Draft version January 4, 2020
Typeset using LaTeX default style in AASTeX62

EinsteinPy: Python for General Relativity*

Shreyas Bapat,[1] Bhavya Bhatt,[1] Ritwik Saha,[1] and Priyanshu Khandelwal[1]
(EinsteinPy Core Development Committee)
Sofía Ortín Vela,[2] Jialin Ma,[3] Varun Singh,[1] Shilpi Jain,[4] Alpesh Jamgade,[5] Manvi Gupta,[1] Tushar Tyagi,[1]
Tanmay Rustagi,[1] Abhijeet Manhas,[1] Raphael Reyna,[6] and Sashank Mishra[7]
(EinsteinPy Package Contributors)

[1] *School of Computing and Electrical Engineering, Indian Institute of Technology Mandi, India*
[2] *Department of Theoretical Physics, University of Zaragoza, Spain*
[3] *Georgia Institute of Technology, USA*
[4] *Department of Earth Sciences, Indian Institute of Technology Roorkee, India*
[5] *Department of Mathematics, Bharath Institute of Higher Education and Research, Chennai, India*
[6] *California State Polytechnic University, Pomona*
[7] *Department of Information Technology, Indian Institute of Information Technology Allahabad*

(Received January 1, 2018; Revised January 7, 2018; Accepted January 4, 2020)

Submitted to ApJ

ABSTRACT

This paper presents EinsteinPy (version 0.2), a community-developed Python package for gravitational and relativistic astrophysics. Python is a free, easy to use high level programming language which has seen a huge expansion in the number of its users and developers in recent years. Specifically, a lot of recent studies show that the use of Python in Astrophysics and in general physics has increased exponentially. Many great frameworks came as Python packages which provide a very high level of abstraction over the dirty nitty-gritty of complex algorithms and provide an easy to use interfaceand pleasing user experience. One such example is Keras - framework for deep learning which has made deep learning so easy that a person with zero programming knowledge can also train a neural network classifier. This example really demonstrates the power of abstraction which is achievable in Python. The aim of the EinsteinPy is no different and is developed keeping in mind the state of a theoretical gravitational physicist with a little or no background in computer programming and trying to work in the field of numerical relativity or trying to use simulations in their research. Currently EinsteinPy supports simulation of time-like and null geodesics and calculate trajectories in different background geometries some of which are Schwarzschild, Kerr and KerrNewmann along with coordinate inter-conversion pipeline. It has a partially developed pipeline for plotting and visualization with dependencies on libraries like plotly, matplotlib etc. One of the unique feature of EinsteinPy is a sufficiently developed symbolic tensor manipulation utilites which is a great tool in itself for teaching yourself tensor algebra which for many beginner students can be overwhelmingly tricky. Currently EinsteinPy also provide few utility functions for hypersurface embedding of Schwarzschild spacetime which further will be extended to model gravitational lensing simulation. The current version of the library is in a state that can be used by any serious student of general relativity trying to get essence of this beautiful subject but is somewhere lost in the heavy mathematical formalism of the subject. EinsteinPy provides such students to really see through the equations and visualize whats really happening.

Keywords: gravitational physics, astrophysics — simulations — black holes — gravitational waves

* Released on July, 15th, 2019

1. INTRODUCTION

The relativistic theory of gravity paved its way to the ground in 1915, when Einstein published his paper on the general theory of relativity. This elegant and rigorous framework was a generalized version of gravity-free theory - special relativity, which he published earlier in 1905 and since then the whole physics community was against the bold ideas of the young genius and resisted the same in the beginning, as all people were worried that if the hypothesis stood true, complete notion of how they perceived space and time would get altered. Sooner, they started to realize and conceive the true depth in the formalism of general relativity and its ability to explain the fundamental laws of nature. It took a huge span of time for the acceptability of general theory of relativity as the "theory of gravity" in the classical regime. All the attempts to formulate a quantum theory of gravity in some way or the other borrow its ideas from General Relativity. Almost a hundred years after Einstein wrote down the equations of General Relativity, solutions of the Einstein equation remain extremely difficult to find beyond those which exhibit significant symmetries. The central problem then and now remains to be the solutions of Einstein's field equation. Many times we can study the behaviour of solutions under a high degree of symmetry considerations and could even solve analytically for highly unrealistic systems but solving it for problems relevant to astrophysical and gravitational physics research. It still remains a big daunting question on how to get around the problem of solving these field equations. This question is so profound that it has a separate profound field of research which goes with the name of numerical relativity which attempts to use computation science and programming to numerically obtain solutions of the equations (which would be the background geometry) for some turbulent region where most of the interesting dynamics are happening (as we can not solve for an infinitely large grid due to restrictions imposed on us by space and time complexity and computability).

Numerical methods have been used to solve the Einstein equation for many decades, but the past decade has seen tremendous advances and adding to that, the interest of the community grew when the field found applications in areas of radio astronomy, cosmology, signal processing, data mining. The field of numerical relativity is growing rampantly with the vast literature on algorithms, numerical methods and theoretical formulations (from basic 3+1 decomposition formulation to more sophisticated ones) with the development of robust and involved frameworks that provide a complete programming ecosystem and have proved to be essential tools for any numerical relativity researcher. Reflecting the other end of the research community stands the major section of theoretical physicists which counts the majority as a novice in programming. The head challenge is the fact that the usage of these frameworks demands heavy use of high-level programming languages like C, C++, Python etc. Though Python provides a vast room for abstractions no library existed that focused towards numerical relativity, eventually, the need of python library dedicated to general relativity became the propellant for the team and thus they made dauntless efforts to cover the space for this Python library. Since then, EinsteinPy has seen a lot of contributions from people all over the world and many "good to go" functionalities are already provided in this and previous versions. Like any other numerical relativity library, EinsteinPy is made to provide a set of tools which can make numerical computations for solving Einstein's field equations an easy job for anyone who does not want to deal with intricacies of the subject along with few, very basic but powerful functionalities that could be used by anyone who wants to learn the subject of general relativity with every minute detail .in general[1]. In further coming section, we discuss with some of the features of the current as well previous version and the future plans which are yet to be implemented in the upcoming versions. Also, we describe few code snippets to explain the usage of the library on which more details can be found on the organisation website[2] Software development and tools form a significant part of research in astronomy and moreover, the field itself has long supported the development of software tools for astronomical tasks.. These packages therefore typically provide some level of documentation and user support or training. . The development of the EinsteinPy core package began as a community-driven effort to standardize core functionality for software in Python which renders a user-friendly interface for supporting numerical relativity and relativistic astrophysics research. The library is as an attempt to provide programming and numerical environment for numerous numerical relativity problems like geodesics plotter, gravitational lensing and ray tracing, solving and simulating relativistic hydrodynamical equations, plotting of black hole event horizons, solving Einstein's field equations and simulating various dynamical systems like binary merger etc.

[1] More details on this are given in further coming sections
[2] https://einsteinpy.org/

At the same time, the scientific community contributing to the package has grown tremendously and an ecosystem of packages supporting or affiliated with the EinsteinPycore has developed. EinsteinPy provides an open-source and open-development core package and an ecosystem of affiliated packages that support numeric relativity functionality in the Python programming language. "Open development" describes a process where anybody with an internet connection can suggest changes to the source code and contribute their opinion when new features, bug fixes or other code changes, governance, or any other aspect of the development process is discussed. EinsteinPy core package is now a feature-rich library of sufficiently general tools and classes that support the development of more specialized code

In this paper, we describe the current status of the EinsteinPy core package and discuss goals for future development. We start by describing the way EinsteinPy functions followed by the main software efforts developed by the EinsteinPy itself with proper detailed documentation of the current version release.We end with a short vision for the future of EinsteinPy to its contribution in the field of computational methodology in gravitational and relativistic Astrophysics.

2. UNITS HANDLING

EinsteinPy has dependencies on Astropy for handling units and user have to use appropriate Astropy units[3] while declaring the input data to various methods which are further used in the computation.

3. DATA TYPES

Since its inception, EinsteinPy has aimed to provide is achieved by some set of data structures that are specifically designed to ease with a variety of gravitational physics problems. The most important and central quantity in all of the general relativity is the metric tensor $g_{\mu\nu}$ which defines the background geometry of spacetime on which all the dynamics happen. Informally the problem that packages of numerical relativity tend to solve is that we have a system (or matter field) defined by energy-momentum tensor $T_{\mu\nu}$ and we want to solve for the $g_{\mu\nu}$ such that it satisfies the following Einstein's field equation

$$R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu} + \Lambda g_{\mu\nu} = \frac{8\pi G}{c^4}T_{\mu\nu}$$

where $R_{\mu\nu}$ is contracted Riemannian tensor or Ricci tensor, R is Ricci scalar, Λ is cosmological constant. All the above quantities are derivatives of metric tensor[4] so in simple terms, on the left-hand side of the equation are terms involving $g_{\mu\nu}$ and on the right-hand side are the matter field terms which only characterize the distribution of matter. EinsteinPy provides a core module: `metric` which has core data types of various metric tensors some which include Schwarzschild, Kerr, Kerr-Newman spacetime geometries(which are the exact solution of field equation for very specific matter distributions). The immediate functionalities which come along with this module are the solutions of geodesic equations. The module has coupled integration with `coordinates` module which deals with different coordinate system transformations[5]. Another core data types are in the `plotting` module which handles all the plotting, rendering, animation and simulation related methods and provides user-friendly interfacing of plots. One interesting feature of EinsteinPy is the support for symbolic tensor algebra which is extremely important in the formalism of general relativity and one of the first challenges faced by many beginners in their journey of learning the subject. EinsteinPy provides an excellent way to learn the math behind tensor manipulation and index gymnastics with the help of symbolic module: `symbolic`, on which more will be discussed in its section ahead.

Features of EinsteinPy and section division. **Paper organisation**

Organisation and Infrastructure EinsteinPy equations and notations: The purpose of this subsection is to clearly state the equations of motion that we have implemented and our conventions for completeness

Evolution Equation

EinsteinPy core package version 0.2 Integrators

Metric

Utils

Plotting

Constant

Coordinates

[3] more on astropy units can be referred from their official website documentation

[4] more on this can be referred from any standard general relativity text

[5] more on this in the 'coordinates' section

Geodesic
Bodies
Hypersurface
symbolic Support of EinsteinPy The EinsteinPy Project, as of the version 0.2release, does not receive any direct financial support for the development of EinsteinPy. Development of the software, all supporting materials, and community support are provided by individuals who work on EinsteinPy in their own personal time, by individuals or groups contributing to EinsteinPy as part of a research project. (A list of organizations that have contributed to EinsteinPy in this manner can be found in the Acknowledgments)

Release Cycle and Long-term Support The EinsteinPy package is updated with significant changes and bug fixes in every release. The major releases contain new features or any significant changes, whereas the bugfix releases only contain fixes to code or documentation but no new features.

Following sections will give a breif overview of every core module and some of their important functionalities.

3.1. *tensor*

3.2. *metric*

`metric` module give users the freedom to define metric object which can be further used in the computations related to spacetime geometry. Let's see an small example of solving geodesic equation in Kerr spacetime and obtaining trajectories using EinsteinPy

3.3. *coordinates*

from astropy import units as u import numpy as np from einsteinpy.metric import Kerr from einsteinpy.coordinates import BoyerLindquistDifferential

3.4. *plotting*

3.5. *symbolic*

General Relativity required heavy mathematical formulation to derive and describe various tensors attributing to various features essential for cogent description of a arbitrary space-time. We would describe the various quantities currently supported by this module while maintaining brevity. Each quantity has its own class, even scalars, which are treated as 0^{th} order tensors.

3.5.1. *BaseRelativityTensor*

This is a base class from which other tensors are derived. The user can use this class to create his/her own tensor while using the various in-built functions provided by this class. The class also maintains the indicial configuration of the tensor(contravariant and co-variant indices), calculates the free variables used in describing the tensor other than those which depicts the axis of space-time itself (e.g. t,x,y,z). The class has a function to convert the symbolic tensorial quantities into functions where actual numerical values can be substitued.

This class along with its parent class contains the following functions/properties.

- `order` (Property) : Returns the order of the tensor.
- `config` (Property) : Returns the arrangement of contravariant(upper)/covariant(lower) indices of the tensor. For example, 'ulll'.
- `parent_metric` (Property) : Returns the metric tensor of the space-time associated with the tensor, if present. Else returns `None`.
- `tensor` (Function) : Returns the tensorial quantity contained within the class instance.
- `subs` (Function) : Returns a new tensor upon substitution of current variables with new variables/values.
- `simplify` (Function) : Returns a tensor contain simplified expressions from the current tensor.
- `symbols` (Function) : Returns the symbols denoting each axis of the space-time.
- `tensor_lambdify` (Function) : Returns lambdified function of symbolic tensors. This means that the returned functions can accept numerical values and return numerical quantities.

3.5.2. *MetricTensor*

Metric Tensor describes the differential length elements required to measure distance in a curved(also applies to flat) space-time. It is a second order tensor and is fundamental to describe any space-time geometry. The `MetricTensor` class, being inherited from `BaseRelativityTensor`, inherits all its functions and also have some functions and constraints of its own. For example, metric tensor is bound to be a second order tensor.
The defined class functions are :

- `change_config` : returns a different instance of the same class with different index (contravariant or covariant) configuration.
- `inv` : returns inverse of the metric

3.5.3. *ChristoffelSymbols*

Although Christoffel Symbols are belong to class of pseudo-tensors, it is inherited from `BaseRelativityTensor` to maintain homogenity within the module. Bein inherited from `BaseRelativityTensor`, it already supports the functions of its parent class and imposes some restrictions of its own, such as order of the tensor should be 3.
Instance of the class can be made by directly passing a 3$^{\text{rd}}$ order tensor or can be obtained using the following classmethods

- `from_metric` : Calculates christoffel symbols from a given instance of `MetricTensor` class.

The defined class function(s) are :

- `change_config` : returns a different instance of the same class with different index (contravariant or covariant) configuration.

3.5.4. *RiemannCurvatureTensor*

This quantity is the most extensive measure of curvature of a sapce-time. It is inherited from `BaseRelativityTensor` and supports all the functions of its parent class.
Instance of the class can be made by directly passing a 4$^{\text{th}}$ order tensor or can be obtained using the following classmethods

- `from_metric` : Calculates Riemann Tensor from a given instance of `MetricTensor` class.
- `from_christoffels` : Calculates Riemann Tensor from a given instance of `ChristoffelSymbols` class.

The defined class function(s) are :

- `change_config` : returns a different instance of the same class with different index (contravariant or covariant) configuration.

3.5.5. *RicciTensor*

This tensor is used in solving Eintein's equation and basically contraction of Riemann Curvature Tensor. It is inherited from `BaseRelativityTensor` and supports all the functions of its parent class.
Instance of the class can be made by directly passing a 2$^{\text{nd}}$ order tensor or can be obtained using the following classmethods

- `from_metric` : Calculates Ricci Tensor from a given instance of `MetricTensor` class.
- `from_christoffels` : Calculates Ricci Tensor from a given instance of `ChristoffelSymbols` class.
- `from_riemann` : Calculates Ricci Tensor from a given instance of `RiemannCurvatureTensor` class.

The defined class function(s) are :

- `change_config` : returns a different instance of the same class with different index (contravariant or covariant) configuration.

3.5.6. *RicciScalar*

It is a scalar quantity obtained by contracting a given Ricci Tensor. It is inherited from `BaseRelativityTensor` and supports all the functions of its parent class. Being a scalar quantity, it does not support `change_config` unlike other tensors as discussed above.

Instance of the class can be made by directly passing a symbolic expression or can be obtained using the following classmethods

- `from_metric` : Calculates Ricci Scalar from a given instance of `MetricTensor` class.
- `from_christoffels` : Calculates Ricci Scalar from a given instance of `ChristoffelSymbols` class.
- `from_riemann` : Calculates Ricci Scalar from a given instance of `RiemannCurvatureTensor` class.
- `from_riccitensor` : Calculates Ricci Scalar from a given instance of `RicciTensor` class.

The defined class properties are :

- `expr` : returns the scalar symbolic expression contained in the class.

3.5.7. *Other Defined Quantities*

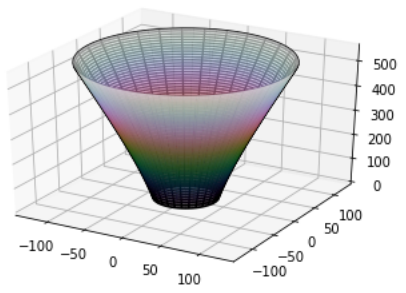
As the library is in heavy development, new quantities are being added as and when required. Some other quantities currently supported are listed below. The quantities are all inherited from `BaseRelativityTensor`, and a similar API (same as that of discussed above) has been implemented to maintain compatiblity and homegenity. The class names denote usual quantities relevant to General Relativity.

- `EinsteinTensor`
- `StressEnergyMomentumTensor`
- `WeylTensor`
- `SchoutenTensor`

Further, various constants like c(speed of light), G(Gravitational constant) are defined in their symbolic forms, for use within the symbolic module.

3.6. *hypersurface*

This module provides with basic computational functions which are essential for modelling the space-like hypersurface for any spacetime geometry. Currently module has implementation for Schwarzschild geometry with conversion functions from Schwarzschild coordinates to 3-D spherical coordinates and some plotting utilities. The whole module is expected to extend for providing a superclass for implementations related to gravitationl lensing simulations and numerical calculations of null geodesics on hypersurfaces and deviation angles for systems which are relevant to relativistic astrophysical problems. A small code snippet for plotting the hypersurface is as follows: from einsteinpy.hypersurface import SchwarzschildEmbedding from astropy import units as u



4. DISPLAYING MATHEMATICS

The most common mathematical symbols and formulas are in the amsmath package. AASTEX requires this package so there is no need to specifically call for it in the document preamble. Most modern LaTEX distributions already contain this package. If you do not have this package or the other required packages, revtex4-1, latexsym, graphicx, amssymb, longtable, and epsf, they can be obtained from http://www.ctan.org

Mathematics can be displayed either within the text, e.g. $E = mc^2$, or separate from in an equation. In order to be properly rendered, all inline math text has to be declared by surrounding the math by dollar signs ($).

A complex equation example with inline math as part of the explanation follows.

$$\bar{v}(p_2, \sigma_2) P_{-\tau} \hat{a}_1 \hat{a}_2 \cdots \hat{a}_n u(p_1, \sigma_1), \tag{1}$$

where p and σ label the initial $e^{\pm}$ four-momenta and helicities ($\sigma = \pm 1$), $\hat{a}_i = a_i^\mu \gamma_\nu$ and $P_\tau = \frac{1}{2}(1 + \tau\gamma_5)$ is a chirality projection operator ($\tau = \pm 1$). This produces a single line formula. LaTEX will auto-number this and any subsequent equations. If no number is desired then the `equation` call should be replaced with `displaymath`.

LaTEX can also handle a a multi-line equation. Use `eqnarray` for more than one line and end each line with a \\. Each line will be numbered unless the \\ is preceded by a `\nonumber` command. Alignment points can be added with ampersands (&). There should be two ampersands per line. In the examples they are centered on the equal symbol.

$$\gamma^\mu = \begin{pmatrix} 0 & \sigma_+^\mu \\ \sigma_-^\mu & 0 \end{pmatrix}, \gamma^5 = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{2}$$

$$\sigma_\pm^\mu = (\mathbf{1}, \pm\sigma), \tag{3}$$

$$\begin{aligned} \hat{a} &= \begin{pmatrix} 0 & (\hat{a})_+ \\ (\hat{a})_- & 0 \end{pmatrix}, \\ (\hat{a})_\pm &= a_\mu \sigma_\pm^\mu \end{aligned} \tag{4}$$

5. REVISION TRACKING AND COLOR HIGHLIGHTING

Authors sometimes use color to highlight changes to their manuscript in response to editor and referee comments. In AASTEX new commands have been introduced to make this easier and formalize the process.

The first method is through a new set of editing mark up commands that specifically identify what has been changed. These commands are `\added{<text>}`, `\deleted{<text>}`, and `\replaced{<old text>}{<replaced text>}`. To activate these commands the `trackchanges` option must be used in the `\documentclass` call. When compiled this will produce the marked text in red. The `\explain{<text>}` can be used to add text to provide information to the reader describing the change. Its output is purple italic font. To see how `\added{<important added info>}`, `\deleted{<this can be deleted text>}`, `\replaced{<old data>}{<replaced data>}`, and `\explain{<text explaining the change>}` commands will produce important added information and replaced data, toggle between versions compiled with and without the `trackchanges` option.

A summary list of all these tracking commands can be produced at the end of the article by adding the `\listofchanges` just before the `\end{document}` call. The page number for each change will be provided. If the `linenumbers` option is also included in the documentcall call then not only will all the lines in the article be numbered for handy reference but the summary list will also include the line number for each change.

The second method does not have the ability to highlight the specific nature of the changes but does allow the author to document changes over multiple revisions. The commands are `\edit1{<text>}`, `\edit2{<text>}` and `\edit3{<text>}` and they produce `<text>` that is highlighted in bold red, italic blue and underlined purple, respectively. Authors should use the first command to **indicated which text has been changed from the first revision.** The second command is to highlight *new or modified text from a second revision*. If a third revision is needed then the last command should be used to show this changed text. Since over 90% of all manuscripts are accepted after the 3rd revision these commands make it easy to identify what text has been added and when. Once the article is accepted all the highlight color can be turned off simply by adding the `\turnoffediting` command in the preamble. Likewise, the new commands `\turnoffeditone`, `\turnoffedittwo`, and `\turnoffeditthree` can be used to only turn off the `\edit1{<text>}`, `\edit2{<text>}` and `\edit3{<text>}`, respectively.

Similar to marking editing changes with the `\edit` options there are also the `\authorcomments1{<text>}`, `\authorcomments2{<text>}` and `\authorcomments3{<text>}` commands. These produce the same bold red, italic blue and underlined purple text but when the `\turnoffediting` command is present the `<text>` material does not appear in the manuscript. Authors can use these commands to mark up text that they are not sure should appear in the final manuscript or as a way to communicate comments between co-authors when writing the article.

6. SOFTWARE AND THIRD PARTY DATA REPOSITORY CITATIONS

The AAS Journals would like to encourage authors to change software and third party data repository references from the current standard of a footnote to a first class citation in the bibliography. As a bibliographic citation these important references will be more easily captured and credit will be given to the appropriate people.

The first step to making this happen is to have the data or software in a long term repository that has made these items available via a persistent identifier like a Digital Object Identifier (DOI). A list of repositories that satisfy this criteria plus each one's pros and cons are given at https://github.com/AASJournals/Tutorials/tree/master/Repositories.

In the bibliography the format for data or code follows this format:

author year, title, version, publisher, prefix:identifier

Corrales (2015) provides a example of how the citation in the article references the external code at https://doi.org/10.5281/zenod
Unfortunately, bibtex does not have specific bibtex entries for these types of references so the "@misc" type should be used. The Repository tutorial explains how to code the "@misc" type correctly. The most recent aasjournal.bst file, available with AASTEX v6, will output bibtex "@misc" type properly.

We thank all the people that have made this AASTeX what it is today. This includes but not limited to Bob Hanisch, Chris Biemesderfer, Lee Brotzman, Pierre Landau, Arthur Ogawa, Maxim Markevitch, Alexey Vikhlinin and Amy Hendrickson. Also special thanks to David Hogg and Daniel Foreman-Mackey for the new "modern" style design. Considerable help was provided via bug reports and hacks from numerous people including Patricio Cubillos, Alex Drlica-Wagner, Sean Lake, Michele Bannister, Peter Williams, and Jonathan Gagne.

Facilities: HST(STIS), Swift(XRT and UVOT), AAVSO, CTIO:1.3m, CTIO:1.5m,CXO

Software: astropy (Astropy Collaboration et al. 2013), Cloudy (Ferland et al. 2013), SExtractor (Bertin & Arnouts 1996)

APPENDIX

A. APPENDIX INFORMATION

Appendices can be broken into separate sections just like in the main text. The only difference is that each appendix section is indexed by a letter (A, B, C, etc.) instead of a number. Likewise numbered equations have the section letter appended. Here is an equation as an example.

$$I = \frac{1}{1 + d_1^{P(1+d_2)}} \tag{A1}$$

Appendix tables and figures should not be numbered like equations. Instead they should continue the sequence from the main article body.

B. AUTHOR PUBLICATION CHARGES

Finally some information about the AAS Journal's publication charges. In April 2011 the traditional way of calculating author charges based on the number of printed pages was changed. The reason for the change was due to a recognition of the growing number of article items that could not be represented in print. Now author charges are determined by a number of digital "quanta". A single quantum is 350 words, one figure, one table, and one enhanced digital item. For the latter this includes machine readable tables, figure sets, animations, and interactive figures. The current cost is $27 per word quantum and $30 for all other quantum type.

C. ROTATING TABLES

The process of rotating tables into landscape mode is slightly different in AASTeXv6.2. Instead of the `\rotate` command, a new environment has been created to handle this task. To place a single page table in a landscape mode start the table portion with `\begin{rotatetable}` and end with `\end{rotatetable}`.

Tables that exceed a print page take a slightly different environment since both rotation and long table printing are required. In these cases start with `\begin{longrotatetable}` and end with `\end{longrotatetable}`. Table 1 is an example of a multi-page, rotated table.

Table 1. Observable Characteristics of Galactic/Magellanic Cloud novae with X-ray observations

Name	V$_{max}$	Date	t$_2$	FWHM	E(B-V)	N$_H$	Period	D	Dust?	RN?
	(mag)	(JD)	(d)	(km s^{-1})	(mag)	(cm^{-2})	(d)	(kpc)		
CI Aql	8.83 (1)	2451665.5 (1)	32 (2)	2300 (3)	0.8±0.2 (4)	1.2e+22	0.62 (4)	6.25±5 (4)	N	Y
CSS081007	···	2454596.5	···	···	0.146	1.1e+21	1.77 (5)	4.45±1.95 (6)	···	···
GQ Mus	7.2 (7)	2445352.5 (7)	18 (7)	1000 (8)	0.45 (9)	3.8e+21	0.059375 (10)	4.8±1 (9)	N (7)	···
IM Nor	7.84 (11)	2452289 (2)	50 (2)	1150 (12)	0.8±0.2 (4)	8e+21	0.102 (13)	4.25±3.4 (4)	N	Y
KT Eri	5.42 (14)	2455150.17 (14)	6.6 (14)	3000 (15)	0.08 (15)	5.5e+20	···	6.5 (15)	N	M
LMC 1995	10.7 (16)	2449778.5 (16)	15±2 (17)	···	0.15 (203)	7.8e+20	···	50	···	···
LMC 2000	11.45 (18)	2451737.5 (18)	9±2 (19)	1700 (20)	0.15 (203)	7.8e+20	···	50	···	···
LMC 2005	11.5 (21)	2453700.5 (21)	63 (22)	900 (23)	0.15 (203)	1e+21	···	50	M (24)	···
LMC 2009a	10.6 (25)	2454867.5 (25)	4±1	3900 (25)	0.15 (203)	5.7e+20	1.19 (26)	50	N	Y
SMC 2005	10.4 (27)	2453588.5 (27)	···	3200 (28)	···	5e+20	···	61	···	···
QY Mus	8.1 (29)	2454739.90 (29)	60:	···	0.71 (30)	4.2e+21	···	···	M	···
RS Oph	4.5 (31)	2453779.44 (14)	7.9 (14)	3930 (31)	0.73 (32)	2.25e+21	456 (33)	1.6±0.3 (33)	N (34)	Y
U Sco	8.05 (35)	2455224.94 (35)	1.2 (36)	7600 (37)	0.2±0.1 (4)	1.2e+21	1.23056 (36)	12±2 (4)	N	Y
V1047 Cen	8.5 (38)	2453614.5 (39)	6 (40)	840 (38)	···	1.4e+22	···	···	···	···
V1065 Cen	8.2 (41)	2454123.5 (41)	11 (42)	2700 (43)	0.5±0.1 (42)	3.75e+21	···	9.05±2.8 (42)	Y (42)	···
V1187 Sco	7.4 (44)	2453220.5 (44)	7: (45)	3000 (44)	1.56 (44)	8.0e+21	···	4.9±0.5 (44)	N	···
V1188 Sco	8.7 (46)	2453577.5 (46)	7 (40)	1730 (47)	···	5.0e+21	···	7.5 (39)	···	···
V1213 Cen	8.53 (48)	2454959.5 (48)	11±2 (49)	2300 (50)	2.07 (30)	1.0e+22	···	···	···	···
V1280 Sco	3.79 (51)	2454147.65 (14)	21 (52)	640 (53)	0.36 (54)	1.6e+21	···	1.6±0.4 (54)	Y (54)	···
V1281 Sco	8.8 (55)	2454152.21 (55)	15:	1800 (56)	0.7 (57)	3.2e+21	···	···	N	···
V1309 Sco	7.1 (58)	2454714.5 (58)	23±2 (59)	670 (60)	1.2 (30)	4.0e+21	···	···	···	···
V1494 Aql	3.8 (61)	2451515.5 (61)	6.6±0.5 (61)	1200 (62)	0.6 (63)	3.6e+21	0.13467 (64)	1.6±0.1 (63)	N	···
V1663 Aql	10.5 (65)	2453531.5 (65)	17 (66)	1900 (67)	2: (68)	1.6e+22	···	8.9±3.6 (69)	N	···
V1974 Cyg	4.3 (70)	2448654.5 (70)	17 (71)	2000 (19)	0.36±0.04 (71)	2.7e+21	0.081263 (70)	1.8±0.1 (72)	N	···
V2361 Cyg	9.3 (73)	2453412.5 (73)	6 (40)	3200 (74)	1.2: (75)	7.0e+21	···	···	Y (40)	···
V2362 Cyg	7.8 (76)	2453831.5 (76)	9 (77)	1850 (78)	0.575±0.015 (79)	4.4e+21	0.06577 (80)	7.75±3 (77)	Y (81)	···
V2467 Cyg	6.7 (82)	2454176.27 (82)	7 (83)	950 (82)	1.5 (84)	1.4e+22	0.159 (85)	3.1±0.5 (86)	M (87)	···
V2468 Cyg	7.4 (88)	2454534.2 (88)	10:	1000 (88)	0.77 (89)	1.0e+22	0.242 (90)	···	N	···
V2491 Cyg	7.54 (91)	2454567.86 (91)	4.6 (92)	4860 (93)	0.43 (94)	4.7e+21	0.09580: (95)	10.5 (96)	N	M
V2487 Oph	9.5 (97)	2450979.5 (97)	6.3 (98)	10000 (98)	0.38±0.08 (98)	2.0e+21	···	27.5±3 (99)	N (100)	Y (101)
V2540 Oph	8.5 (102)	2452295.5 (102)	···	···	···	2.3e+21	0.284781 (103)	5.2±0.8 (103)	N	···
V2575 Oph	11.1 (104)	2453778.8 (104)	20:	560 (104)	1.4 (105)	3.3e+21	···	···	N (105)	···
V2576 Oph	9.2 (106)	2453832.5 (106)	8:	1470 (106)	0.25 (107)	2.6e+21	···	···	N	···
V2615 Oph	8.52 (108)	2454187.5 (108)	26.5 (108)	800 (109)	0.9 (108)	3.1e+21	···	3.7±0.2 (108)	Y (110)	···
V2670 Oph	9.9 (111)	2454613.11 (111)	15:	600 (112)	1.3: (113)	2.9e+21	···	···	N (114)	···
V2671 Oph	11.1 (115)	2454617.5 (115)	8:	1210 (116)	2.0 (117)	3.3e+21	···	···	M (117)	···

Table 1 continued on next page

Table 1 *(continued)*

Name	V$_{max}$	Date	t$_2$	FWHM	E(B-V)	N$_H$	Period	D	Dust?	RN?
	(mag)	(JD)	(d)	(km s^{-1})	(mag)	(cm^{-2})	(d)	(kpc)		
V2672 Oph	10.0 (118)	2455060.02 (118)	2.3 (119)	8000 (118)	1.6±0.1 (119)	4.0e+21	· · ·	19±2 (119)	· · ·	M
V351 Pup	6.5 (120)	2448617.5 (120)	16 (121)	· · ·	0.72±0.1 (122)	6.2e+21	0.1182 (123)	2.7±0.7 (122)	N	· · ·
V382 Nor	8.9 (124)	2453447.5 (124)	12 (40)	1850 (23)	· · ·	1.7e+22	· · ·	· · ·	· · ·	· · ·
V382 Vel	2.85 (125)	2451320.5 (125)	4.5 (126)	2400 (126)	0.05: (126)	3.4e+21	0.146126 (127)	1.68±0.3 (126)	N	· · ·
V407 Cyg	6.8 (128)	2455266.314 (128)	5.9 (129)	2760 (129)	0.5±0.05 (130)	8.8e+21	15595 (131)	2.7 (131)	· · ·	Y
V458 Vul	8.24 (132)	2454322.39 (132)	7 (133)	1750 (134)	0.6 (135)	3.6e+21	0.06812255 (136)	8.5±1.8 (133)	N (135)	· · ·
V459 Vul	7.57 (137)	2454461.5 (137)	18 (138)	910 (139)	1.0 (140)	5.5e+21	· · ·	3.65±1.35 (138)	Y (140)	· · ·
V4633 Sgr	7.8 (141)	2450895.5 (141)	19±3 (142)	1700 (143)	0.21 (142)	1.4e+21	0.125576 (144)	8.9±2.5 (142)	N	· · ·
V4643 Sgr	8.07 (145)	2451965.867 (145)	4.8 (146)	4700 (147)	1.67 (148)	1.4e+22	· · ·	3 (148)	N	· · ·
V4743 Sgr	5.0 (149)	2452537.5 (149)	9 (150)	2400 (149)	0.25 (151)	1.2e+21	0.281 (152)	3.9±0.3 (151)	N	· · ·
V4745 Sgr	7.41 (153)	2452747.5 (153)	8.6 (154)	1600 (155)	0.1 (154)	9.0e+20	0.20782 (156)	14±5 (154)	· · ·	· · ·
V476 Sct	10.3 (157)	2453643.5 (157)	15 (158)	· · ·	1.9 (158)	1.2e+22	· · ·	4±1 (158)	M (159)	· · ·
V477 Sct	9.8 (160)	2453655.5 (160)	3 (160)	2900 (161)	1.2: (162)	4e+21	· · ·	· · ·	M (163)	· · ·
V5114 Sgr	8.38 (164)	2453081.5 (164)	11 (165)	2000 (23)	· · ·	1.5e+21	· · ·	7.7±0.7 (165)	N (166)	· · ·
V5115 Sgr	7.7 (167)	2453459.5 (167)	7 (40)	1300 (168)	0.53 (169)	2.3e+21	· · ·	· · ·	N (169)	· · ·
V5116 Sgr	8.15 (170)	2453556.91 (170)	6.5 (171)	970 (172)	0.25 (173)	1.5e+21	0.1238 (171)	11±3 (173)	N (174)	· · ·
V5558 Sgr	6.53 (175)	2454291.5 (175)	125 (176)	1000 (177)	0.80 (178)	1.6e+22	· · ·	1.3±0.3 (176)	N (179)	· · ·
V5579 Sgr	5.56 (180)	2454579.62 (180)	7:	1500 (23)	1.2 (181)	3.3e+21	· · ·	· · ·	Y (181)	· · ·
V5583 Sgr	7.43 (182)	2455051.07 (182)	5:	2300 (182)	0.39 (30)	2.0e+21	· · ·	10.5	· · ·	· · ·
V574 Pup	6.93 (183)	2453332.22 (183)	13 (184)	2800 (184)	0.5±0.1	6.2e+21	· · ·	6.5±1	M (185)	· · ·
V597 Pup	7.0 (186)	2454418.75 (186)	3:	1800 (187)	0.3 (188)	5.0e+21	0.11119 (189)	· · ·	N (188)	· · ·
V598 Pup	3.46 (14)	2454257.79 (14)	9±1 (190)	· · ·	0.16 (190)	1.4e+21	· · ·	2.95±0.8 (190)	· · ·	· · ·
V679 Car	7.55 (191)	2454797.77 (191)	20:	· · ·	· · ·	1.3e+22	· · ·	· · ·	· · ·	· · ·
V723 Cas	7.1 (192)	2450069.0 (192)	263 (2)	600 (193)	0.5 (194)	2.35e+21	0.69 (195)	3.86±0.23 (196)	N	· · ·
V838 Her	5 (197)	2448340.5 (197)	2 (198)	· · ·	0.5±0.1 (198)	2.6e+21	0.2975 (199)	3±1 (198)	Y (200)	· · ·
XMMSL1 J06	12 (201)	2453643.5 (202)	8±2 (202)	· · ·	0.15 (203)	8.7e+20	· · ·	50	· · ·	· · ·

A handy "cheat sheat" that provides the necessary LaTeX to produce 17 different types of tables is available at http://journals.aas.org/authors/aastex/aasguide.html#table_cheat_sheet.

REFERENCES

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33

Bertin, E., & Arnouts, S. 1996, A&AS, 117, 393

Corrales, L. 2015, ApJ, 805, 23

Ferland, G. J., Porter, R. L., van Hoof, P. A. M., et al. 2013, RMxAA, 49, 137

Hanisch, R. J., & Biemesderfer, C. D. 1989, BAAS, 21, 780

Lamport, L. 1994, LaTeX: A Document Preparation System, 2nd Edition (Boston, Addison-Wesley Professional)

Schwarz, G. J., Ness, J.-U., Osborne, J. P., et al. 2011, ApJS, 197, 31

Vogt, F. P. A., Dopita, M. A., Kewley, L. J., et al. 2014, ApJ, 793, 127