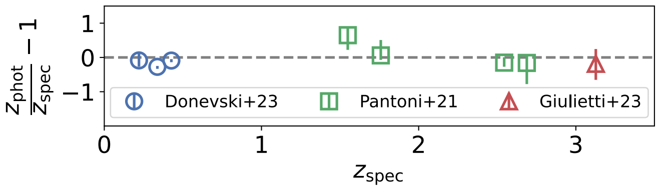

Fig. 27. Relative difference of photometric redshift estimated with GalaPy against the real spectroscopic values for the four dusty star-forming galaxies of the Pantoni et al. (2021) sample (green squares), the three quiescent galaxies of the Donevski et al. (2023) sample (blue circles), and the lensed galaxy with upper limits from (Giulietti et al. 2023, red triangle). Markers with error bars trace the median and 68% credible interval of the samples.

terms of age (Table 3) and stellar mass content (Table 4). In the original work, the authors used a truncated delayed SFH which, in case after truncation the star formation drops to zero, has a functional form which can be easily emulated by our quenched In situ shape, as also anticipated in Sect. 4.2.2.

4.2.6. Photometric redshift

We performed a final validation test on our machinery by inferring an estimate for the photometric redshift of real sources. We selected the sources from Sects. 4.2.1, 4.2.5, and 4.2.3, for which a measurement of the spectroscopic redshift is available. We then sampled again the parameter space by letting the `redshift` parameter vary along with the other free parameters.

In Fig. 27, we compare the photometric redshift prediction to the real value measured spectroscopically, in terms of the relative redshift difference between estimated ($z_{\rm phot}$) and fiducial value ($z_{\rm spec}$). We use coloured markers with error bars for the median value and 68% credible interval of the samples. The dashed grey line marks the real spectroscopic value.

For all the sources, the median values show at most a 2-σ difference with respect to the spectroscopic measurement (i.e. the fiducial value is within the 95% credible interval). Apart from the low redshift sources and most of the P21 sources, the photometric prediction of the J1135 redshift from Giulietti et al. (2023) is extremely close to the expected value (order of percent relative difference). This is a remarkable result, considering the large number of fluxes for which only upper limits are available, especially in the UV/optical part of the spectrum. The reason for this agreement lays on the interplay between the thorough sampling of the dust peak and the precision modelling allowed by our two-component, age-dependent dust model.

These results confirm the reliability of photometric redshift estimates obtained with GalaPy on real sources, as had already been demonstrated on mock sources. This is an asset that will prove powerful for future observational campaingns targeting distant sources (e.g. JWST). In the next future, we also plan to further test the photometric redshift determination capabilities of GalaPy against large datasets up to the highest redshifts currently available, for instance, A3COSMOS (Liu et al. 2019a,b; Fudamoto et al. 2020) and COSMOS-Web (Casey et al. 2023).

5. Summary

We present GalaPy, a highly optimised, open-source, hybrid library for parameterised fitting of the spectral energy distributions (SEDs) of galaxies. The tool currently focuses on photometric SED fitting from galaxies, but future versions will extend its functionalities to include spectroscopic fitting at variable resolutions and AGN modelling. The API is readily available through terminal entry-points or by importing modules from the `galapy` package. The full documentation, including examples and API usage manual, is available on ReadTheDocs and the code is available on GitHub.

In Sect. 2, we provide a detailed description of the physical models implemented in GalaPy, with a particular focus on the in situ star formation histories and the two-component age-dependent dust model. The former provides a model for the evolution of the extended structure components of a galaxy that depends on both the infall of material in the DM halo and on the evolution of the nuclear regions, driven by the central black hole (Lapi et al. 2018, 2020; Pantoni et al. 2019). The latter, provides a physically motivated model of the time evolution of dust, with overall attenuation directly derived by the contribution of each single simple stellar population hosted in the galaxy. Additionally, GalaPy uses an age-dependent, energy-conservation scheme to derive the evolution of dust temperatures in an analytic way.

In Sect. 3, we describe the statistical tools used to obtain parameter posteriors. The parameter space sampling is based on Bayesian inference methods and we provide interfaces to two samplers, `emcee` and `dynesty`. In Sect. 4, we demonstrate the efficacy of GalaPy by testing it on various cases, including dusty star-forming galaxies at high redshift, local late-type, and early-type galaxies, along with a NIR-dark, lensed high-redshift galaxy with mostly upper limits. We have demonstrated that GalaPy can be used to study the main physical characteristics of galaxies, such as their star formation histories, matter content, and physical parameters.

Future extensions of GalaPy include spectroscopic fitting and Hamiltonian parameter space sampling, as well as a hierarchical Bayesian scheme for modelling datasets from large catalogues with correlated systematic errors (see e.g. Kelly et al. 2012; Galliano 2018). Additionally, a consistent modelling of the AGN within the BH-galaxy co-evolution In situ scenario will be introduced soon. Finally, we plan to accelerate posterior inference using active learning techniques.

In conclusion, GalaPy is a timely and valuable tool for the astrophysical community that offers a powerful, self-consistent framework for modelling the SED of galaxies, based on physically-motivated models and a Bayesian statistical approach (in Appendix C.3, we provide recommendations on how to properly acknowledge usage of the library). The physical models implemented in GalaPy, together with the optimisations made to the fitting algorithms, enable the tool to provide robust and accurate parameter estimates for a wide range of astrophysical applications. The main characterising features of GalaPy are

– self-consistent modelling of the SFH and derived physical properties that not only reduces the size of the parameter space, but it also allows for a straightforward derivation of the physical properties characterising the galaxy and is specifically designed to follow the evolution of high redshift progenitors up to their quiescence, leading to the formation of local early type galaxies;
– two component time-dependent energy-conserving treatment of dust attenuation and re-radiation that allows for both a physical treatment of the process without assuming unknown physics of the dust-grain and for a computationally-efficient balancing of energy;
– high resolution integration of stellar populations for the study of primordial galaxies that does not burden the computation, thanks to a memory-efficient caching of the SSP grid (thoroughly treated in Appendix A.1.1);

- easily extensible database of cosmological models, SSP libraries, and photometric band-pass filters;
- user-friendly API and extensive documentation, allowing for high level of customisation;
- state-of-the-art hybrid C++/Python implementation, reaching high performances with minimal memory consumption;
- Bayesian framework for the inference of posteriors in the parameter space.

As current and upcoming observational campaigns (e.g. JWST, LSST, SKAO) continue to generate ever-increasing amounts of data, the capabilities of GalaPy will become increasingly important for understanding the physical properties of galaxies, especially in the high-redshift Universe, and their evolution over cosmic time.

Acknowledgements. The authors are grateful to the anonymous referee for the detailed and thorough comments that contributed to improve substantially the present work. The authors are grateful to Nicoletta Krachmalnicoff for the extensive and patient discussion on statistical inference and for her useful suggestions that improved the readability of the software-related sections of this work. The authors also thank Stephàn Charlot for stimulating discussion and for his support to the project. This paper is supported by: "Data Science methods for MultiMessenger Astrophysics & Multi-Survey Cosmology" funded by the Italian Ministry of University and Research, Programmazione triennale 2021/2023 (DM n.2503 dd. 9 December 2019), Programma Congiunto Scuole; EU H2020-MSCA-ITN-2019 no. 860744 BiD4BESt: Big Data applications for black hole Evolution STudies; Italian Research Center on High Performance Computing Big Data and Quantum Computing (ICSC), project funded by European Union – NextGenerationEU – and National Recovery and Resilience Plan (NRRP) – Mission 4 Component 2 within the activities of Spoke 3 (Astrophysics and Cosmos Observations); PRIN MUR 2022 project no. 20224JR28W "Charting unexplored avenues in Dark Matter"; INAF Large Grant 2022 funding scheme with the project "MeerKAT and LOFAR Team up: a Unique Radio Window on Galaxy/AGN co-Evolution; INAF GO-GTO Normal 2023 funding scheme with the project "Serendipitous H-ATLAS-fields Observations of Radio Extragalactic Sources (SHORES)". DD acknowledges support from the National Science Center (NCN) grant SONATA (UMO-2020/39/D/ST9/00720). This work made use of the C++ (Stroustrup 2013) and Python (Van Rossum et al. 2007) programming languages, and of the following software: Astropy (The Astropy Collaboration 2013), NumPy (Harris et al. 2020), SciPy (Virtanen et al. 2020), Matplotlib (Hunter 2007), Emcee (Foreman-Mackey et al. 2013), dynesty (Speagle 2020), GetDist (Lewis 2019), pybind11 (Jakob et al. 2017), HDF5 (Folk et al. 2011), h5py (Collette et al. 2021).

References

Adams, N. J., Conselice, C. J., Ferreira, L., et al. 2023, MNRAS, 518, 4755
Amendola, L., Appleby, S., Avgoustidis, A., et al. 2018, Living Rev. Relativ., 21, 2
Atek, H., Shuntov, M., Furtak, L. J., et al. 2023, MNRAS, 519, 1201
Battisti, A. J., da Cunha, E., Grasha, K., et al. 2019, ApJ, 882, 61
Beckwith, S. V. W., Stiavelli, M., Koekemoer, A. M., et al. 2006, AJ, 132, 1729
Behiri, M., Talia, M., Cimatti, A., et al. 2023, ApJ, 957, 63
Bianchi, S., De Vis, P., Viaene, S., et al. 2018, A&A, 620, A112
Bischetti, M., Maiolino, R., Carniani, S., et al. 2019, A&A, 630, A59
Blyth, S., van der Hulst, T. M., Verheijen, M. A. W., et al. 2015, in Advancing Astrophysics with the Square Kilometre Array (AASKA14), 128
Booth, R. S., & Jonas, J. L. 2012, Afr. Skies, 16, 101
Boquien, Burgarella, D., Roehlly, Y., et al. 2019, A&A, 622, A103
Bressan, A., Chiosi, C., & Fagotto, F. 1994, ApJS, 94, 63
Bressan, A., Granato, G. L., & Silva, L. 1998, A&A, 332, 135
Bressan, A., Silva, L., & Granato, G. L. 2002, A&A, 392, 377
Bressan, A., Marigo, P., Girardi, L., et al. 2012, MNRAS, 427, 127
Brisbin, D., Miettinen, O., Aravena, M., et al. 2017, A&A, 608, A15
Bruzual, G., & Charlot, S. 2003, MNRAS, 344, 1000
Calzetti, D., Armus, L., Bohlin, R. C., et al. 2000, ApJ, 533, 682
Camps, P., & Baes, M. 2020, Astron. Comput., 31, 100381
Carnall, A. C., McLure, R. J., Dunlop, J. S., & Davé, R. 2018, MNRAS, 480, 4379
Casasola, V., Bianchi, S., De Vis, P., et al. 2020, A&A, 633, A100
Casey, C. M., Kartaltepe, J. S., Drakos, N. E., et al. 2023, ApJ, 954, 31
Castellano, M., Fontana, A., Treu, T., et al. 2022, ApJ, 938, L15
Chabrier, G. 2003, PASP, 115, 763
Charlot, S., & Fall, S. M. 2000, ApJ, 539, 718
Chen, Y., Girardi, L., Bressan, A., et al. 2014, MNRAS, 444, 2525
Chen, Y., Bressan, A., Girardi, L., et al. 2015, MNRAS, 452, 1068
Chevallard, J., & Charlot, S. 2016, MNRAS, 462, 1415
Cimatti, A., Fraternali, F., & Nipoti, C. 2020, Introduction to Galaxy Formation and Evolution: From Primordial Gas to Present-day Galaxies (Cambridge University Press)
Clark, C. J. R., Verstocken, S., Bianchi, S., et al. 2018, A&A, 609, A37
Collette, A., Kluyver, T., Caswell, T. A., et al. 2021, `https://doi.org/10.5281/zenodo.5585380`
Cropper, M., Pottinger, S., Niemi, S., et al. 2016, SPIE Conf. Ser., 9904, 99040Q
Da Cunha, E., Charlot, S., & Elbaz, D. 2008, MNRAS, 388, 1595
Davies, J. I., Baes, M., Bianchi, S., et al. 2017, PASP, 129, 044102
De Vis, P., Jones, A., Viaene, S., et al. 2019, A&A, 623, A5
DESI Collaboration 2016, arXiv e-prints [arXiv:1611.00036]
Donevski, D., Damjanov, I., Nanni, A., et al. 2023, A&A, 678, A35
Doore, K., Monson, E. B., Eufrasio, R. T., et al. 2023, ApJS, 266, 39
Draine, B. T. 2011, Physics of the Interstellar and Intergalactic Medium (Princeton University Press)
Draine, B. T., & Li, A. 2001, ApJ, 551, 807
Draine, B. T., & Li, A. 2007, ApJ, 657, 810
Dudzevičiūtė, U., Smail, I., Swinbank, A. M., et al. 2020, MNRAS, 494, 3828
Dunlop, J. S., McLure, R. J., Biggs, A. D., et al. 2017, MNRAS, 466, 861
Duras, F., Bongiorno, A., Ricci, F., et al. 2020, A&A, 636, A73
Eales, S., Dunne, L., Clements, D., et al. 2010, PASP, 122, 499
Enia, A., Talia, M., Pozzi, F., et al. 2022, ApJ, 927, 204
Fabbiano, G. 2006, ARA&A, 44, 323
Ferland, G. J., Korista, K. T., Verner, D. A., et al. 1998, PASP, 110, 761
Ferland, G. J., Porter, R. L., van Hoof, P. A. M., et al. 2013, Rev. Mex. Astron. Astrofis., 49, 137
Ferland, G. J., Chatzikos, M., Guzmán, F., et al. 2017, Rev. Mex. Astron. Astrofis., 53, 385
Feroz, F., Hobson, M. P., & Bridges, M. 2009, MNRAS, 398, 1601
Ferrara, A., Sommovigo, L., Dayal, P., et al. 2022, MNRAS, 512, 58
Finkelstein, S. L., Bagley, M. B., Haro, P. A., et al. 2022, ApJ, 940, L55
Finkelstein, S. L., Bagley, M. B., Ferguson, H. C., et al. 2023, ApJ, 946, L13
Folk, M., Heber, G., Koziol, Q., Pourmal, E., & Robinson, D. 2011, in Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, 36
Fontanot, F., La Barbera, F., De Lucia, G., Pasquali, A., & Vazdekis, A. 2018, MNRAS, 479, 5678
Foreman-Mackey, D., Hogg, D. W., Lang, D., & Goodman, J. 2013, PASP, 125, 306
Förster Schreiber, N. M., & Wuyts, S. 2020, ARA&A, 58, 661
Fragos, T., Lehmer, B. D., Naoz, S., Zezas, A., & Basu-Zych, A. 2013, ApJ, 776, L31
Franco, M., Elbaz, D., Béthermin, M., et al. 2018, A&A, 620, A152
Fritz, J., Franceschini, A., & Hatziminaoglou, E. 2006, MNRAS, 366, 767
Fudamoto, Y., Oesch, P. A., Magnelli, B., et al. 2020, MNRAS, 491, 4724
Galliano, F. 2018, MNRAS, 476, 1445
Giavalisco, M., Ferguson, H. C., Koekemoer, A. M., et al. 2004, ApJ, 600, L93
Giulietti, M., Lapi, A., Massardi, M., et al. 2023, ApJ, 943, 151
González-López, J., Bauer, F. E., Romero-Cañizales, C., et al. 2017, A&A, 597, A41
Goodman, J., & Weare, J. 2010, Commun. Appl. Math. Comput. Sci., 5, 65
Gruppioni, C., Béthermin, M., Loiacono, F., et al. 2020, A&A, 643, A8
Hamed, M., Małek, K., Buat, V., et al. 2023, A&A, 674, A99
Harikane, Y., Ouchi, M., Oguri, M., et al. 2023, ApJS, 265, 5
Harris, A. I., Baker, A. J., Frayer, D. T., et al. 2012, ApJ, 752, 152
Harris, C. R., Millman, K. J., Van Der Walt, S. J., et al. 2020, Nature, 585, 357
Higson, E., Handley, W., Hobson, M., & Lasenby, A. 2019, Stat. Comput., 29, 891
Hodge, J. A., & da Cunha, E. 2020, Roy. Soc. Open Sci., 7, 200556
Hopkins, P. F., Quataert, E., & Murray, N. 2012, MNRAS, 421, 3522
Hotan, A. W., Bunton, J. D., Chippendale, A. P., et al. 2021, PASA, 38, e009
Hunt, L. K., De Looze, I., Boquien, M., et al. 2019, A&A, 621, A51
Hunter, J. D. 2007, Comput. Sci. Eng., 9, 90
Inoue, A. K., Shimizu, I., Iwata, I., & Tanaka, M. 2014, MNRAS, 442, 1805
Isobe, T., & Feigelson, E. D. 1986, Bull. Inform. Centre Données Stellaires, 31, 209
Jakob, W., Rhinelander, J., & Moldovan, D. 2017, pybind11 – Seamless operability between C++11 and Python, `https://github.com/pybind/pybind11`
Jarvis, M., Taylor, R., Agudo, I., et al. 2016, in MeerKAT Science: On the Pathway to the SKA, 6
Jin, S., Daddi, E., Liu, D., et al. 2018, ApJ, 864, 56
Jin, S., Daddi, E., Magdis, G. E., et al. 2022, A&A, 665, A3
Johnson, B. D., Leja, J., Conroy, C., & Speagle, J. S. 2021, ApJS, 254, 22
Johnston, S., Bailes, M., Bartel, N., et al. 2007, PASA, 24, 174

Johnston, S., Taylor, R., Bailes, M., et al. 2008, Exp. Astron., 22, 151
Jonas, J., & MeerKAT Team 2016, in MeerKAT Science: On the Pathway to the SKA, 1
Kelly, B. C., Shetty, R., Stutz, A. M., et al. 2012, ApJ, 752, 55
Koposov, S., Speagle, J., Barbary, K., et al. 2023, https://doi.org/10.5281/zenodo.7600689
Kroupa, P., Weidner, C., Pflamm-Altenburg, J., et al. 2013, in Planets, Stars and Stellar Systems, 5: Galactic Structure and Stellar Populations, eds. T. D. Oswalt, & G. Gilmore (Springer), 115
Labbé, I., van Dokkum, P., Nelson, E., et al. 2023, Nature, 616, 266
Lacey, C. G., Baugh, C. M., Frenk, C. S., et al. 2016, MNRAS, 462, 3854
Lapi, A., González-Nuevo, J., Fan, L., et al. 2011, ApJ, 742, 24
Lapi, A., Pantoni, L., Zanisi, L., et al. 2018, ApJ, 857, 22
Lapi, A., Pantoni, L., Boco, L., & Danese, L. 2020, ApJ, 897, 81
Lewis, A. 2019, arXiv e-prints [arXiv:1910.13970]
Liu, D., Lang, P., Magnelli, B., et al. 2019a, ApJS, 244, 40
Liu, D., Schinnerer, E., Groves, B., et al. 2019b, ApJ, 887, 235
LSST Science Collaboration 2009, arXiv e-prints [arXiv:0912.0201]
Lutz, D., Poglitsch, A., Altieri, B., et al. 2011, A&A, 532, A90
Mancuso, C., Lapi, A., Prandoni, I., et al. 2017, ApJ, 842, 95
Massardi, M., Stoehr, F., Bendo, G. J., et al. 2021, PASP, 133, 085001
Mayya, Y. D., Bressan, A., Rodríguez, M., Valdes, J. R., & Chavez, M. 2004, ApJ, 600, 188
McConnell, D., Allison, J. R., Bannister, K., et al. 2016, PASA, 33, e042
Murphy, E. J., Bremseth, J., Mason, B. S., et al. 2012, ApJ, 761, 97
Nagao, T., Maiolino, R., & Marconi, A. 2006, A&A, 459, 85
Naidu, R. P., Oesch, P. A., Setton, D. J., et al. 2022, ApJ, submitted [arXiv:2208.02794]
Negrello, M., Amber, S., Amvrosiadis, A., et al. 2016, MNRAS, 465, 3558
Noll, S., Burgarella, D., Giovannoli, E., et al. 2009, A&A, 507, 1793
Norris, R. P., Marvil, J., Collier, J. D., et al. 2021, PASA, 38, e046
Novak, M., Smolčić, V., Delhaize, J., et al. 2017, A&A, 602, A5
Oliver, S. J., Bock, J., Altieri, B., et al. 2012, MNRAS, 424, 1614
Pantoni, L., Lapi, A., Massardi, M., Goswami, S., & Danese, L. 2019, ApJ, 880, 129
Pantoni, L., Lapi, A., Massardi, M., et al. 2021, MNRAS, 504, 928
Pensabene, A., Carniani, S., Perna, M., et al. 2020, A&A, 637, A84
Pensabene, A., Decarli, R., Bañados, E., et al. 2021, A&A, 652, A66
Planck Collaboration VI. 2020, A&A, 641, A6
Rodighiero, G., Bisigello, L., Iani, E., et al. 2022, MNRAS, 518, L19
Salim, S., & Narayanan, D. 2020, ARA&A, 58, 529
Sawicki, M. 2012, PASP, 124, 1208
Schweitzer, M., Bender, R., Katterloher, R., et al. 2010, SPIE Conf. Ser., 7731, 77311K
Scoville, N., Aussel, H., Brusa, M., et al. 2007, ApJS, 172, 1
Scoville, N., Lee, N., Bout, P. V., et al. 2017, ApJ, 837, 150
Shapley, A. E. 2011, ARA&A, 49, 525
Shirley, R., Roehlly, Y., Hurley, P. D., et al. 2019, MNRAS, 490, 634
Shirley, R., Duncan, K., Campos Varillas, M. C., et al. 2021, MNRAS, 507, 129
Silva, L., Granato, G. L., Bressan, A., & Danese, L. 1998, ApJ, 509, 103
Simpson, J. M., Swinbank, A. M., Smail, I., et al. 2014, ApJ, 788, 125
Simpson, J. M., Smail, I., Swinbank, A. M., et al. 2017, ApJ, 839, 58
Simpson, J. M., Smail, I., Dudzevičiūtė, U., et al. 2020, MNRAS, 495, 3409
Skilling, J. 2004, AIP Conf. Ser., 735 395
Skilling, J. 2006, Bayesian Anal., 1, 833
Smail, I., Dudzevičiūtė, U., Stach, S. M., et al. 2021, MNRAS, 502, 3426
Smolčić, V., Delvecchio, I., Zamorani, G., et al. 2017, A&A, 602, A2
Speagle, J. S. 2020, MNRAS, 493, 3132
Stroustrup, B. 2013, The C++ Programming Language (Pearson Education)
Tacconi, L. J., Genzel, R., Saintonge, A., et al. 2018, ApJ, 853, 179
Tacconi, L. J., Genzel, R., & Sternberg, A. 2020, ARA&A, 58, 157
Talia, M., Cimatti, A., Giulietti, M., et al. 2021, ApJ, 909, 23
Targett, T. A., Dunlop, J. S., Cirasuolo, M., et al. 2013, MNRAS, 432, 2012
Taylor, A. R., & Jarvis, M. 2017, in Materials Science and Engineering Conference Series, 198, 012014
The Astropy Collaboration (Robitaille, T. P., et al.) 2013, A&A, 558, A33
Van Rossum, G. 2007, in USENIX annual technical conference, Python Programming Language, 41, 1
Vega, O., Clemens, M. S., Bressan, A., et al. 2008, A&A, 484, 631
Vidal-García, A., Plat, A., Curtis-Lake, E., et al. 2024, MNRAS, 527, 7217
Virtanen, P., Gommers, R., Oliphant, T. E., et al. 2020, Nature Methods, 17, 261
Walter, F., Decarli, R., Carilli, C., et al. 2012, Nature, 486, 233
Wang, T., Schreiber, C., Elbaz, D., et al. 2019, Nature, 572, 211
Yan, H., Ma, Z., Ling, C., Cheng, C., & Huang, J.-S. 2023, ApJ, 942, L9
Yun, M. S., Scott, K. S., Guo, Y., et al. 2012, MNRAS, 420, 957

Appendix A: Code design

In this section we provide insights on the design choices made both for optimising the performances of the library and with the intent of keeping the structure modular and user friendly. The bulk of the library resides in the computation of the parameterised models described in Sect. 2. Given that the main (but not only) purpose of GalaPy is to provide a lightning fast tool for parameter inference, these functions have been implemented to reach high performances on a single core. We reach this requirement by exploiting different advanced programming techniques, from register proximity, minimisation of operations and interpolation tactics. A description of the chosen strategies is provided in Sects. A.1 and A.2 showcases briefly the main modules and subpackages building up the library, while in Sect. A.3 we show some loose performance measurements of the main functionalities deployed in GalaPy. All the performance measurements presented in this Section have been obtained by running on a Intel i9-10885H CPU @ 2.40GHz personal computer with a `x86_64` architecture. The cache available to CPUs is of average size for modern machines, it has 8 private instances of L1 with 32 KB of memory per instance, 8 private instances of L2 with 256 KB per instance and 1 shared instance of L3 with 2 MB of memory.

Appendix A.1. Implementation strategy

GalaPy has a hybrid implementation which allows us to exploit both the performance efficiency of a compiled language (C++) as well as the flexibility of an interpreted language (Python). The Bushido of GalaPy software development can be summarised in the brief points below:

- compiled Object-Oriented **C++** crunches all the modelling framework, constituted of complex mathematical relations that burden the computation. The physical components described in Sect. 2 are implemented as independent classes, all of which share a common interface for parameter setting and computation of the eventual emission as a function of wavelength, age and metallicity. At construction time, all the quantities that do not depend on the free parameters of the given component are computed in advance and cached, therefore minimising the amount of operations the machine has to perform. Modelling though is still extremely light on the RAM, as the volatile memory occupied by this cached information does not go beyond a few tens of MB, mostly depending on the size of the SSP library chosen. This choice represents a compromise between the acceleration provided by SED grid interpolation and the flexibility of on-the-fly model computation. All of these objects can be serialised (i.e. converted in a sequential string of bits), allowing them to be picklable, therefore completely Python-compliant. Besides from physical models, the compiled sector of the library also implements a set of data-structures and algorithms for speeding up operations (in Appendix A.1.2 we describe the linear interpolation scheme we use in some parts of the library). Finally, the compiled sector also manages loading the SSP libraries used to compute stellar emission, this is done to favour CPU cache management as described in Appendix A.1.1. The only C++ library used is the STL, therefore minimising the problems that might arise in the installation of the package on different systems.
- **Python** deals with the interplay between all the components and modules, internal and external, that build up the library. It also provides the user-interface and an extensive documentation. Lastly, the terminal commands allowing for quick-access parameterised SED-fitting that come out-of-the-box with library installation (e.g. the `galapy-fit` command mentioned in Sect. C) are implemented as Python entry-points. By importing the `galapy` package and subpackages the GalaPy API is exposed, allowing for complete customisation of the algorithms as well as providing the tools for astrophysical modelling and analysis of the sampling results.
- **pybind11** is a library to generate Python bindings of compiled C++ code. We bind compiled classes and expose our optimised C++ implementation to the Python interface providing access to our functions to users. All the functions that can be applied to arrays of values are vectorised, providing a straightforward integration with the most common python packages for scientific computing (e.g. NumPy and SciPy) and therefore allow for array programming. We have chosen this strategy because, compared to a CPython wrapping layer, it delivers bindings with negligible latency while providing a more intuitive interface.

The primary purpose of GalaPy is to derive the parameters that can be inferred from the spectral properties of galaxies. Our code aims at delivering a high performance serial implementation of parametric SEDs, so that parallelism is not necessary in model generation (some performance testing is shown in Appendix A.3). In this way, the only bottleneck of the work-flow is parameter-space sampling.

Both `emcee` and `dynesty` allow for passing a pool of workers to the functions running the sampling. In GalaPy we generate pools exploiting the `multiprocessing` package of the Python standard library. Because of the structural limits of Python (i.e. the existence of a Global Interpreter Lock that guarantees parallel threads are not modifying concurrently the reference count in the Python interpreter), allocating a pool of parallel workers, with the intent of speeding-up CPU-bound workflows, requires us to generate a copy of the environment. Copying the whole environment though, results in the necessity of generating deep copies of all the variables that can be referenced in a given scope. This not only means a larger memory usage, but it also reduces the effectiveness of shared memory parallelism, as passing around chunks of memory slows down severely the computation.

In the entry-point provided for fitting SEDs (i.e. `galapy-fit`) the default behaviour tries to reach a compromise between memory usage and parallelism. The variables that require the larger memory budget (e.g. the SSP libraries and the parametric models) are made global, therefore accessible for all the workers in the pool. In the meantime, we spawn as many workers as possible to squeeze all the computing power from the architecture.

In future extensions of the library we will investigate more in parallelism and speed-up of the sampling. We are also considering to implement our own specialised sampler and to test compiled sampling interfaces, that could possibly provide more control on the memory management as well as on the parallel exploitation of CPUs.

Appendix A.1.1. Ordering of the SSP tables and computation of the intrinsic stellar luminosity

A frequently overlooked aspect in scientific software development is the process behind RAM usage and, specifically, the way chunks of data loaded in the volatile memory reach the CPU for usage. To simplify, sequential data is cached on a hierarchy of memory slots with given size. The hierarchy ladder is set by the physical proximity of the memory slots to the CPU performing the computation, the closer the higher. CPUs can access directly only the highest levels of this hierarchy, called registers, which can host a small number of bytes (typically, the amount corresponding to a few floating point numbers).

Registers tend to remain full-filled all the time, meaning that if the CPU needs a number from memory which is not already in one of the registers, this must firstly be emptied and then filled with the number required together with all the numbers which are close to it in memory, until complete occupation. This process takes also place for the lower levels in the hierarchy ladder, namely, caches. Tipically, modern computers own three levels of cache, L1 (tens of kBs) and L2 (hundreds of kBs) are private to each CPU in the processor, while L3 (tens of MBs) is shared between all the CPUs. Since the process of moving cached data from lower to higher levels of cache, up to the registers, is time-consuming, it is desirable that data used in logically sequential operations are also stored sequentially in memory.

This is the reason behind our custom format for storage of SSP libraries, as the operation in GalaPy that makes the most massive usage of cached data is integration of SSPs to generate CSPs (Sect. 2.2). Computing Eq. (16) requires us to perform, for each wavelength, an integral in time and an interpolation in metallicity. This can be easily approximated with a linear integration in time and a linear interpolation in metallicity. The approximated and implemented version of Eq. (16) is:

$$L_{\text{CSP}}(\lambda_i, \tau_{\text{GXY}}) = \sum_{\forall\ j>0\ |\ \tau_j \lesssim \tau_{\text{GXY}}} \frac{\tau_j - \tau_{j-1}}{2} \times$$
$$\times \left\{ \psi(\tau_j)\, P^{(1)}_{L_{\text{SSP}}}\left[\lambda_i, \tau_j, Z_\star(\tau_{\text{GXY}} - \tau_j)\right] + \right.$$
$$\left. + \psi(\tau_{j-1})\, P^{(1)}_{L_{\text{SSP}}}\left[\lambda_i, \tau_{j-1}, Z_\star(\tau_{\text{GXY}} - \tau_{j-1})\right]\right\}, \quad \text{(A.1)}$$

where λ_i is the wavelength, τ_{GXY} is the age of the galaxy, τ_j is the indexed SSP age, $\psi(\tau)$ is the SFR at given time and $P^{(1)}_{L_{\text{SSP}}}$ is the first order polynomial interpolating linearly the SSP emission between its two tabulated metallicities $Z_k \le Z_\star(\tau_{\text{GXY}} - \tau_j) \le Z_{k+1}$.

As made evident from Eq. (A.1), for each wavelength we first perform an interpolation between 2 metallicities, then sum along the time dimension. Even though it might seem that the most logical dimension to keep closest in memory is metallicity, by inspecting Fig. A.1, we can easily see this is not true. In each panel, along the x-axis, we vary the value of one of the model parameters that affect the integration of Eq. (A.1) while, along the y-axis, we show the integration time in milliseconds for the whole λ-grid. Boxes on the lower right show the fixed value of the two non-varying parameters. Each different colour marks the performance of a different ordering of the 3-dimensional matrix storing the SSP library, as encoded in the Figure's legend, where the shaded regions show fluctuations over ten runs and the solid line marks the mean execution time. It is clear that the most efficient ordering is $[Z\ \lambda\ \tau]$ (in purple). The reason for this is found in the slow variation of the Z-dimension as a function of galaxy age, which means that the metallicity of SSPs in the highest cache levels is updated rarely.

SSP tables are objects counting some millions of double precision floating point numbers and their transposition can easily slow down the code. For this reason having the SSPs directly stored with the $[Z\ \lambda\ \tau]$ ordering allows to accelerate the process of building objects that depend on them (i.e. the class `galapy.CompositeStellarPopulation.CSP` provides the most direct user interface to these functionalities). Nonetheless, we provide functions in GalaPy for converting eventual user-defined SSP tables into the format described above, to foster extensibility and customisation.

Appendix A.1.2. Interpolation technique

Interpolation is used for many different purposes in GalaPy: from the computation of SSP emission between the tabulated values of metallicity to the addition of templated emission on the wavelength grid. While for some of these cases the values over which to interpolate change with the variation of the model free parameters, for the majority of the occurrences, the interpolation grid is fixed for all the parameter-space sampling[16]. We have developed an `interpolator` object exploiting this condition to speed up the computation.

The interface is optimised for computing interpolated values on 1-dimensional grids with un-evenly spaced values. This is achieved with a high level of specialisation for the functionalities, making therefore the software tool not flexible but extremely efficient when used for all the problem sizes coming up in GalaPy. This results in a smaller efficiency when building the object itself but, since this operation will be done only once for each galaxy object built, we can safely give up on it.

The `interpolator` object is based on an interval binary search tree (IBST) without overlapping. This data structure provides access to nodes that perform analytic linear interpolation, integration and derivation on a single interval of the grid with $\log N$ time scaling, where N is the size of the grid. The `find` function of the IBST implemented in the core C++ sector of GalaPy is an order of magnitude faster than its C++ STL equivalent (i.e. `std::map::find`). The interface to the `interpolator` available from GalaPy's Python API is up to two orders of magnitude faster than NumPy's linear interpolation module (i.e. `scipy.interpolate.interp1d` class with `kind='linear'` which is equivalent to the NumPy function `numpy.interp`) on problem sizes comparable to those of interest for our library. It has to be stressed that `interpolator` objects from GalaPy are not universally more efficient than equivalent functions and classes from external, wide spread and powerful packages such as SciPy and NumPy. We reach better performances only when the resolution of the interpolation grid (order of 10^3 points) and the number of interpolated points (order of $< 10^2$) is comparable to those arising from the computation of GalaPy models. Our implementation comes with the additional advantage of being available on both the C++ and the Python sectors as well as providing a uniformed interface for interpolation, numerical integration and numerical derivation.

Appendix A.2. Python API structure

A complete description of the classes and functions implemented in the GalaPy package is available in the on-line documentation, in the section Python API. We hereby provide just a short description of the package structure and of the functionalities provided by each module/sub-package.

GalaPy contains modules in the top-level package and on sub-packages as well, divided as follows

```
galapy
    |-- galapy.analysis
    |-- galapy.configuration
```

[16] E.g. when assuming an interpolated SFH model, see Sect. 2.1.2.

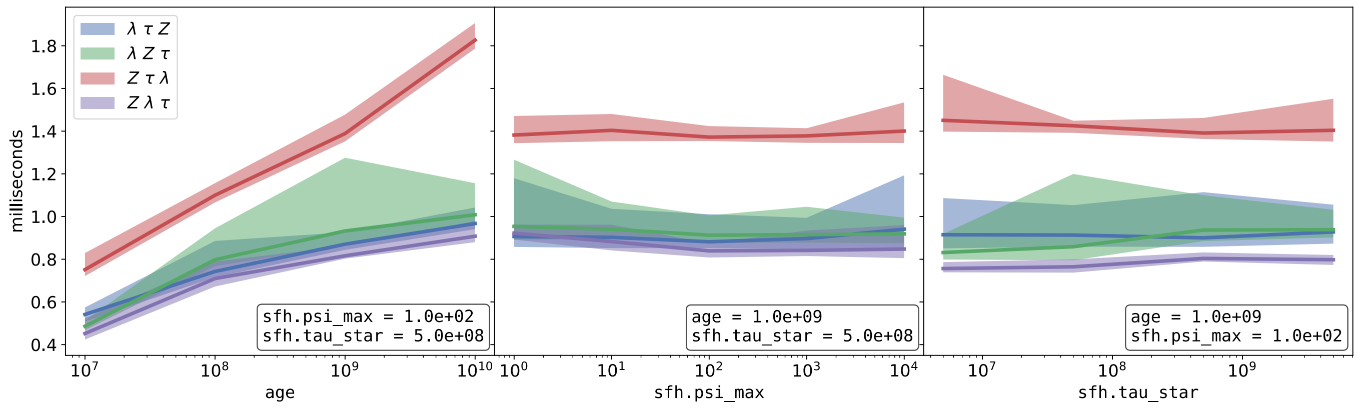

Fig. A.1. Dependency of SSP integration performances on the 3-dimensional matrix ordering. Different colours mark different orderings, as reported in the legend. Each panel shows how the integration time changes as a function of one of the three model parameters on which Eq. (A.1) depends.

```
|-- galapy.internal
|-- galapy.io
`-- galapy.sampling
```

The content of each sub-package provides different functionalities:

- **`galapy`**: the root package contains all the modules providing access to the models described in Sect. 2. Specifically:
  - `galapy.StarFormationHistory`: contains the class `SFH` that can be used to build either empirical, In situ or the interpolated SFH models of Sect. 2.1.
  - `galapy.CompositeStellarPopulation`: contains functions for listing and loading SSP libraries and the `CSP` class, used to build composite stellar populations (Sect. 2.2).
  - `galapy.InterStellarMedium`: provides access to the dust-model described in Sect. 2.3. It defines several objects: a base `ismPhase` class, from which two derived classes inherit, `MC` and `DD`, modelling the attenuation and emission due to the two separate dust components; additionally, a `ISM` type is defined, wrapping the other two components and combining their contributions.
  - `galapy.NebularFreeFree`, `galapy.Synchrotron` and `galapy.XRayBinaries`: these modules implement (optional) the additional sources of stellar continuum described in Sect. 2.4. They respectively define the classes `NFF`, `SNSYN`, and `XRB`.
  - `galapy.ActiveGalacticNucleus`: provides an interface for loading the Fritz et al. (2006) templates and consistently adding on top of them an eventual X-Ray contribution, as described in Sect. 2.5. These functionalities are accessed through the class `AGN`, defined in this module.
  - `galapy.Cosmology` and `galapy.InterGalacticMedium`, define respectively the classes `CSM` and `IGM`, whose implementation provide access to the models of Sect. 2.6.1 and Sect. 2.6.2, respectively.
  - `galapy.Galaxy` and `galapy.Handlers` define utility classes and methods designed to ease modelling through the Python API. In particular, the former defines the class `GXY` which wraps up the models implemented in the other modules, their interplay and parameter settings, optimising the performances through a minimisation of the number of operations. By instantiating one of this objects, i.e.

    ```
    from galapy.Galaxy import GXY
    gxy = GXY( age = 1.e9, redshift = 1.0 )
    ```

    users can easily modify the value of the free-parameters (e.g. `gxy.set_parameters(age = 1.e10)`, see Table B.3 for a list of all the tunable parameters), get the emission or flux (e.g. `flux = gxy.SED()`), or compute derived parameters (e.g. `Mstar = gxy.sfh.Mstar( 1.e8 )`, for the stellar mass at an age of $\tau = 10^8$ yr). We note that all these functionalities are obtained by a combination of tools implemented in the modules listed above. The latter, `galapy.Handlers` module, is designed for managing the free-parameters when sampling.
  - `galapy.PhotometricSystem`: implements the class `PMS` that can be used to manage band-pass transmission filters, both loaded from the database or user-defined.

- **`galapy.sampling`**: contains sub-modules used for sampling the parameter space, i.e. the two sub-modules `Sampler` and `Results` which unify the interface to the different sampling algorithms implemented in the library along with their results (note in particular, the `Results` class described in Sect. 3.3), the `Observation` module which collects observational datasets and the `Statistics` sub-module, defining statistical functions such as likelihoods and estimators.

- **`galapy.analysis`**: provides the two sub-modules `funcs` and `plot`, both defining functions that facilitate the analysis of sampling results. While the former mainly produces tables with the estimates of several statistics (most of the tables in Sect. 4 have been produced with these functions), the latter produces plots of fitted SEDs, residuals and posteriors (most of the figures in Sect. 4 have been produced with these functions).

- **`galapy.io`**: used to load and store object types defined in the package.

- **`galapy.configuration`** and **`galapy.internal`**: are mainly for internal usage, even though some classes and functions of `galapy.internal` might be useful in some parts of the analysis. An example are the `interpolator` objects described in Appendix A.1.2.

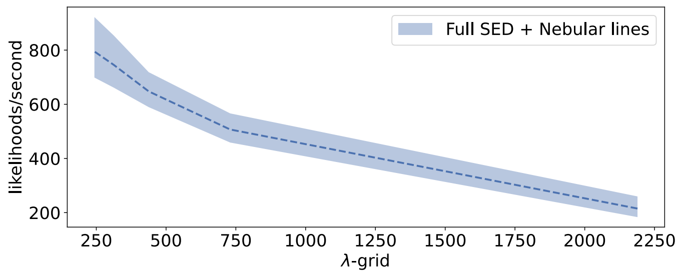

Fig. A.2. Likelihoods generated per second with GalaPy as a function of the wavelength grid resolution. The dashed line marks the average over 1000 measurements and the shaded region highlights the 1-σ confidence intervals.

Appendix A.3. Insights on performances and scaling

A solid comparison of performances against other libraries would require a thorough analysis that goes beyond the scope of this presentation work. We just mention that the computation of a single model (including parameters setting, computation of the flux and band-averaging) requires $\sim$ 10 milliseconds, depending on the resolution of the wavelength grid over which the flux is computed.

In our Bayesian framework, the time required for convergence of the free-parameters inference algorithms strongly depends on the likelihood estimation time (defined as the time necessary to set a new position in the parameter-space, compute the corresponding flux model, extract the band-averaged fluxes and compute the likelihood). We therefore measure our performances in terms of likelihood computations per second, even though, for how the code is structured and given the simplicity of the likelihood of Eq. (61) and Eq. (66), this interval of time is obviously dominated by model computation. This is shown in Fig. A.2 as a function of the wavelength grid thinness for the least optimal model set-up: BC03 SSP libraries with full modelling from X-ray band to radio including nebular and synchrotron emission. As the Figure shows, the performance of the code decreases with wavelength grid size increasing. This is expected as, the thinner the wavelength grid, the larger the number of times the code has to compute, for instance: Eq. (A.1).

As a term of comparison, on a similar problem set-up the Prospector (Johnson et al. 2021) documentation[17] declares 25 likelihoods per second, to be compared with our 200 $\div$ 300 result shown in Fig. A.2. Additionally and differently from other libraries, the problem size seems not to affect too much the computation of likelihoods as we do not measure significant variations on performances when increasing the number of photometric bands, or when making the model more complex. This is mostly due to the highly optimised implementation of GalaPy. The execution time of most of the components is in fact negligible with respect to the computation of Eq. (A.1) whose scaling also affects how the likelihood-per-second execution time scales.

We point out that the measurements provided in this Section are obtained by running on a single core, as the parallelisation scheme of GalaPy is still under development and will be in its final form on future extensions focussed on boosting the performances. At current state, we exploit the parallel strategy already implemented in the samplers available in GalaPy (i.e. `emcee` and `dynesty`) by passing to the sampling algorithm a pool of processes obtained with the `multiprocessing.Pool` method of the Python standard library. This approach may prove not to be optimal in some cases and we will therefore explore different strategies in the future.

[17] Specifically at this link: `prospect.readthedocs.io/faq` section "How long will it take to fit my data" version `v1.2.0`.

Appendix B: Additional modelling information

Here, we provide an extension on the description of models reviewed in Sect. 2 with further details.

Appendix B.1. Difference between CSP emission assuming different SSP libraries

Stellar emission, as already explained in Sect. 2.2, is computed by assuming a SFH and integrating SSP emission. In GalaPy we provide the tabulated emission from 4 main SSP libraries: the classic Bruzual & Charlot (2003) libraries in both the low (`bc03.basel`) and high (`bc03.stelib`) resolution version, and two libraries, produced specifically for the publication of this package, obtained using the PARSEC code (Bressan et al. 2012; Chen et al. 2014). The latter are delivered in two flavours:

– `parsec22.nt` contains SSP continuum emission including the non-thermal low energy contribution produced by SN synchrotron;
– `parsec22.ntl` also accounts for the thermal and line emission produced in ionised nebular regions around young massive stars.

In Table B.1, we summarise the size of the aforementioned SSP tables along each of the 3 dimensions: wavelength, age and metallicity. In BC03 tables the time domain spans from 0 to 2×10^{10} yr while PARSEC22 tables go from 0 to 1.4×10^{10} yr. In both the libraries we have extended the wavelength domain from 1 to 10^{10} Å.

The emission predicted by the models tabulated in the 4 libraries do agree in general even though they show minor differences. To highlight how choosing one library over the other contributes differently to the panchromatic emission from a galaxy, we compute the total spectrum due to the composite stellar emission. Fig. B.1 shows the ratio between CSP emission predicted with the `bc03.basel` library (left panel) and with the `parsec22.nt` library (right panel) with respect to the `parsec22.ntl` library. To have a meaningful comparison, we also include the radio components in the plot, that therefore spans from 100 Å to 10^{10} Å. This means that, while CSPs built with `parsec22.ntl` self-consistently contain the contribution from nebular thermal emission and SN synchrotron non thermal emission, in the other two cases the latter components are added to the final CSP emission using models described in Sect. 2.4.1 and Sect. 2.4.2. Fig. B.1 shows how the `parsec22.ntl` library steals energy from the continuum at the shorter wavelengths (Optical to UV) and re-emits it in lines mainly in the IR region of the spectrum. This effect is more relevant for younger galaxies (blue and green solid lines) while becoming less important to irrelevant for older stellar populations (red solid line). In the radio bands the emission predicted by the models of Sect. 2.4.1 and Sect. 2.4.2 are in good agreement with the one obtained using the PARSEC library, even though it seems to deviate more for old stellar populations in the left panel, suggesting that the synchrotron emission might be slightly over-estimated with the model of Sect. 2.4.2. In general we suggest, whenever possible, to use the PARSEC22 libraries as it also provide the additional advantage of reducing the size of the parameter space.

For reference, we provide the values, entering Eq. (46), computed for the 7 metallicities available in the BC03 SSP

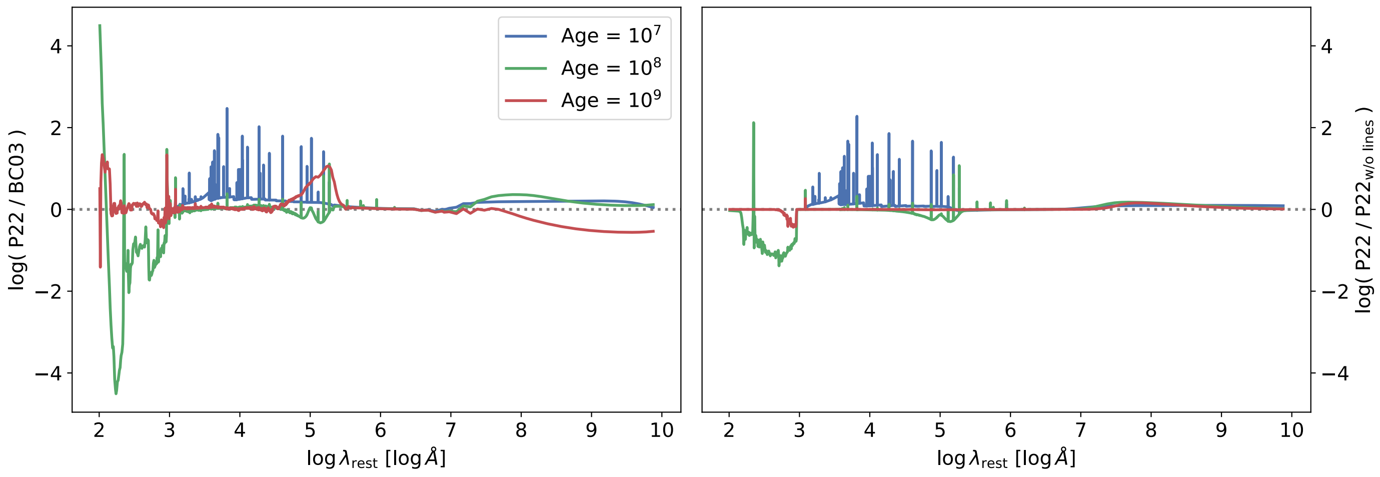

Fig. B.1. Logarithm of the ratio between CSPs obtained with different SSP libraries at fixed SFH and age. The left panel shows the ratio between PARSEC22 and BC03 while the right panel shows the ratio between PARSEC22 with and without lines.

Table B.1: Main properties of the different SSP tables delivered with GalaPy.

Library	N_λ	N_τ	N_Z	Metallicities
BC03				
Stelib	7325	221	7	[0.0001, 0.0004, 0.004, 0.008, 0.02, 0.05, 0.1]
BaSeL	2223			
PARSEC22				
NT	1562	146	6	[0.0001, 0.0005, 0.001, 0.004, 0.008, 0.02]
NTL				

Notes. The first column reports the library name, columns from the second to the fourth list the dimensions of the grids in the wavelength, age and metallicity domains, respectively. The last column provides a list of the tabulated metallicities expressed in absolute units.

libraries. The two metallicity-dependent parameters, R_0 and R_1, are tabulated in Table B.2.

Appendix B.2. Tunable parameters

In Table B.3 we provide a complete list of the parameters that can be tuned with GalaPy. All of the parameters are available from the `galapy.Galaxy.GXY` object (as well as from derived objects). We divided the table in sections describing which of the class-objects they model. The first column contains the API keyword used to access the parameter, the second column contains the symbol used in this manuscript to refer that parameter, in the third column we give a brief description of the parameter and the last column contains the eventual Eq.(s) where the parameter appears.

Each of the tunable parameters can be either fixed or set as free. In the latter case it will add a dimension to the parameter space explored by the sampler during SED-fitting. Always remember that, the larger the parameter space (both in terms of prior volume and dimensionality), the longer it will take for the sampler to converge. It is therefore always crucial to carefully select which parameters to sample and which to keep fixed. The volume, prior shape and fixed value used for each run depends on several considerations on the dataset that has to be fitted with GalaPy. The choice of these hyper-parameters is left to the user.

The parameters regulating the shape of the AGN template are not described in this manuscript and can be found in Fritz et al. (2006). Since in the current version of GalaPy we do not provide a template fitting interface yet, we discourage setting them as free, as it would imply sampling a discrete parameter space. Nonetheless, this custom behaviour can be achieved by modifying the likelihood in the sampling algorithm through the Python API of GalaPy.

Table B.2: Tabulated values of the two parameters R_0 and R_1 regulating the rate of CCSN in Eq. (46) with metallicity dependence.

$Z_\star$	0.02	0.008	0.004	0.001	0.0005	0.0001
R_0	1.5141	1.2679	1.1820	1.0924	1.0692	1.0425
R_1	0.5146	0.4454	0.4140	0.3789	0.3673	0.3531

Notes. Metallicity is given in absolute value, parameter R_0 is given in units of [Gyr$^{-1}M_\odot^{-1}$] while parameter R_1 is adimensional.

Table B.3: Complete list of the tunable parameters available in GalaPy.

Keyword	Symbol in text	Description	Section ref.
		Global	
age	τ	Age of the galaxy	2
redshift	z	Redshift of the galaxy	2.6.1
		Star Formation History	
sfh.model	—	SFH model: one among the keywords listed below	2.1
sfh.tau_quench	τ_{quench}	Age of the abrupt quenching	2.1
		Constant (keyword: `constant`)	
sfh.psi	ψ_0	Value of the constant SFR	2.1
sfh.Mdust	M_{dust}	Total dust mass in galaxy at the given age	2.1.1
sfh.Zgxy	Z_{gxy}	Metallicity of all phases in galaxy at the given age	2.1.1
		Delayed Exponential (keyword: `delayedexp`)	
sfh.psi_norm	ψ_{norm}	Normalisation	2.1
sfh.k_shape	κ	Shape parameter of the early evolution	2.1
sfh.tau_star	$\tau_\star$	Characteristic timescale	2.1
sfh.Mdust	M_{dust}	[same as for constant SFH]	2.1.1
sfh.Zgxy	Z_{gxy}	[same as for constant SFH]	2.1.1
		Log-Normal (keyword: `lognormal`)	
sfh.psi_norm	ψ_{norm}	Normalisation	2.1
sfh.sigma_star	$\sigma_\star$	Characteristic width	2.1
sfh.tau_star	$\tau_\star$	Peak age	2.1
sfh.Mdust	M_{dust}	[same as for constant SFH]	2.1.1
sfh.Zgxy	Z_{gxy}	[same as for constant SFH]	2.1.1
		Interpolated (keyword: `interpolated`)	
sfh.Mdust	M_{dust}	[same as for constant SFH]	2.1.1
sfh.Zgxy	Z_{gxy}	[same as for constant SFH]	2.1.1
		In situ (keyword: `insitu`)	
sfh.psi_max	ψ_{max}	Normalisation	2.1
sfh.tau_star	$\tau_\star$	Characteristic timescale	2.1
		Inter-Stellar Medium	
ism.f_MC	f_{MC}	Fraction of dust in the MC phase	2.3
		Molecular Clouds	
ism.norm_MC	C_V^{MC}	Normalisation of the MC extinction in the visible band	2.3
ism.N_MC	N_{MC}	Number of MCs in the galaxy	2.3
ism.R_MC	R_{MC}	Average radius of a MC	2.3
ism.tau_esc	τ_{esc}	Time required by stars to start escaping their MC	2.3
ism.dMClow	$\delta_{\text{MC}}^{\text{l}}$	Extinction power-law index at wavelength $\lesssim 100\mu m$ (10^6Å)	2.3
ism.dMCupp	$\delta_{\text{MC}}^{\text{u}}$	Extinction power-law index at wavelength $\gtrsim 100\mu m$ (10^6Å)	2.3
		Diffuse Dust	
ism.norm_DD	C_V^{DD}	Normalisation of the DD extinction in the visible band	2.3
ism.Rdust	R_{DD}	Radius of the diffuse dust region embedding stars and MCs	2.3
ism.f_PAH	f_{PAH}	Fraction of the total DD luminosity radiated by PAH	2.3
ism.dDDlow	$\delta_{\text{DD}}^{\text{l}}$	Extinction power-law index at wavelength $\lesssim 100\mu m$ (10^6Å)	2.3
ism.dDDupp	$\delta_{\text{DD}}^{\text{u}}$	Extinction power-law index at wavelength $\gtrsim 100\mu m$ (10^6Å)	2.3
		Nebular Free-Free	
nff.Zi	Z_i	Average atomic number of ions	2.4.1
		Synchrotron	
syn.alpha_syn	α_{syn}	Spectral index	2.4.2
syn.nu_self_syn	ν_{self}	Self-absorption frequency	2.4.2

Table B.3: continued.

Keyword	Symbol in text	Description	Section ref.
Active Galactic Nucleus			
`agn.fAGN`	f_{AGN}	AGN fraction	2.5
Templates			
`agn.ct`	Θ	Torus half-aperture angle	—
`agn.al`	α	Density parameter (exponential part)	—
`agn.be`	β	Density parameter (power-law part)	—
`agn.ta`	$\tau_{9.7}^{\mathrm{AGN}}$	Optical depth at 9.7μm	—
`agn.rm`	$R_{\mathrm{torus}}^{\mathrm{AGN}}$	Radial ratio of the torus	—
`agn.ia`	$\Psi_{\mathrm{los}}^{\mathrm{AGN}}$	Inclination angle	—

Notes. Each of the parameters in this Table (except for `sfh.model` and the Templates parameters of the Active Galactic Nucleus module) can be set as an additional free dimension for the sampler to explore.

Appendix C: Practical information

Appendix C.1. Installation and post-installation operations

GalaPy is available on the Python Package Index (PyPI) and can be installed by running

```
$ pip install galapy-fit
```

on a terminal[18]. Once the package has been installed, before the first usage run

```
$ galapy-download-database
```

to download on the file-system the database necessary for running. The database contains the formatted SSP libraries (Section 2.2), a collection of bandpass transmission filters (Sect. 2.6.3), the AGN templates (Sect. 2.5) and pre-computed tables for cosmological calculations (Sect. 2.6.1).

These two steps should be sufficient to obtain a working installation of GalaPy. For further details on how to install in developer mode and on the required dependencies of the library, the user can refer to the installation guide available in the documentation (`galapy.readthedocs.io/en/latest/general/install_guide.html`).

Appendix C.2. Running the automatised sampling script

We provide command-line tools for sampling the parameter space with our automatised set-up. The automatic fitting requires us to set up a parameter file that can be generated by calling

```
$ galapy-genparams [-n/--name NAME]
```

If the optional argument `NAME` is not provided, the generated file will be assigned the default name `galapy_hyper_parameters.py`. By modifying this self-explanatory file the user can set all the relevant information (e.g. free-parameters and priors, data-set, input/output files) required for running a sampling. The file is divided in four main sections:

- **Loading of the data-set**: users can use their preferred method for loading the data-set (band-pass transmission filters, either from the data-base or custom, fluxes, errors and upper-limits). Since the parameter file is effectively a python script, external libraries (such as NumPy or Pandas) can be imported to ease the process. We also provide a function (i.e. `galapy.internal.utils.cat_to_dict`) for the conversion of ASCII catalogues[19] to dictionaries.
- **Galaxy model set-up**: here the user chooses their preferred models among those available in the package. Namely, the SFH model, the SSP library, whether to include an AGN, X-ray emission, radio support, cosmology and an eventual treatment of noise.
- **Sampling parameters**: for choosing priors on the free parameters or fixing part of them to values different from their default values.
- **Sampler and output choices**: choose the sampler and format for the sampling output file(s).

Once the parameter file has been set, by calling the command

```
$ galapy-fit galapy_hyper_parameters.py
```

[18] GalaPy can also be installed from source by cloning the github repository: `github.com/TommasoRonconi/galapy`

[19] Like those compiled with TopCat `https://www.star.bris.ac.uk/ mbt/topcat/`

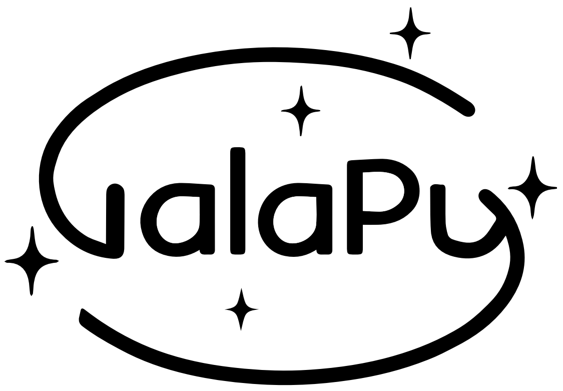

Fig. C.1. Version `1.0.0` of the GalaPy logo. We encourage authors presenting results obtained with GalaPy to add their preferred version of the logo in public presentations. A variety of formats is available on the website and in the data-base.

the sampling starts on all the available parallel CPUs (see the documentation for further details on how to customize the parallel scheme or to run serially on a single CPU).

Besides this terminal entry-points, the GalaPy API is made available upon installation of the library. On a python script, shell or notebook the user can import modules, classes and functions from the `galapy` Python package. The entry-point themselves are made available for inspection and customisation on the sub-module `galapy.sampling.Run`. For more in-detail description, please inspect the API documentation[20].

Appendix C.3. Acknowledging usage of the library

GalaPy relies on models and samplers that might require additional references along with this manuscript. We encourage authors to check the documentation for further instructions on how to acknowledge the relevant works.

Several data-formats for the GalaPy logo (Fig. C.1) are also available in the documentation. Even though we do not imprint the logo on the figures produced by our plotting API, we encourage authors in adding it to their presentations if presenting results obtained using GalaPy.

[20] `galapy.readthedocs.io/en/latest/index.html`