZKF Erlangen (grant N10), State of Baden-Wuerttemberg within the Centers for Personalized Medicine Baden-Wuerttemberg (ZPM) and a research grant from the German Society for Rheumatology (DGRh). The authors acknowledge support by the State of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

All datasets used in our study are available at FlowRepository. For the accession numbers, refer to Table 1. The cytonormpy package is freely available on GitHub: https://github.com/TarikExner/CytoNormPy. The code to reproduce the data and figures used in this paper is provided in the S2.

References

1. S. Van Gassen, B. Gaudilliere, M. S. Angst, Y. Saeys, and N. Aghaeepour, "CytoNorm: A Normalization Algorithm for Cytometry Data," *Cytometry, Part A* 97, no. 3 (2020): 268–278.

2. K. L. A. Quintelier, M. Willemsen, V. Bosteels, J. Aerts, Y. Saeys, and S. Van Gassen, "CytoNorm 2.0: A Flexible Normalization Framework for Cytometry Data Without Requiring Dedicated Controls," *Cytometry, Part A* 107, no. 2 (2025): 69–87.

3. L. van Olst, A. Kamermans, S. Halters, et al., "Adaptive Immune Changes Associate With Clinical Progression of Alzheimer's Disease," *Molecular Neurodegeneration* 19, no. 1 (2024): 38.

4. A. Cevirgel, M. Vos, A. F. Holtrop, et al., "Delineating Immune Variation Between Adult and Children COVID-19 Cases and Associations With Disease Severity," *Scientific Reports* 14, no. 1 (2024): 5090.

5. M. Ogishi, R. Yang, C. Gruber, et al., "Multibatch Cytometry Data Integration for Optimal Immunophenotyping," *Journal of Immunology* 206, no. 1 (2021): 206–213.

6. M. Trussart, C. E. Teh, T. Tan, L. Leong, D. H. D. Gray, and T. P. Speed, "Removing Unwanted Variation With CytofRUV to Integrate Multiple CyTOF Datasets," *eLife* 9 (2020): e59630.

7. R. P. Schuyler, C. Jackson, J. E. Garcia-Perez, et al., "Minimizing Batch Effects in Mass Cytometry Data," *Frontiers in Immunology* 10 (2019): 2367.

8. C. B. Pedersen, S. H. Dam, M. B. Barnkob, et al., "cyCombine Allows for Robust Integration of Single-Cell Cytometry Datasets Within and Across Technologies," *Nature Communications* 13, no. 1 (2022): 1698.

9. I. Virshup, S. Rybakov, F. J. Theis, P. Angerer, and F. A. Wolf, "anndata: Annotated Data," *bioRxiv* (2021): 2021.12.16.473007, https://doi.org/10.1101/2021.12.16.473007.

10. I. Virshup, D. Bredikhin, L. Heumos, et al., "The Scverse Project Provides a Computational Ecosystem for Single-Cell Omics Data Analysis," *Nature Biotechnology* 41, no. 5 (2023): 604–606.

11. F. A. Wolf, P. Angerer, and F. J. Theis, "SCANPY: Large-Scale Single-Cell Gene Expression Data Analysis," *Genome Biology* 19, no. 1 (2018): 15.

12. M. Büttner, F. Hempel, T. Ryborz, F. J. Theis, and J. L. Schultze, "Pytometry: Flow and Mass Cytometry Analytics in Python," *bioRxiv* (2022): 2022.10.10.511546, https://doi.org/10.1101/2022.10.10.511546.

13. P. Liu, S. Liu, Y. Fang, et al., "Recent Advances in Computer-Assisted Algorithms for Cell Subtype Identification of Cytometry Data," *Frontiers in Cell and Developmental Biology* 8 (2020): 234.

14. X. Liu, W. Song, B. Y. Wong, et al., "A Comparison Framework and Guideline of Clustering Methods for Mass Cytometry Data," *Genome Biology* 20, no. 1 (2019): 297.

15. N. Aghaeepour, R. Nikolic, H. H. Hoos, and R. R. Brinkman, "Rapid Cell Population Identification in Flow Cytometry Data," *Cytometry, Part A* 79, no. 1 (2011): 6–13.

16. D. Jeffries, I. Zaidi, B. de Jong, M. J. Holland, and D. J. C. Miles, "Analysis of Flow Cytometry Data Using an Automatic Processing Tool," *Cytometry, Part A* 73A, no. 9 (2008): 857–867.

17. S. White, J. Quinn, J. Enzor, et al., "FlowKit: A Python Toolkit for Integrated Manual and Automated Cytometry Analysis Workflows," *Frontiers in Immunology* 12 (2021): 768541.

18. S. Van Gassen, B. Callebaut, M. J. Van Helden, et al., "FlowSOM: Using Self-Organizing Maps for Visualization and Interpretation of Cytometry Data," *Cytometry, Part A* 87, no. 7 (2015): 636–645.

19. A. Couckuyt, B. Rombaut, Y. Saeys, and S. Van Gassen, "Efficient Cytometry Analysis With FlowSOM in Python Boosts Interoperability With Other Single-Cell Tools," *Bioinformatics* 40, no. 4 (2024): btae179.

20. F. Pedregosa, G. Varoquaux, A. Gramfort, et al., "Scikit-Learn: Machine Learning in Python," *Journal of Machine Learning Research* 12 (2011): 2825–2830.

21. P. Virtanen, R. Gommers, T. E. Oliphant, et al., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods* 17, no. 3 (2020): 261–272.

22. F. N. Fritsch and R. E. Carlson, "Monotone Piecewise Cubic Interpolation," *SIAM Journal on Numerical Analysis* 17, no. 2 (1980): 238–246.

23. S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-Based Python JIT Compiler," (2015): 1–6.

Supporting Information

Additional supporting information can be found online in the Supporting Information section. **Data S1:** cytoa24953-sup-0001-Supinfo.zip. **Figure S1:** Function benchmark and comparison. (A) Runtime comparison of the individual function parts using FCS files as a data input. The runtime was measured for the indicated steps of cytonormpy. The majority of runtime is spent in read/write (I/O) and clustering operations. The relative difference between the ratios is best explained by the number of cells in the reference- and validation-set, respectively (compare Table 1). Reading (I) includes both the data reading from disk as well as a transformation step. Writing (O) includes a reverse transformation and writing to disk. (B) Approximate runtime comparison of individual function stages using FCS files as input. Functionspecific timings for R were obtained via Rprof and allocated post hoc to each algorithm step (covering ~90% of total runtime, due to ambiguous calls and function calls that could not be unambiguously assigned to individual steps). Although the Python implementation outperforms R across all stages, the clustering- and intermediate-I/O-steps account for most of the performance gap. (C) Deviation of normalized expression values. Data were subjected to the cytonormpy algorithm, omitting the clustering step. The root mean squared error (RMSE) between the normalized values calculated from the R implementation and the Python implementation was calculated per file and channel, normalized to the respective channel range and plotted as average over all channels per file (represented by the data points).