Testing Methods for Machine Comparison of Real and Simulated Interacting Galaxy Pair Morphologies

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

May 2018

TESTING METHODS FOR MACHINE COMPARISON OF REAL AND SIMULATED INTERACTING GALAXY PAIR MORPHOLOGIES

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

Although restricted three-body codes are in some cases less physically revealing than full n-body codes in simulating interactions and mergers of galaxy pairs, they are much faster and more computationally efficient in sampling a large solution space. For this reason, they are still widely used to reduce the initial parameter space of merger simulations before proceeding to more sophisticated, n-body models. In this work, we provide `jspamcli.py`, a user-friendly interface for interacting with an existing restricted three-body code used in the Galaxy Zoo: Mergers project, JSPAM (8), and to be used in conjunction with an rendered image creation and comparison suite which has been co-developed specifically to integrate with `jspamcli.py`.

TABLE OF CONTENTS

LIST OF FIGURES

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 be predisposed to assuming that the "very faint nebula" with two touching atmospheres he describes would be the often-imaged interacting galaxy pair, Messier 51a and b. Four years after Messier (5), German philosopher Immanuel Kant only 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 **??**.

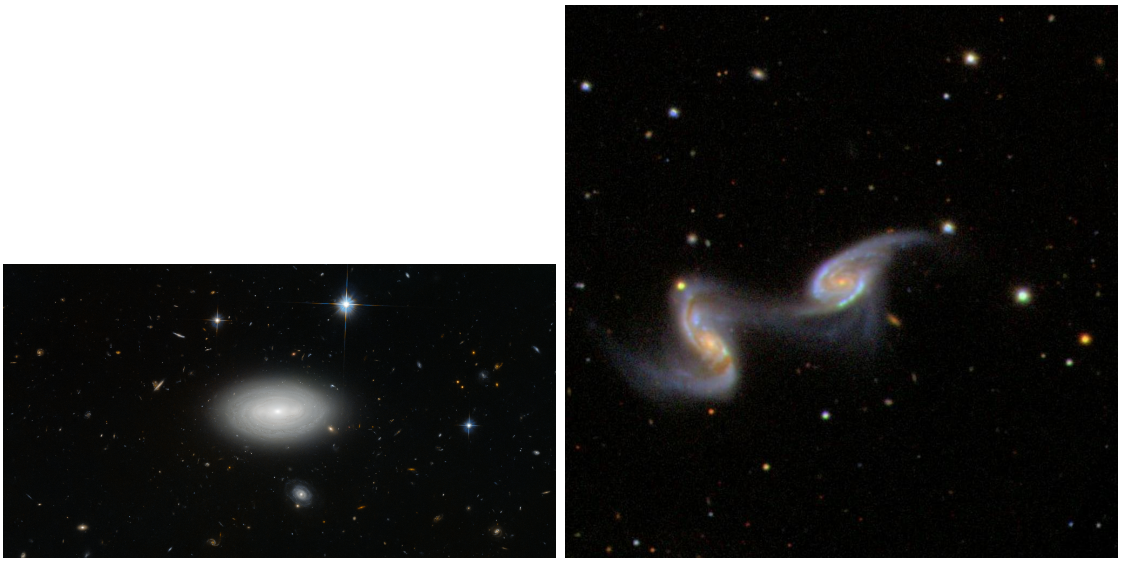
Galaxies are much less frequently found in isolation than they are found in *systems* of galaxies. We can therefore expect to observe many cases in which a pair of galaxies are interacting or have interacted in some way in the history of the system. Because these interactions are such a common occurrence, a natural logical next-step is to consider the role which these interactions, or more narrowly, collisions and mergers, play in the overall dynamics of the system. Alladin (1) claimed that collisions between galaxies

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

drastically increases the internal energy of a particular galaxy and can potentially lead to changes in overall stellar populations and further, that the changes in internal structure of galaxies caused by collisions is significant throughout the observable 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 structures are commonly referred to as "bridges and tails" (7).

Optical and spectroscopic observations can readily yield accurate measurements of the final positions of the galaxies in a merger, and occasionally the radial velocities of the members of a system can be observed as well. However, the primary interest lies in the mechanisms that result in the observed bridges and tails; this immediately suggests the need to find 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, much work has been done in the last 70 years to advance simulations of interacting systems, and further, to optimize the existing simulations for better convergence on solutions.



(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 the observed disturbed morphological features of interacting galaxy pairs, described as "bridges and tails", related 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 seen 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 the bridges and tails were thought 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 morpholoigcal 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 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. They purposely initialize slow, parabolic encounters between members of the galaxy pair, as they assume that faster encounters result in thinner tidal 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 the following assumptions. Numerous observable encounters of galaxies traveling along hyperbolic orbits seems highly improbable, and accordingly, observational data of the interacting pairs point to the interaction *not* having been a chance encounter. It seems much more probable that the interacting pair were already bound, and while still highly eccentric, their approach orbits were still sub-parabolic. In their work, they initialize a primary disk that is composed of a point mass at the center, and several concentric rings of test particles surrounding. 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).

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 certainly 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 to be calculated. These 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. The primary benefit is that they are significantly more computationally economical than their n-body counterpart. These are also considered valid, as the self gravity of the galactic disks is assumed to be almost negligent 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 was 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 on the basis of computational economy *when our solution space still contains "bad" solutions.* Essentially, we can now use the faster three-body algorithms to accurately simulate significant kinematic interactions (e.g. those between each point mass and the centers of each galactic disk), and then use either fitness-functions, automatic image recognition and classification via machine learning techniques, Citizen Science, or some combination of the three to constrain the solution space to only "good" or "okay" solutions. From our "good" solutions, we can begin to use full n-body simulations for further analysis.

It should be reiterated that in comparison to restricted three-body codes, full n-body simulations are naturally computationally expensive. However, simulations of dynamical processes within the disks of galaxies are better served by the computational complexity of n-body codes than are purely kinematic processes under the assumptions given by Toomre and Toomre (7). If the initial goal of a particular simulation can be oriented toward finding a valid solution for the initial conditions that contribute to the

observed morphological features, restricted three-body codes, such as JSPAM (8), serve well to lessen the computational overhead in running large numbers of simulations with reasonable resolution. Rather than immediately deal with the computationally expensive approach of calculating the self gravity within the disk material (among the stars) using n-body codes, researchers can use restricted three-body codes to run significantly large numbers of simulations relatively quickly, which allows simulation of solutions across the *entire* solution space.

In short, we want to make efficient use of computational time and to reduce the length of time the human-in-the-loop must spend "in the loop." In recent years, the Galaxy Zoo project made use of 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 on observed morphologies of interacting galaxies. This work aims primarily to develop a framework for testing general purpose machine fitness-functions. The simulated data come from the JSPAM code, found at `https://github.com/JSPAM-Manga/WallinCode`; this is a restricted three-body code originally written in FORTRAN that is described in Wallin et al. (8).

The working fork of the project is currently in regular development and can be found at `https://github.com/jacksonlanecole/WallinCode`.

III METHODOLOGY

A Overview

The primary goal of our work in this project is focused on the development of a framework that can be used to test general purpose fitness-functions for comparison of simulated disturbed morphologies with their real counterparts. Having a general purpose

fitness-function allows for the generation of preliminary machine scores for all sets of initial simulation parameters present for each of the 62 Galaxy Zoo: Mergers targets found at `https://data.galaxyzoo.org/mergers.html`. The Galaxy Zoo: Mergers project produced ranked scores that are indicative of the degree to which the resultant morphologies of different sets of initial conditions for a merger correlate with their real counterparts. These rankings result from the 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 review of over 3 million simulations by volunteers (3). We make use of the points in the parameter space that have received a citizen science score.

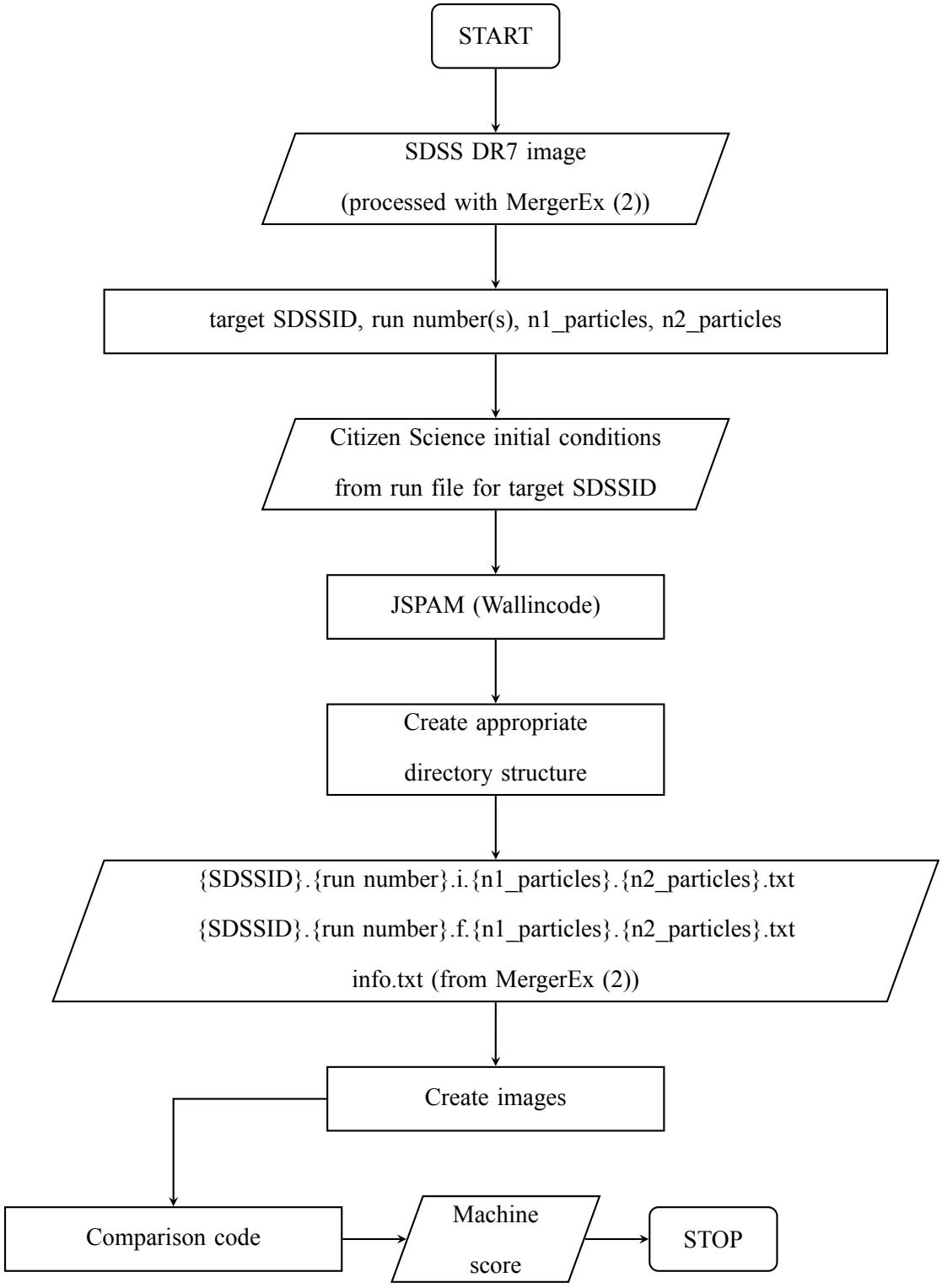


Figure 3: Project flowchart. Here, it should be noted that our primary goal was developing a framework for testing comparison codes. This endeavor will likely see work in the near future.

The primary motivation for having a machine scoring mechanism for these types of simulations is to reserve human effort for distinguishing the best fitting morphologies out of the good rendered morphologies. This is in contrast with the current *modus operandi*, which in some cases results in humans being tasked with classifying a *clearly* ill-fitting rendered morphology as "bad". If we can task a computer with making classifications that require less acute discernment, humans in the loop can be tasked with finding the *best of the best* rendered morphologies.

Therefore, we operate under the assumption that an ideal machine scoring mechanism should be able to essentially recreate the results of the Galaxy Zoo: Mergers Citizen Science effort nearly exactly. However, in this work, we focus more on developing a framework to handle 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. In other words, while the ultimate goal for JSPAM is the creation of a general purpose fitness function to compare the output of the JSPAM simulations to the real systems it is supposed to model, the fitness function used in our current working framework is largely a placeholder to ensure that the framework is working properly score. We recognize that further work must be done in developing fitness-functions, and expect that our work will aid further success in doing so.

B Image Preparation and Target Data Acquisition

The software we use for target image preparation and for constraining the initial simulation parameter values is MergerEx, described in Holincheck (2) and 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 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 shown 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, we follow the process for estimation of the disks as described in Appendix C of Holincheck (2).

Rather than leave this task up to users in future work, a directory containing target images, the initial simulation parameters, and the full-color press release images has been included in the project repository, referenced in **??**.

C Input Storage: Targets and Target Input Files

In many cases during this project, we wanted to work with *only* parameters that result in simulations that would have received a Citizen Science score, and therefore elected to simply remove a significant number of simulation parameters that resulted in a rejected simulation.

The input data files that contain the initial simulation parameters are located in the `input` directory in the root 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 initial simula-

tion parameters from scored runs. They are made available in the GitHub repository.

D `merger` Python Package

One of the primary challenges present in many projects in astronomy or astrophysics is simply doing adequate book-keeping of data and related information. 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 `MergerRun` and `Galaxy` classes, both of which are made available from the top level in the package. They can be imported from the directory containing the merger package via "`from merger import MergerRun, Galaxy`" or simply "`from merger import .`", although this second import statement also imports a `data_tools` package and `get_target_data` module which will be described later.

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 simply turns this text file into a dict, which it then initializes as several of the class instance attributes. An instance of this class also requires 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. This can all be done in a statement such as

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

Having a dedicated MergerRun class allows for *all* of the necessary information for a particular run of a merger to be passed between functions and programs. While

this method has potential to be a bit slower, we expect programming using this class to remain a bit simpler and perhaps more clear.

This class also instantiates twice the `Galaxy` class, one referred to as `primary` and the other referred to as `secondary`. We can therefore more clearly keep track of the SDSS ID or name, and regarding 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. In other words, the following code block

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

will create the following path in your current working directory: `root_name/child_1/child_2/child_3/`. Although this may seem unnecessary, this reduces the chance of error in organization of the large volume of simulation data that will be produced.

Further, in the top level of the package also exists `get_target_data`. This is a module that specifically scrapes `https://data.galaxyzoo.org/mergers.html` for all of the available target data, allows for a user to select from the targets available, downloads and unarchives the data into an `input` directory in the working directory. This is more apparently useful in the interactive mode of `jspamcli.py`.

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

The original JSPAM code described in Wallin et al. (8) can be called from the command line via `./basic_run` with a required string argument of initial simulation parameters, and it accepts a number of options to specify run settings and simulation quantities such as the number of particles to be initialized around each center, for instance. While essentially all the existing functionality has been fundamentally unchanged, we set out to make interaction with the software more user friendly and intuitive. Further, we wanted to initiate the minimization of the human-in-the-loop even in the initial program setup for new users. After the Fortran90 code is compiled, users can run `jspamcli.py` using Python3. This program can be run in four modes selected via command line options. These modes are given as follows: Mode 1 allows interactive processing of multiple

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 process 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.

runs of one target and can be called by `python3 jspamcli.py -i`. Interaction with JSPAM is easy in this context, as all options are handled interactively. The user is given an option to download input files using the `get_target_data` module, and then can choose 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 program runs in essentially the same way as in the batch processing modes.

Before `jspamcli.py` can be run in any of the batch processing modes (modes $2-4$), 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 are below:

```
#target,n1_particles,n2_particles,first_run,last_run
587722984435351614,500,500,100,100
```

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

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

Modes $2-4$ are variations on batch processing modes and all make use of the available batch run files. Mode 2 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.

Mode 3 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 contains methods for parallelizing the `basic_run` process. We wanted to minimize the need to change the original Fortran90 code, so the Python `multiprocessing` module was used parallelize the execution of the JSPAM simulation via `basic_run` across multiple processors with different arguments passed to the simulation in each. 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 sim-

ply 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 argument 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 particular 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.

Although we can now execute `basic_run` processes across multiple cores, we needed to take into account that this program writes out the initial and final simulation data file. While the processes are not using a shared memory location during the run, 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.

F Output Storage: Directory and File Naming Conventions

We explored the possibility of adopting binary data formats such as HDF5 rather than defaulting to flat files to store data. While making use of binary data formats either by implementing an IO interface that works across our existing programs in Fortran90, Python, and C++ or by simply using HDF5 could potentially reduce IO overhead, it was

determined to not be an effective use of time at this point in the project. Therefore, flat ASCII files are the working paradigm until another solution is 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

```
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 4

```
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 4: 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.

IV RESULTS

The primary objective in carrying out this project was to build an infrastructure to support the investigation into methods of optimization of initial simulation parameters based on the output of the JSPAM simulations. We have accomplished this objective by establishing a working framework 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 `merger` class with member functions that implement most necessary tasks relating to the merger data. Rather than deal with data I/O 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 I/O. There also exist several member functions that aid in visualization of a particular simulation from start to finish.

The `merger` 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 intuitive and gives the user several modes of interaction that we found to be useful in different contexts. Further, we have now standardized I/O from JSPAM so that future work on the project can rely on predictable I/O 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 ANALYSIS

At this point, there are no results to analyze as the project is still in development.

VI CONCLUSIONS

While our goal of establishing a framework for testing various fitness functions and interacting with the JSPAM code has been achieved, it is clear that the actual fitness function used is decidedly *not* the ideal fitness function to use in the future. That being said, 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.

Further, users now have available to them in the project repository all available sets of initial simulation parameters that received a Citizen Science score, as well as a directory containing real color and calibrated (to the needs of the comparison code) target photographic images and initial simulation parameter files all organized by the target name.

Because this framework is now well established, we can easily 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 scored set of simulation data that can serve as a significant training

set on which to train models in future attempts at applying appropriate machine learning algorithms to quickly reduce the solution space presented to a human for analysis to the "best of the best."

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

VII 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.