An Interface for Future Comparison of Real and Simulated Interacting Galaxy Pair Morphologies using JSPAM

THESIS

Presented to the Faculty of the Department of Physics and Astronomy
in Partial Fulfillment of the Major Requirements
for the Degree of

BACHELOR OF SCIENCE IN
PHYSICS

Jackson L. Cole

December 2018

AN INTERFACE FOR FUTURE COMPARISON OF REAL AND SIMULATED INTERACTING GALAXY PAIR MORPHOLOGIES USING JSPAM

Jackson L. Cole

Signature of Author:

Department of Physics and Astronomy
May 2018

Certified by:

Dr. John Wallin
Professor of Physics & Astronomy
Thesis Supervisor

Accepted by:

Dr. Ronald Henderson
Professor of Physics & Astronomy
Chair, Physics & Astronomy

ABSTRACT

Restricted three-body codes are in some cases less physically revealing than their n-body counterparts in simulating interactions of galaxy pairs, but they are much faster and more computationally efficient. For this reason, restricted three-body codes are still widely used as a preliminary tool to reduce the size of the solution space of systems before proceeding to more computationally expensive n-body models, and further, can accurately simulate the large scale kinematic interactions in merging galaxies quite well. When simulating interactions of galaxy pairs, the initial conditions that result in the observed morphological features after the interaction are largely unknown, and therefore become a focus of work in the field of galactic mergers and collisions today. Citizen science efforts to systematically rank the convergence of these simulations have been carried out with good results (3), but these projects naturally rely heavily on human-in-the-loop processing. While the human eye is very successful at matching tasks based on sometimes subtle features as were important in Holincheck et al. (3), computational processes that involve humans are always subject to an upper limit on efficiency and volume of data that can be processed in a reasonable amount of time. A natural solution in modern times is to use machine learning techniques to systematically improve solutions, and is the logical next step in using JSPAM Wallin et al. (8). Regardless of the computational technique, all data-intensive tasks in astronomy and astrophysics can benefit from having a standard pipeline through which data are collected, used, and stored, as well as a standard set of methods for interacting with the software. In support of this, and co-developed with an image creation and comparison suite we provide in this work `jspamcli.py`, a user-friendly interface developed to simplify interaction with an existing restricted three-body code used in the Galaxy Zoo: Mergers project, JSPAM (8). We also restructure the project to aid in future development and add several tools that will be useful in continued efforts.

Table of Contents

List of Figures

Listings

I INTRODUCTION

French astronomer Charles Messier's Catalogue des Nébuleuses et des Amas d'Étoiles (Catalog of Nebulae and Star Clusters) contains 104 objects visible over the Parisian night sky that were frequently encountered during his efforts as a comet hunter. Although compiled in the eighteenth century and officially published from his personal notes in 1781, the positions and characterizations of objects given by Messier (5) are well-enough described that they are all easily and frequently observed today by amateur astronomers across the planet. One such object carries the following description (translated from the original French and sourced from `http://www.messier.seds.org/xtra/history/m-cat81.html`):

> It is double, each has a bright center, which are separated $4'35''$. The two "atmospheres" touch each other, the one is even fainter than the other. (5)

Of course, in the late eighteenth century, Messier would not have been predisposed to assuming that the "very faint nebula" with two touching atmospheres that he describes would be the often-imaged interacting galaxy pair, Messier 51a and b. Four years after Messier (5), German philosopher Immanuel Kant theorizes that perhaps, the exceedingly dim yet huge "stars" are not stars but seem to be most easily characterized as other Milky Ways (4). The main member of the pair is more colloquially referred to as the Whirlpool Galaxy, which is shown in Figure 1.

Galaxies are much less frequently found in isolation than they are found in systems of galaxies, which suggests that the frequency with which events leading to their interactions occur is relatively high. Further, Alladin (1) found that even in simulated collisions of relatively simple extended galaxy models, close encounters of galaxies could increase their internal energies (relating to the motions of stars) on the same order of magnitude as their respective initial internal energies. This implied that their

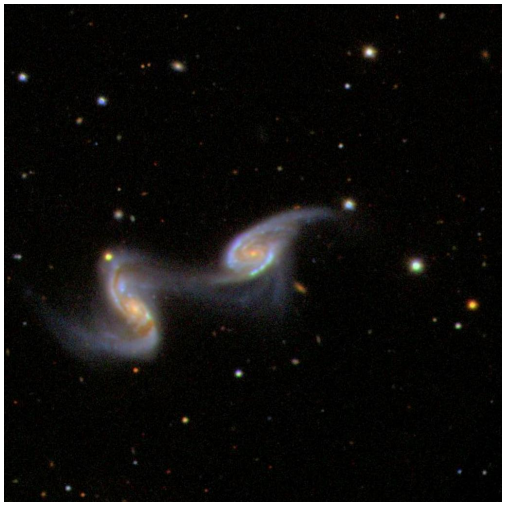
Figure 1: Composite image of Messier 51, The Whirlpool Galaxy. (Credit: X-ray: NASA/CXC/SAO; Optical: Detlef Hartmann; Infrared: NASA/JPL-Caltech)

first-order approximation of the center-of-mass motions of each the galaxies as constant was grossly inaccurate. Further results showed that as energy is exchanged from the centers-of-mass to the internal energy of the system, the structures of each galaxy could be dramatically disturbed, and in some cases could fling stars and other material into intergalactic space. The importance of collisions and mergers, therefore, is not limited in scope to the structure of interacting galaxies, but can be extended to the large-scale evolution of galaxies over the lifetime of the universe.

Nearly immediately one notices the visually striking and scientifically puzzling structures of members of an interacting system as can be seen in Figure 2. These disturbed morphological features are commonly referred to as “bridges and tails” (7). Optical and spectroscopic observations can readily yield accurate measurements of the final positions of galaxies in a merger, and occasionally the radial velocities of the members of a system can be observed as well, but the primary interest lies in the discovering the mechanisms that result in the observed bridges and tails. This necessitates finding precisely the final positions in space, the overall velocity field of system, and the angular orientations of the system with respect to our position in space. For this reason, work has been done in the last 70 years to advance simulations of interacting systems, and

further, to optimize existing simulations for better convergence on solutions. In essence, we are readily able to observe mergers how they appear now, but work must be done to infer the conditions that resulted in the morphologies we see today.

(a) Example of *undisturbed* morphology, MCG+01-02-015 (Credit: ESA/Hubble & NASA and N. Gorin (STScI))



(b) Example of *disturbed* morphology, SDSS DR7 image of 587722984435351614. This target is part of the Galaxy Zoo: Mergers data set.

Figure 2: Comparison between undisturbed and disturbed morphologies

II BACKGROUND

The primary question regarding observed disturbed morphological features of interacting galaxy pairs, described as "bridges and tails", relates to the mechanism by which they were formed. In 1972, Toomre and Toomre (7) presented some of the first widely accepted supporting evidence of what they tellingly referred to as "old-fashioned gravity" as a basis for the disturbed morphologies observed in apparently-neighboring galaxies. Pfleiderer and Siedentopf (6) provided the foundation for their work, but their results were largely rejected in the 1950s and 1960s when the complexities of galactic bridges and tails were assumed to result from equally complex mechanisms. During this period, astrophysicists were in some cases vehemently opposed to using gravitational interactions as the basis of explanation of disturbed morphological features. However, it was then shown using restricted three-body simulations of colliding galaxies that the supposed static nature of transient tidal forces between members of galaxy pairs matters less when considering the magnitude of the intensity of those tidal forces (7).

Toomre and Toomre (7) focuses on four simple examples of interacting systems, each of which serve to illustrate their proposal that the main contributors to the bridges and tails are kinematic and can be described simply in terms of gravity. In each example, they purposely initialize slow, parabolic encounters between members of the galaxy pair, as they assume that faster encounters result in thinner tidal features, and the primary goals of their work focused on proving that gravity could be one of the sole causes of the observed features. This immediately seems like an ad hoc initialization, in which one should find cause for worry that the result may have been biased towards the desired result. However, they justify their logic by making several assumptions.

A relatively high frequency of observable encounters of galaxies traveling along chance, unbound, hyperbolic orbits is highly improbable, and accordingly, observational

data from many interacting pairs point to their interactions not having been chance encounters. It is much more probable that a significant number of interacting pairs were already bound, and while still highly eccentric, their approach orbits were still sub-parabolic, meaning that their orbits could readily become bound. In work the work of Toomre and Toomre (7), a primary disk is initialized that is composed of a point mass at the center that is surrounded by several concentric rings of massless test particles. The secondary "disk" is simply a point mass representing the center of the secondary galaxy. All numerical integrations were performed separately for each particle in the primary disk, such that the computational time could be optimized for each particle in the system (7). From the work of Toomre and Toomre (7), it becomes clear that gravity alone could be a fundamental cause of the observed morphological features.

In modern times, the primary challenges involved in simulating interactions between galaxies are not very far removed from those of computational economy which were highly necessary considerations in the work of the brothers Toomre in the early 1970s, although we now at least perform the necessary integration for each particle at each time step. In many cases, simulations of galactic collisions still make use of restricted three-body codes similar to those used in the original work in the field. Restricted three-body codes only require that the gravitational force between each of the two most massive bodies in the system and a particular test mass is calculated. These codes work when the gravitational interactions between each smaller (essentially massless) test mass in the system is assumed to be negligible in comparison to those involving the larger masses. Their primary advantage over their n-body counterpart is their computational economy. These codes are also still considered valid, as the self gravity of the galactic disks is assumed to be almost negligible in comparison to the gravitational interaction of the two galactic centers and each smaller mass (7). Of course, computational economy still is a factor, but its role in guiding the methods for simulations has changed.

Today, the computational power available to researchers exceeds by far what has been available in the last six decades. Whereas in decades past, restricted three-body codes were used primarily on the basis of computational economy, we can now use these same types of codes as a still-valid method to quickly reduce the size of the solution space while the space is still populated with obviously poor solutions. Essentially, faster three-body algorithms can be used to accurately simulate more significant kinematic interactions (e.g. those between each point mass and the centers of each galactic disk), and then fitness-functions, automatic image recognition and classification via machine learning techniques, Citizen Science, or some combination of the three can be used to constrain the solution space to solutions that have a high likelihood of having reasonable convergence on the actual solution. From the constrained solution space, full n-body simulations can be used for further analysis.

The computational expense of full n-body codes is justified by when simulating the dynamical processes within the disks of galaxies, as these effects can be accurately simulated for each particle. If restricted three-body codes, such as JSPAM (8), can be used to accurately simulate the large-scale kinematic interactions with reasonable resolution, then they can be used to accurately simulate solutions across the entire solution space fairly quickly. From that point, these computationally expensive codes can be used to explore the reduced solutions space.

In short, we want to make efficient use of computational time and to reduce the size but increase the complexity of the job left for the human-in-the-loop. In recent years, the Galaxy Zoo project used what essentially was a human fitness-function comprised of Citizen Scientists to review over three million simulations (3). However, as of writing of Holincheck et al. (3), there existed no general purpose machine fitness-function for use in determining the level of convergence of solutions on observed morphologies of interacting galaxies. This work does not provide a machine fitness-function, but provides

tools that are useful in support of this effort.

This work primarily focuses on developing methods for interacting with JSPAM, a restricted three-body code originally written in FORTRAN that is described in Wallin et al. (8) that can be found at `https://github.com/JSPAM-Manga/WallinCode`. JSPAM simulates these interactions and produces data that can be used to render the simulated morphologies for eventual use in machine fitness-functions. However, a standard procedure and pipeline for operating the JSPAM code and handling the resulting data needed to be developed.

III METHODOLOGY

A Overview

The Galaxy Zoo: Mergers project produced ranked scores that are indicative of the degree to which the morphologies resulting from different sets of initial conditions correlate with their real counterparts. Essentially, the scores should relate to how closely the simulated galaxy merger looks like the real thing. These rankings are the product of a substantial Citizen Science effort comprising the Galaxy Zoo: Mergers project, and at the completion of the project, 66,395 sets of initial parameters had been scored through volunteer review of over 3 million simulations (3). While this effort was successful, moving towards a general-purpose machine scoring mechanism for these types of simulations would allow for human effort to be reserved for classifying only solutions that have a reasonable chance of being correct.

An ideal mechanism would recognize and remove morphologies that clearly do not match real life; this in contrast with the current *modus operandi*, which in some cases results in human effort being wasted on classifying a "clearly bad" solution as incorrect. If we can task a computer with making classifications that require less acute discernment, humans in-the-loop can be tasked with performing classifications that take advantage of the visual facilities of the human eye. Therefore, we operate under the assumption that an ideal machine scoring mechanism should be able to closely recreate the results of the Galaxy Zoo: Mergers Citizen Science effort, and that human volunteers could improve upon those results from that point.

In support of these efforts, and to initiate the reduction of the role of the human-in-the-loop, the primary goal of our work focuses on developing an interface and framework that can be used to aid users in interaction with JSPAM, deal with the challenges of data storage through standard methods, and aid in future work on projects using JSPAM.

Running the simulation software efficiently, storing and organizing the large volumes of data that are inherent to this project, accessing these data efficiently, image creation, and image analysis, are all intrinsic tasks in this project, and we aim to provide methods for doing all of these things effectively. We recognize that further work must be done in developing fitness-functions, but expect that our work will aid further success in doing so and should greatly reduce the overhead for continuing the work on these types of projects.

The methods described in this section will closely adhere to the order in which they appear in Figure 3, and will end with a discussion of `jspamcli.py`, which handles most of the aforementioned tasks, in subsection F.

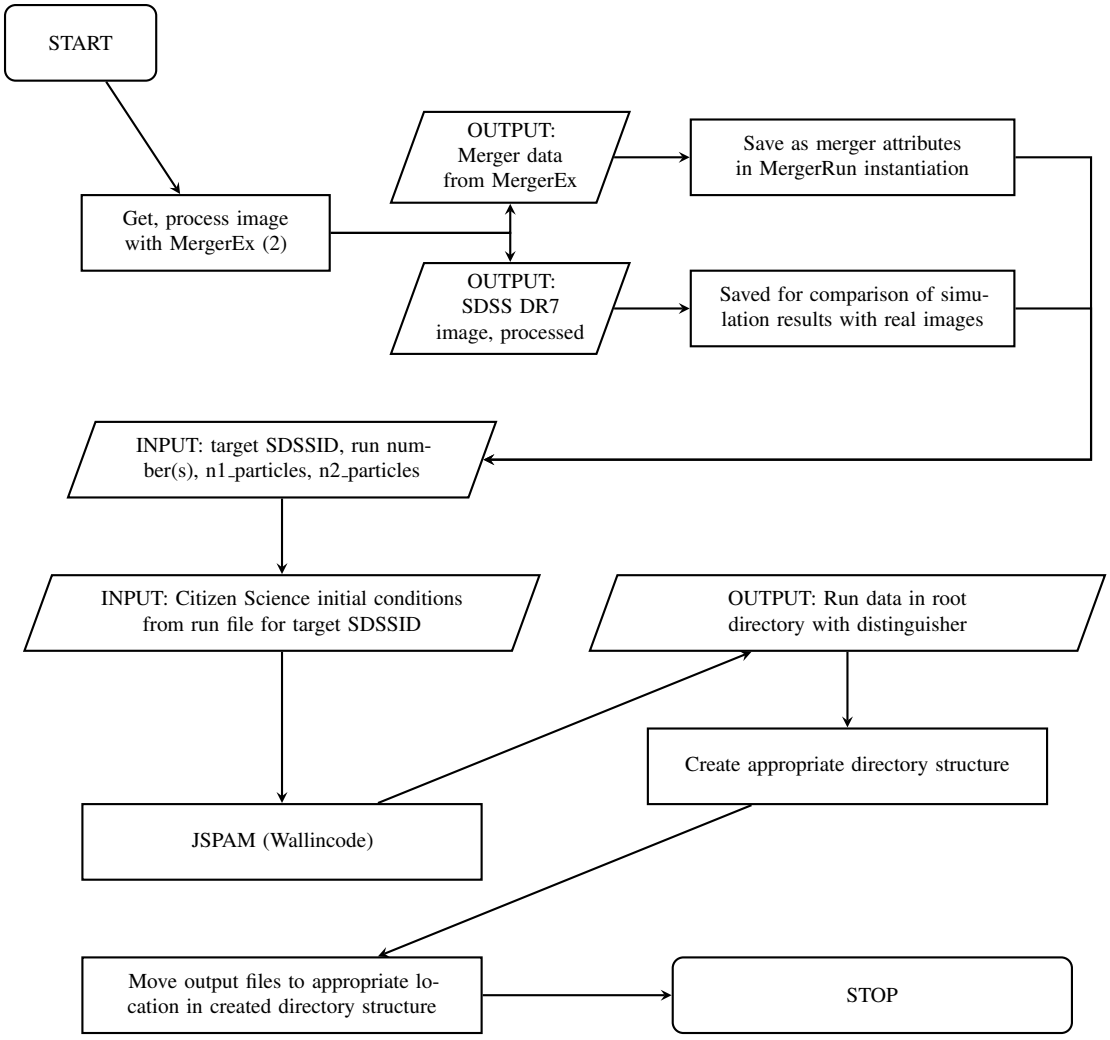


Figure 3: Flowchart describing how a run is processed and stored. It should be noted that tasks related to processing images with MergerEx must be done manually, but must only be done once. Further, the image creation suite that was developed alongside this project is not included here, as it is considered to be largely separate. It can be incorporated immediately prior to "STOP".

B Image Preparation and Target Data Acquisition

While ultimately, our primary interest is the observed morphology of some interacting galaxy pair, we need to identify a significant number of parameters relating to the sizes and orientations of the objects in the sky and in images, respectively, as well as the velocities of the primary and secondary disks. Essentially, we want to have available to us all possible data points describing the merger as it is observed and the positional data to systematically locate each object in the produced images for further analysis.

The software we use for target image preparation and for constraining the initial simulation parameter values is MergerEx, which is described in Holincheck (2) and is found at `https://github.com/aholinch/MergerEx`. MergerEx greatly simplifies and speeds up the process of querying various astronomical image servers for specific sky coordinates, calibrating the images, and estimating initial parameters that describe the sizes, orientations, and velocities of the primary and secondary galactic disks in the merger.

Because the primary data in which we are interested is only the morphology, we can rely on whichever images display the morphological features most clearly. The MergerEx software allows us to choose the most appropriate image source from either the NASA/IPAC Extragalactic Database/STScI Digital Sky Survey (NED/DSS), or Sloan Digital Sky Survey Data Release 7, 8, or 9 (SDSS DR7, DR8, DR9). For our purposes, neither the wavelengths comprising the images retrieved nor the source matter as long as the distribution of luminous stars in the system is clearly seen in the image (3). In some cases, Holincheck et al. (3) describes that these images can even come from non-scientific data sets as long as the previously mentioned condition is met.

Holincheck et al. (3) provides a list of 54 SDSS and 8 Hubble Space Telescope (HST) targets. For each of these targets, the process of estimation of the disks and

subsequently the parameter ranges must be carried out as described in Appendix C of Holincheck (2). In the project repository's `targets` directory, there now exists a Python script that allows the user to keep track of their work as they progress through all 62 targets, or through any number of targets that are defined within that directory's `all_targets.txt` file.

C Targets and Target Input Files

When run from the command line, the JSPAM software accepts a string containing estimations of initial conditions that result in the observed morphologies. Files containing the initial condition strings for each target are found at `https://data.galaxyzoo.org/mergers.html`, but in order for a new user to get started on the project, they must download one, several, or all of the files containing the thousands of sets of initial conditions. In order to simplify this process, the module `get_target_data` is included in the `merger` package that scrapes `https://data.galaxyzoo.org/mergers.html` for all of the available files to download, and then subsequently downloads the requested file or files, and unzips to the correct location. There is also an interactive feature that can be selected via an argument passed to the function. As of right now, all data are present in the project repository and were downloaded and organized using this module.

In many cases during this project, we wanted to work only with initial conditions that result in simulations that would have received a Citizen Science score, and therefore elected to remove sets of initial conditions that resulted in morphologies that were rejected in the citizen science effort. To do this, a small, ad hoc script was written that can now be found in the `archive` directory in the project repository. The input data files that contain the initial simulation parameters are located in the `input` directory (`jspamcli.py` expects this to be their location). These have been reduced in size from

the original files found at `https://data.galaxyzoo.org/mergers.html`, as they now contain only initial simulation parameters from scored runs.

D `merger` Package

One of the primary challenges that arises in astronomy or astrophysics is simply doing adequate book-keeping of data and related meta-data. This project is no different. It was immediately necessary to develop a small, purpose-built Python package to do just that. Within the `merger` package found in the project repository, we make available the `Galaxy` and `MergerRun` classes, which can be see in Figures 4 and 5, respectively. The `Galaxy` class only carries information related to the number of particles instantiated in the galaxy and the real and image positional information. `MergerRun` encapsulates all information and data relating to any particular run of JSPAM given some set of initial parameters. This class also contains methods for determining storage locations for output data, renaming files according to the appropriate naming conventions, handling the scores that are generated from fitness-functions, and even some handy plotting and visualization tools.

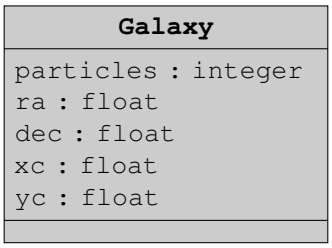


Figure 4: Class diagram: Galaxy

Figure 5: Class diagram: MergerRun

Instantiation of a `MergerRun` is intended to require the fewest number of arguments possible to reduce book-keeping. It does so by utilizing an information file written by MergerEx during image preparation, which contains a list of pertinent information. One of the member functions within the `MergerRun` class creates a dictionary from this file, from which it then initializes several of the class instance attributes. An instance of this class also requires as parameters the number of particles to be initialized in each disk, the run number, and the initial simulation parameters that are found in the input files. In general, this can be done via the line in Listing 1.

Listing 1: Instantiating `MergerRun`

```
merger_instance = MergerRun(path_to_info_file, n1_particles,
   n2_particles, run_number, init_sim_param_string)
```

Having a dedicated MergerRun class allows for improved portability of data and information relating to a particular run of a merger. We expect programming using this class to remain a bit simpler and perhaps more clear.

The MergerRun class also instantiates twice the previously mentioned `Galaxy` class; one is referred to as `primary` and the other `secondary`. We can therefore more clearly keep track of the SDSS ID or name and information relating to the disk centers, the right ascension and declination, and the Cartesian coordinates relative to the pixels in the frame.

Within this package also a small package called `data_tools` containing, as of writing, one class called `Structure`. `Structure` accepts an ordered list of directory and subdirectory names that will comprise a directory structure to be created, and further, provides a method that will check for the prior existence of the structure to prevent overwriting of data. Although this may seem unnecessary, this reduces the chance of error in organization of the large volume of simulation data that will be produced. As an example, the line in Listing 2 will create the following path in your current working directory: `root_name/child_1/child_2/child_3/`.

Listing 2: `Structure` usage

```
dir_structure = Structure(["root_name", "child_1", "child_2",
   "child_3"])
```

E Naming and Storage Conventions

We heavily explored several alternative methods for passing large volumes of data between the working languages in the project (Fortran90, Python, C++), as writing out data to flat files carries an intrinsic overhead that could be avoided altogether. Binary data formats such as HDF5 seemed promising initially, but the benefit of having a small improvement in speed at run-time did not outweigh the cost of having to write HDF5 handlers in all of the working languages of the project.

We were also well aware that many languages have interfaces for passing arrays and other common data types between languages (e.g. Python's ctypes library, or using some implementation of a boost wrapper). However, for similar reasons to that which kept us from pursuing HDF5, we opted not to write custom interfaces for passing data. While we would likely have seen a speed increase at run-time if we were also creating and analyzing images in the loop, the overhead of writing three separate interfaces was not outweighed by the marginal gains in efficiency we would have seen. Therefore, data storage using flat ASCII files are the working paradigm until other solutions are deemed necessary.

Because we are using flat files, we adopted a standard naming convention for all output files regardless of their storage location. For any one particular run of the JSPAM code, we identify a unique output file via the SDSSID or name used, the run number corresponding to the line number in the input file, a flag indicating whether the data corresponds to the initial output or the final output, and the number of particles initialized in each galaxy. The output files for a unique run would have the form shown in Listing 3.

Listing 3: Output files from JSPAM

```
name.run_number.i.n1_particles.n2_particles.txt
name.run_number.f.n1_particles.n2_particles.txt
```

Further, we needed to have an appropriate directory tree set up for any particular run. We can visualize this tree in Figure 6

```
root
└── output
    └── {SDSSID}
        ├── {SDSSID}.humanscores.txt
        ├── run0000
        │   ├── {SDSSID}.0000.i.{n1_particles}.{n2_particles}.txt
        │   └── {SDSSID}.0000.f.{n1_particles}.{n2_particles}.txt
        ├── run0001
        │   ├── {SDSSID}.0001.i.{n1_particles}.{n2_particles}.txt
        │   └── {SDSSID}.0001.f.{n1_particles}.{n2_particles}.txt
        ├── run0002
        │   ├── {SDSSID}.0002.i.{n1_particles}.{n2_particles}.txt
        │   └── {SDSSID}.0002.f.{n1_particles}.{n2_particles}.txt
        ├── run...
        │   ├── {SDSSID}.{...}.i.{n1_particles}.{n2_particles}.txt
        │   └── {SDSSID}.{...}.f.{n1_particles}.{n2_particles}.txt
```

Figure 6: Output directory tree

At this point `jspamcli.py` only needs minor additions to work with the rendered image creation software and difference code which comprised the other half of this project. Incorporating all pieces of the project in an automated machine learning framework would require only a few augmentations to the existing code, and would support the ultimate goal of optimizing the parameter space for better convergence on solutions. This is likely to be explored in continuing research on this project.

F JSPAM Command Line Interface (`jspamcli.py`)

Usage of the original JSPAM code described in Wallin et al. (8) requires a string argument of initial simulation parameters, and accepts a number of options to specify run settings and simulation quantities such as the number of particles to be initialized

around each center-of-mass. While all of the existing functionality has been fundamentally unchanged, we set out to make interaction with the software more user friendly and to initiate the minimization of the human-in-the-loop by automating several tedious necessary actions. The result incorporates the previously mentioned classes as a command line interface for JSPAM, `jspamcli.py`, which now handles most of the overhead associated with running the software.

`jspamcli.py` can be used in four modes that are summarized in 1. Mode 1 (`python jspamcli.py -i`) allows for users to process runs for single targets interactively. This is the most straightforward usage, and allows for future users to become familiar with the software. The user is given an option to download files used for input using the `get_target_data` module, and then is allowed to select from available input files to run the program. The user is then asked to enter the number of particles to be initialized in each galaxy. Then, the interface proceeds in essentially the same way as in the batch processing modes, which are here described.

Before `jspamcli.py` can be run in any of the batch processing modes (modes $2 \rightarrow 4$), one or more batch run files must be specified. If none are specified, the program writes out a sample batch run file named `sample.txt` in the `batch_run_file` directory. The contents of a sample batch run file can be seen in Listing 4.

Listing 4: Sample batch run file

```
#target,n1_particles,n2_particles,first_run,last_run
58772298443535l614,500,500,100,100
```

However, if in the last column of a row there is the keyword `ALL`, the values given for `first_run` and `last_run` are ignored and all available non-rejected scored runs are processed. This file would read as shown in Listing 5.

Listing 5: Batch run file for all runs

```
#target,n1_particles,n2_particles,first_run,last_run
587722984435351614,500,500,000,000,ALL
```

Modes $2 \rightarrow 4$ are variations on batch processing modes and all use the available batch run files. Mode 2 (`python jspamcli.py -bi`) essentially gives the user an introduction to how batch processing is handled by `jspamcli.py` and is useful in contexts where the job to process does not have to be left running for a exceedingly long duration of time. This mode eventually does a batch processing job, but walks the user through the steps to successfully create a batch run file.

Mode 3 (`python jspamcli.py -b`) has all of the functionality of mode 2 but none of the interactivity. This mode runs as one would expect and processes all desired runs sequentially.

Mode 4 (`python jspamcli.py -bm`) contains methods for distributing the execution all instances of the `basic_run` required to complete the batch processing job across multiple processes on multiple cores. We wanted to minimize the need to change the original Fortran90 code, so the Python `multiprocessing` module was used to handle distributing calls to JSPAM to available worker processes, effectively increasing the speed with which any batch processing job can be completed by a factor of the number of cores available for use (loosely). While this does not necessarily reduce the time needed for any one simulation, as JSPAM has not necessarily been parallelized itself, this does reduce the amount of time required to run a large number of simulations fairly quickly just by simply doing more things at the same time. We can fairly easily significantly speed up our workflow by making use of the cores available to us on any workstation or machine.

Within the `multiprocessing` module is a `Pool` object that accepts an argu-

ment indicating the number of worker processes to be spawned. Upon execution of `jspamcli.py`, the argument passed to the program indicating the requested number of cores to be used is used to instantiate the `Pool` object, although we limit the number of cores to be used to half of those available to preserve resources and good relations with other users. Each execution of `basic_run` using a unique set of simulation parameters is considered to be completely independent of all other processes, as the inputs for each simulation do not depend on the results of the previous simulations. Therefore, we care very little if all simulations requested run sequentially. We can therefore use the `map_async` module to distribute calls to the appropriate calling function across the spawned worker processes.

While we do not need to have any shared memory in order to complete a "distributed" run of `basic_run`, we still must consider the fact that the program writes out to the same directory with each iteration (or at the end of each run). Again, the processes are not using a shared memory location during the run, but they are writing out these files to the same location. The simple fix was to add an `-m` flag in `basic_run` that indicates that the program should insert a distinguishing number corresponding to the process number in the name of the file. This allows for `jspamcli.py` to organize the output files appropriately.

Mode	Option	Behavior	Arguments
1	`-i`	run interactively	NONE
2	`-bi`	batch process interactively	NONE
3	`-b`	batch process (normal)	`batch_run_file...`
4	`-bm`	batch processing on multiple cores	`num_cores batch_run_file...`
5	`-g`	GIF Creation Tool	NONE

Table 1: List of command line options available in `jspamcli.py`. In the `-b` and `-bm` modes, multiple batch run files may be specified as long as they exist in the `batch_run_files` directory in the root directory.

IV RESULTS

The primary objective of this work was to support to continuing research with JSPAM by building a standardized framework through which future projects can interact with the JSPAM code. We have accomplished this objective by establishing a working framework that retrieves the appropriate data, executes the JSPAM code efficiently, stores and organizes the output of JSPAM, and encapsulates all relevant data and meta-data in instances of a `MergerRun` object. Further, although not discussed in this paper as it was not the focus of this work, the storage scheme does make further analysis and image creation substantially easier, and `jspamcli.py` was successfully used with a elementary approach at comparison of simulation images and real SDSS DR7, DR8, and DR9 images prepared with MergerEx.

There now exists a dedicated `MergerRun` class with member functions that implement most necessary tasks relating to the merger data. Rather than deal with data input and output on a case by case basis, we can now instantiate this class to encapsulate all necessary data for operations common to analysis of mergers in the context with which we are concerned. Further, the output from all member functions has been logically structured to reduce future headaches that come from handling large volumes of file input and output. There also exist several member functions only briefly mentioned here that aid in visualization of the mergers from start to finish.

The `MergerRun` class is used extensively in what we dub the JSPAM Command Line Interface (`jspamcli.py`). This program makes interacting with the JSPAM code a bit more straightforward and gives the user several modes of interaction that we found to be useful in different contexts. Further, we have now standardized input and output from JSPAM so that future work on the project can rely on predictable behavior.

We also created an option to execute `basic_run` asynchronously across a variable

number of cores. While we have not strictly parallelized `basic_run`, we've distributed its execution to make more efficient use of powerful workstations that are readily available.

V CONCLUSIONS

While our goal of establishing an interface for using and interacting with data from the JSPAM code has been achieved, it is clear that this work really lies mostly in a support role of other projects. That being said, successful data-intensive projects rely heavily on user-friendly data pipelines that handle most of the tedious labor and keep track of all relevant data in a predictable, well thought-out manner. We expect that the framework we have created will be useful in that it lessens the overhead necessary to bootstrap a project using the JSPAM code for its simulation software by eliminating the need to create ad-hoc methods for interacting with the simulation data. At this point, users are instructed on how to best interact with the software for their needs using `jspamcli.py`, and there is a clear method by which the simulation output data is stored and organized.

Because this framework is now well established, it becomes trivial to set up a batch run file to produce *all* 66,395 sets of run data and then the associated rendered images. Essentially, this gives us a massive set of ranked simulation data that can serve as a training set for future attempts at improving the results of JSPAM by applying machine learning techniques.

We recognize near the time of completion of this phase of the project that it would be quite useful to have a JSPAM interface object that can be used programmatically and that accepts an initial run string as an argument to a "run" method. In tasks where these initial condition guesses are being optimized, having an implementation of this would be essential for rapid testing.

While this project has not arrived at scientific results, we intend for our code to handle some of the heavy lifting in future projects electing to use the JSPAM code.

VI REFERENCES

[1] S. M. Alladin. The Dynamics of Colliding Galaxies. *The Astrophysical Journal*, 141:768, February 1965. doi: 10.1086/148161. URL http://adsabs.harvard.edu/abs/1965ApJ...141..768A. Provided by the SAO/NASA Astrophysics Data System.

[2] A. Holincheck. *A Pipeline for Constructing a Catalog of Multi-method Models of Interacting Galaxies*. PhD thesis, George Mason University, 2013. URL http://adsabs.harvard.edu/abs/2013PhDT.......176H. Provided by the SAO/NASA Astrophysics Data System.

[3] A. J. Holincheck, J. F. Wallin, K. Borne, L. Fortson, C. Lintott, A. M. Smith, S. Bamford, W. C. Keel, and M. Parrish. Galaxy Zoo: Mergers - Dynamical models of interacting galaxies. *Monthly Notices of the Royal Astronomical Society*, 459:720–745, June 2016. doi: 10.1093/mnras/stw649. URL http://adsabs.harvard.edu/abs/2016MNRAS.459..720H. Provided by the SAO/NASA Astrophysics Data System.

[4] Immanuel Kant. *Universal natural history and theory of the heavens : or, an essay on the constitution and the mechanical origin of the entire structure of the universe based on Newtonian principles*. Richer Resources Publications, Arlington, VA, 2009. ISBN 1935238914.

[5] C. Messier. Catalogue des Nébuleuses et des Amas d'Étoiles (Catalog of Nebulae and Star Clusters). Technical report, 1781. URL http://adsabs.harvard.edu/abs/1781cote.rept..227M. Provided by the SAO/NASA Astrophysics Data System.

[6] J. Pfleiderer and H. Siedentopf. Spiralstrukturen durch Gezeiteneffekte bei der Begegnung zweier Galaxien. Mit 7 Textabbildungen. *Zeitschrift für Astrophysik*, 51: 201, 1961. URL http://adsabs.harvard.edu/abs/1961ZA.....51..201P. Provided by the SAO/NASA Astrophysics Data System.

[7] A. Toomre and J. Toomre. Galactic Bridges and Tails. *The Astrophysical Journal*, 178:623–666, December 1972. doi: 10.1086/151823. URL http://adsabs.harvard.edu/abs/1972ApJ...178..623T. Provided by the SAO/NASA Astrophysics Data System.

[8] J. F. Wallin, A. J. Holincheck, and A. Harvey. JSPAM: A restricted three-body code for simulating interacting galaxies. *Astronomy and Computing*, 16:26–33, July 2016. doi: 10.1016/j.ascom.2016.03.005. URL http://adsabs.harvard.edu/abs/2016A%26C....16...26W. Provided by the SAO/NASA Astrophysics Data System.

Appendix I RESOURCES

In several cases throughout this paper, references are made to the project repository, or to pieces of software we make use of to complete our own work, or to the original JSPAM code. These references will be briefly described here.

All code developed for this project was originally done so at `https://github.com/jacksonlanecole/WallinCode`, but has since been refactored and is now located at `https://github.com/jacksonlanecole/WallinCode-restructure`. There is a detailed `README.md` file that gives setup and usage instructions. This project is a fork of the original codebase located at `https://github.com/JSPAM-Manga/WallinCode`, which is described in full in Wallin et al. (8).

The software used for target data acquisition and image preparation is called MergerEx, and is described in Appendix C of Holincheck (2). This project is located at `https://github.com/aholinch/MergerEx`, and comes with a precompiled version of JSPAM as described in the project `README.md`.

The initial simulation parameters come from a massive Citizen Science effort as part of the Galaxy Zoo: Mergers project. Volunteers essentially determined the better model out of those presented to them, and the results of these comparisons are presented in the data files found at `https://data.galaxyzoo.org/mergers.html`. While these files contain the results of all simulations run, some of these simulations were rejected on the basis of essentially having not undergone any transfer of gravitational energy, and therefore, having not interacted with one another. We remove these runs from the files and present data files containing *only* scored runs in `https://github.com/jacksonlanecole/WallinCode/tree/master/input`.