NRml_TOV Documentation

David Boyer

June 25, 2024

Contents

Abstract

NRml_TOV is a Tolmann-Oppenheimer-Volkoff Equation solver. It uses the OdieGM[1] framework to solve the set of ordinary differential equations and the GRHayL[2] library for Equation of state initialization and calculations. NRml_TOV has support of three different types of EOSs: Simple Polytrope, Piecewise Polytrope, and Tabulated. A user can choose what EOS to use, what numerical ODE method to use, and even adjust tolerances and step sizes to customize the level of speed and accuracy they desire. The solver has been validated against the NRpy[3] library's TOV solver for all three types of EOS.

1 Overview

The Tolmann-Oppenheimer-Volkoff (TOV) equations are a solution to Einstein's Field equation that describes an spherically symmetric isotropic star in equilibrium, most importantly for that of a neutron star:

$$\frac{dP}{dr} = -\frac{Gm}{r^2}\rho(1+\frac{P}{\rho c^2})(1+\frac{4\pi r^3 P}{mc^2})(1-\frac{2Gm}{rc^2})^{-1} \tag{1}$$

$$\frac{dM}{dr} = 4\pi r^2 \rho \tag{2}$$

$$\frac{d\nu}{dr} = -(\frac{2}{P+\rho c^2})(\frac{dP}{dr}) \tag{3}$$

$$\frac{d\bar{r}}{dr} = \frac{\bar{r}}{r\sqrt{1-\frac{2m}{r}}} \tag{4}$$

where normally ($c = G = 1$) for geometerized units. For more info, refer to the original papers of [4, 5].

A full solution to these equations when incorporating an equation of state (EOS) produces initial data about the star that can be used as initial data for time-evolution codes. These solutions cannot be found analytically, though, and must use a numerical solver to find them. NRml_TOV seeks to do that quickly, accurately, and to as much ease to the user as possible.

This documentation seeks to explain how to set up the necessary dependencies and how the user interacts with the program. Afterwards, we will describe how each equation of state type is implemented in the solver, as well as validation for each solver. **This documentation will assume you are using the Linux terminal. Make the necessary edits to your commands if using another operating system.**

2 Getting Set Up

Before you can use `NRml_TOV`, you must first follow these steps:

1. Clone the `NRml_TOV` repository from GitHub
2. Clone the `GRHayL` library and at least compile it (installation is optional)
3. Edit the default filepaths within the makefile and `NRml_TOV_Driver_main.c`
4. make and execute the executable `./NRml_TOV`

2.1 Cloning NRml_TOV from GitHub

If you have this documentation, chances are you have already done this, but if your reading this off the github page, here is the command line to clone the repository and begin making use of it.

```
git clone https://github.com/dboyer7/NRml_TOV.git
```

Make sure you do this in the directory you wish to put `NRml_TOV`.

2.2 Cloning and compiling the GRHayL library

To run `NRml_TOV`, you must also get the `GRHayL` library[2], as `NRml_TOV` makes major use of the library for initializing and using Equations of State.

```
git clone https://github.com/GRHayL/GRHayL.git
```

The `GRHayL` library also needs to be compiled to run with `NRml_TOV`. Although the commands are written within its documentation, I will also put them here, since they are short commands.

These commands should be run within the `GRHayL` directory

```
CC=gcc ./configure
make
```

A new directory named `build` should now be in the `GRHayL` directory. Take note of its filepath, as it will be necessary when editing filepaths in `NRml`'s makefile.

It is only necessary to compile the `GRHayL` library, no installation to the system is necessary. If you do want to install the library to your system, follow the instructions given within `GRHayL`'s documentation and README file.[2]

2.3 Filepath Editing

There are a couple locations in the makefile and `main.c` that require a filepath to locate your necessary files and dependencies. Make sure these are corrected to your system's filepath, else the compiler won't know where to find them.

2.3.1 Makefile filepaths

These are the lines in the makefile that need to be changed with correct filepaths:

```
Line 3|  LDFLAGS = -L/path/to/the/x86_64-linux-gnu/hdf5/serial
    -L/path/to/the/GRHayL/build/lib
    -lhdf5
    -lgrhayl
......
```

```
    ......
    Line 5|  INCLUDEDIRS= -I/path/to/the/include/hdf5/serial
        -I/path/to/the/GRHayL/build/include
```

Please note that I have left examples in the makefile of what the filepath most likely looks like, so minimial editing should be required. Just double check that the filepaths are correct, else it won't compile.

2.3.2 `NRml_TOV_Driver_main.c` filepaths

The main file also has dependencies on the `GRHayL` library, so make sure these filepaths in `NRml_TOV_Driver_main.c` are correct:

```
    Line 3|  #include "path/to/the/GRHayL/GRHayL/include/nrpyeos_tabulated.h"
    ......
    ......
    Line 28|  char* table_path = "/path/to/your/tabulated/EOS/table.h5;
```

In line 3, you are looking for the `GRHayL` directory once again, but instead of looking for the build directory, you need the `GRHayL` directory within the `GRHayL` directory. The file needed (`nrpyeos_tabulated.h`) helps with memory freeing at the end of the program.

On line 28, you only need to edit this if using a tabulated eos, else just ignore it. More will be discussed about `table_path` in section 3.1.

2.4 Make and Execute

All that is left now is to run one more command, compile, and execute. (Note: If you went ahead and installed `GRHayL` to your system, the `export` command is not neccessary. Go ahead and hit make.)

```
    export LD_LIBRARY_PATH=/path/to/the/GRHayL/build/lib:$LD_LIBRARY_PATH
    make
    ./NRml_TOV
```

If done correctly, `NRml_TOV` should run, printing out the initial and final values of the solver's default TOV solution, and the final line printed should be:

```
    ODE Solver "Odie" for NRml_TOV Shutting Down...
```

If this occurs and you now have a written .txt file called `ooData.txt` in the directory, then congratulations, `NRml_TOV` is all set up and ready for you to use and customize.

3 Interfacing

`NRml_TOV` was designed to be simple for a user to use. As such, the only piece to this solver a user need to edit are their EOS parameters, boundaries, and initial conditions. `NRml_TOV` (thanks to the design of the ODE solver, `OdieGM`[1]) also gives the user access to decided what ODE method type they wish to use, and the tolerances and step sizes for the algorithm. The next section will list each of the variables the user has access to and how to use it, while the section following it will describe the general logic behind the rest of the solver. Further details on the solver can be found in the documentation of `OdieGM`[1], the code of which `NRml_TOV` is originally derived.

For those that wish to see examples of solutions from `NRml_TOV`, see section 5, as well as the template main functions in the `template solutions` directory.

3.1 Editing `NRml_TOV_Driver_main.c`

Here is a list of all the variables the user has access to and a description of what they do:

- EOS initialization
  - `char type`
    - Description: Choose one of the three supported EOS types for the solver (Simple Polytrope, Piecewise Polytrope, or Tabulated EOS).
    - Valid Inputs: `s` (Simple Polytrope), `p` (Piecewise Polytrope), `t` (Tabulated EOS)
  - `char* table_path`
    - Description: If using a Tabulated EOS, put in the filepath to the table you wish to use.
    - Valid Inputs: Valid Filepath
  - `int neos`
    - Description: If using a Piecewise Polytrope, select the number of regions for your EOS. A simple EOS will be a piecewise EOS with one region.
    - Valid Inputs: An integer value 1-10 (Piecewise) or 1 (Simple)
- Simple and Piecewise Polytrope Parameters
  - `double K0`
    - Description: K-parameter of the first region of the polytrope. All other K values will be automatically calculated.
    - Valid Inputs: Any valid double. $K_0 = 1.0$ by default.
  - `double GammaTh`
    - Description: Gamma-parameter to calculate the P_{cold} and ϵ_{cold}
    - Valid Inputs: Any valid double. $\Gamma_{th} = 2.0$ by defulat.
  - `Gamma[i]`
    - Description: The i-th region's Gamma Parameter. Indices go from 0 to `neos-1`.
    - Valid Inputs: Each Γ Parameter. Make sure you have `neos` values.
  - `rhoBound[i]`
    - Description: The i-th region boundary of ρ on a P vs. ρ plot. Indices go from 0 to `neos-2`.
    - Valid Inputs: Each $\rho_{boundary}$ Parameter. Make sure you have `neos-1` values.
- Tabulated EOS Parameters
  - `double YeAtm`
    - Description: Atmospheric Electron Fraction

    * Valid Inputs: Value within table bounds
  - `double YeMin`
    * Description: Minimum Electron Fraction
    * Valid Inputs: Any Value. Values outside table bounds will be automatically set to the table end (RECOMMENDED).
  - `double YeMax`
    * Description: Maximum Electron Fraction
    * Valid Inputs: Any Value. Values outside table bounds will be automatically set to the table end (RECOMMENDED).
  - `double TAtm`
    * Description: Atmospheric Temperature
    * Valid Inputs: Value within table bounds
  - `double TMin`
    * Description: Minimum Temperature
    * Valid Inputs: Any Value. Values outside table bounds will be automatically set to the table end (RECOMMENDED).
  - `double TMax`
    * Description: Maximum Temperature
    * Valid Inputs: Any Value. Values outside table bounds will be automatically set to the table end (RECOMMENDED).
  - `double T_in`
    * Description: Beta-Equilibrium Temperature. Used for slicing table.
    * Valid Inputs: Value within table bounds.
- Rho values
  - `double rhoCentral`
    * Description: Central BARYON Density
    * Valid Inputs: Any value, but recommended to be of order 10^{-1}
  - `double rhoAtm`
    * Description: Atmospheric Baryon Density
    * Valid Inputs: Positive value, within table bounds if using a table.
  - `double rhoMin`
    * Description: Minimum Baryon Density
    * Valid Inputs: Any Value. If using a table, values outside table bounds will be automatically set to the table end (RECOMMENDED).
  - `double rhoMax`
    * Description: Maximum Baryon Density
    * Valid Inputs: Any Value. If using a table, values outside table bounds will be automatically set to the table end (RECOMMENDED).
- ODE Method and Output
  - `double step`
    * Description: Initial step size. Remains the same if using a non-adaptive step.
    * Valid Inputs: Any Positive value

  - `double current_position`
    - Description: Initial position.
    - Valid Inputs: Any positive value. Set to zero by default (RECOMMENDED).
  - `const int size`
    - Description: Maximum number of steps to be taken
    - Valid Inputs: Any positive integer. Recommended to be slightly larger to guarantee reaching to outside of the star
  - `int adams_bashforth_order`
    - Description: Order of Adams-Bashforth method, if using one.
    - Valid Inputs: Integer 1-19 (19 orders, but 4 is recommended if using AB)
  - `bool adaptive_step`
    - Description: Are you using an adaptive or non-adaptive method
    - Valid Inputs: `true` (for adaptive methods) or `false` (for non-adaptive)
  - `double error_limit`
    - Description: Error tolerance limit for the solver
    - Valid Inputs: any positive value
  - `char file_name[]`
    - Description: What is the name of the file you wish to save your data to?
    - Valid Inputs: Any string.
  - `step_type`
    - Description: What method are you using? (Hybridizing disabled for `NRml_TOV`)
    - Valid Inputs: See methods below

- Advanced Tolerance and Step Settings
  - `double scale_factor`
    - Description: Control variable in the adaptive methods
    - Valid Inputs: ...Just don't mess with this, okay?
  - `double error_safety`
    - Description: Control variable in the adaptive methods
    - Valid Inputs: ...Just don't mess with this, okay?
  - `double ay_error_scaler`
    - Description: Control variable in the adaptive methods
    - Valid Inputs: ...Just don't mess with this, okay?
  - `double ady_error_scaler`
    - Description: Control variable in the adaptive methods
    - Valid Inputs: ...Just don't mess with this, okay?
  - `double max_step_adjustment`
    - Description: Largest amount to change an adaptive step by.
    - Valid Inputs: Any double. Default recommended.
  - `double min_step_adjustment`
    - Description: Smallest amount to change an adaptive step by.
    - Valid Inputs: Any double. Default recommended.

  - `double absolute_max_step`
    * Description: Largest step size to take.
    * Valid Inputs: Any double. Default recommended.
  - `double absolute_min_step`
    * Description: Smallest step size to take.
    * Valid Inputs: Any double. Default recommended.
  - `double error_upper_tolerance`
    * Description: Upper tolerance for adaptive method
    * Valid Inputs: Any double. Default recommended.
  - `double error_lower_tolerance`
    * Description: Lower tolerance for adaptive method
    * Valid Inputs: Any double. Default recommended.

The foundational ODE solver, `OdieGM`, also gives the the user the ability to choose what ODE method you would like to use. Each of the different methods you can choose from are seen below, and their butcher tables are defined in the file `nrpy_odiegm.h`. Refer to that file, as well as the `OdieGM` documentation for more info.[1]

Set the variable `step_type` to one of the following values:

- Adaptive Methods
  - Adaptive Heun-Euler: `nrpy_odiegm_step_AHE`
  - Adaptive Bogacki-Shampine: `nrpy_odiegm_step_ABS`
  - Adaptive Runge-Kutta-Fehlberg: `nrpy_odiegm_step_ARKF`
  - Adaptive Cash-Karp: `nrpy_odiegm_step_ACK`
  - Adaptive Dormand-Prince Fifth Order: `nrpy_odiegm_step_ADP5`
  - Adaptive Dormand-Prince Eighth Order: `nrpy_odiegm_step_ADP8`
- Non-Adaptive Methods
  - Euler's Method: `nrpy_odiegm_step_euler`
  - Second Order Runge-Kutta Heun: `nrpy_odiegm_step_RK2_Heun`
  - RK2 Midpoint: `nrpy_odiegm_step_RK2_MP`
  - RK2 Ralston: `nrpy_odiegm_step_RK2_Ralston`
  - Runge-Kutta Third Order: `nrpy_odiegm_step_RK3`
  - RK3 Heun: `nrpy_odiegm_step_RK3_Heun`
  - RK3 Ralston: `nrpy_odiegm_step_RK3_Ralston`
  - Strong Stability Preserving `RK3: nrpy_odiegm_step_SSPRK3`
  - Runge-Kutta Fourth Order: `nrpy_odiegm_step_RK4`
  - Dormand-Prince Fifth Order: `nrpy_odiegm_step_DP5`
  - Alternate Dormand-Prince Fifth Order: `nrpy_odiegm_step_DP5A`
  - Cash-Karp Fifth Order: `nrpy_odiegm_step_CK5`
  - Dormand-Prince Sixth Order: `nrpy_odiegm_step_DP6`
  - Luther's Sixth Order Method: `nrpy_odiegm_step_L6`
  - Dormand-Prince Eighth Order: `nrpy_odiegm_step_DP8`
- Adams-Bashforth Methods

- Adams-Bashforth: `nrpy_odiegm_step_AB`
  - Orders 1-19
  - Select order with variable `adams_bashforth_order`

Please note: Although there seems to be many different methods to choose from, I recommend only using either the `Adaptive Runge-Kutta-Fehlberg` or the `Adaptive Dormand-Prince Eighth Order`, so you can take advantage of the adaptive methods.

3.2 Under The Hood

This will be just a general discussion of how `NRml_TOV` finds a TOV solution. Further details of getting that solution can be found in `OdieGM`'s full documentation.[1]

3.2.1 `NRml_TOV_Driver_main.c`

Once a user has customized `NRml_TOV_Driver_main.c` with their EOS parameters and ODE settings, the solver begins setting up.

1. Initialize EOS
   - User variables are used to define the EOS struct using the `GRHayL` library. `GRHayL` creates the EOS based on the choice of EOS given.
2. Error check
   - `NRml_TOV` then does some minor error checks, to make sure you are using a compatible ODE method.
3. Create Parameter Struct
   - `NRml` then creates a parameter struct specifically designed to hold the parameters for the TOV equations, and the `GRHayL`-defined EOS struct becomes a part of it.
4. Create ODE System and Driver
   - An ODE system and driver are then created to move begin solving the TOV equations.
5. Get Initial Condition
   - Initial pressure and total energy density is then found to start the solver off
6. 'Evolve' to next step
   - `NRml` (Through the use of the `OdeiGM` solver), then goes through solving the TOV equations with the ODE method that was specified by the user.
7. Write to output file
   - At each step, the results are written to a file for a full solution to the TOV throughout the star.
8. Repeat steps 6/7 until termination condition
   - Solve the TOVs until the termination condition is hit. For `NRml_TOV`, this condition is when pressure hits zero at the star's surface.
9. Free memory
   - After a full TOV solution is found, `NRml` frees the memory that was allocated.
10. Close

It is important to note that the TOV solution is written to an output file that can be renamed by the user. Each line of this file is written as follows (in exact order):

1. Scwarzchild Radius (r)
2. Total Energy Density (ρ)
3. Baryon Density (ρ_b)
4. Pressure (P)
5. Mass (M)
6. ν
7. 2ν
8. Isotropic Radius ($\bar{r}$)

3.2.2 `NRml_TOV_user_methods.c`

These are the functions that are used to create the ODE system and Driver structs that run the solver. More details about the general functions are in the `OdieGM` Documentation.[1] This documentation mainly describes how they are used in `NRml_TOV`:

- `Struct constant_parameters`
  - Struct to hold parameters of the EOS, EOS type, as well as central baryon density.
- `exception_handler(double, double)`
  - Handles exceptions that may come from numerical errors. `NRml_TOV` only defines one, making sure that if by some chance pressure goes below zero, we set it back to zero and then terminate, since we reached the pressure asymptote at the star's surface.
- `do_we_terminate(double, double [], struct constant_parameters*)`
  - Defines the condition to stop the program. All termination conditions for `NRml_TOV` involve reaching the surface of the star, where the TOV equations are not defined.
- `const_eval(double, const double [], struct constant_parameters*)`
  - Originally defined in `OdieGM` to handle any constants within the ODE system. However, `NRml_TOV` has repurposed this function to instead calculate all values defined within the EOS instead of the TOVs (namely ρ and ρ_b). The function checks what type of EOS is being used, and then uses associated `GRHayL` functions to calculate densities in the `ghl_eos struct`.
- `diffy_Q_eval(double, double [], double [], void*)`
  - This is where the TOVs are actually defined. `OdieGM` takes these ODEs and puts them into a driver struct, and then uses its own algorithms to then solve the system with the ODE method of your choice.
- `known_Q_eval(double, double [])`
  - From `OdieGM`. Used if there is a known solution to the general ODE system. In our case with the TOVs, there are none, so this function is left empty.
- `get_initial_condition(double [], struct constant_parameters*)`
  - Get the initial pressure from `GRHayL` functions to begin solving the TOVs.
- `assign_constants(double [], struct constant_parameters*)`
  - Like `const_eval()`, this function also is repurposed to report the EOS quantities ρ and ρ_b.

3.2.3 `NRml_TOV_proto.c`, `NRml_TOV_funcs.c`, and `nrpy_odiegm.h`

There are several other files in this program that define the system and driver structs and butcher tables for each of the different ODE methods you can choose from. These were ripped directly from `OdieGM` without changes, so refer to its documentation for more info.

4 Implementations of the Equation of State

This section will describe how the three implementations of EOS differ in logic. While the TOV solver itself does not differ between EOS, function calls that are used to initialize and use the EOS with `GRHayL` differ. Overall, though, the general approach is to find the initial pressure using a `GRHayL` function call, and then find densities each step of the solution using another `GRHayL` call using the `GRHayL` defined EOS throughout.

4.1 Simple Polytrope Solve

The simple polytrope is an 1-region EOS defined by the function $P(\rho_b) = K\rho_b^{\Gamma}$.

1. Initial Conditions

   - `NRml_TOV` starts by using the `K` and `Gamma` within the `GRHayL` defined EOS and uses it to calculate initial pressure from the central density provided.

2. Density Evaluations

   - Just like the initial condition, we use the polytropic equation along with the `GRHayL` defined `K` and `Gamma` values to recalculate ρ_b and ρ each step of the solution.

4.2 Piecewise Polytrope Solve

The Piecewise Polytrope is the more general version of the simple polytrope. Multiple regions are defined by the user, each defined by there own polytrope, $P(\rho_b) = K_i\rho_b^{\Gamma_i}$. The user must define all $Gamma_i$ values, and the $\rho_b ound$ values where the regions shift in the piecewise function.

1. Initial Conditions

   - Instead of calculating the initial pressure directly from K and Γ, We call a `GRHayL` function to do it directly by just giving the central density. `GRHayL` handles the calculation from there.

2. Density Evaluations

   - Once again, `GRHayL` calculates the new ρ_b and ρ from the current pressure position.

4.3 Tabulated EOS Solve

Unlike the previous two EOS types, the tabulated solver relies completely on an EOS table, which can be found online. When using this EOS type, make sure you know the filepath to your table, and make sure it is a `.h5` file, as `GRHayL` reads `.h5` files.[2] Instead of parameters K and Γ, `GRHayL` uses a linear interpolator to pull values off the table.[2]

1. Initial Conditions

   - `NRml` calls a `GRHayL` function to interpolate the table and find the initial pressure at a given central baryon density.

2. Density Evaluations

   - Just have to call the associated `GRHayL` functions to find our densities.

5 Validation and Examples

To validate the results calcualted by `NRml_TOV`, we tested this solver against the already trusted TOV solver in the `NRpytutorial` library.[3] We have included plots of the relative error for baryonic density (ρ_b), total mass/energy density (ρ), pressure (P), and mass (m). All relative error is calculated by:

$$\epsilon_{rel} = \frac{|x_{NRpy} - x_{NRml}|}{x_{NRpy}} \tag{5}$$

Each of these tests were redesigned to be templates for editing your own `main` function, and are included in the `template_solutions` directory. Refer to those if you are wanting to practice using `NRml_TOV`.

5.1 Simple Polytrope Solve

Please see the `Simple` template for parameters and values used for this test.

The TOV solution agrees with the values from the `NRPy` TOV solution to around 10^{-13} in relative error among all values.

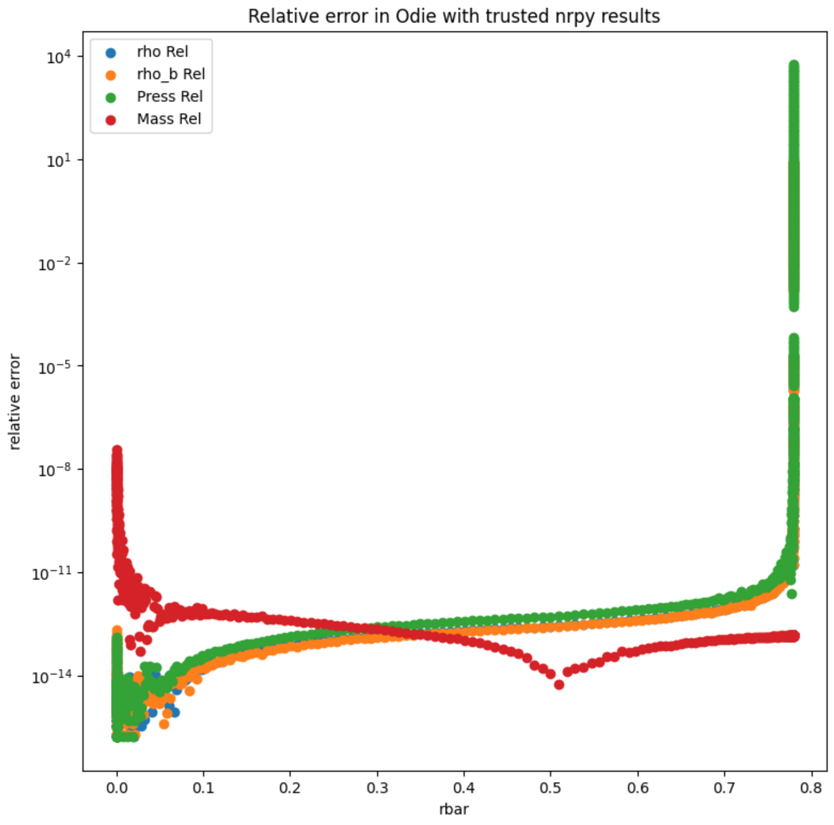

Figure 1: Relative error between `NRml_TOV` and `NRPy` TOV solutions. Most error around 10^{-13}

5.2 Piecewise Polytrope Solve

Please see the Piecewise template for parameters and values used for this test.

The TOV solution agrees with the values from the `NRPy` TOV solution to around 10^{-13} in relative error among all values.

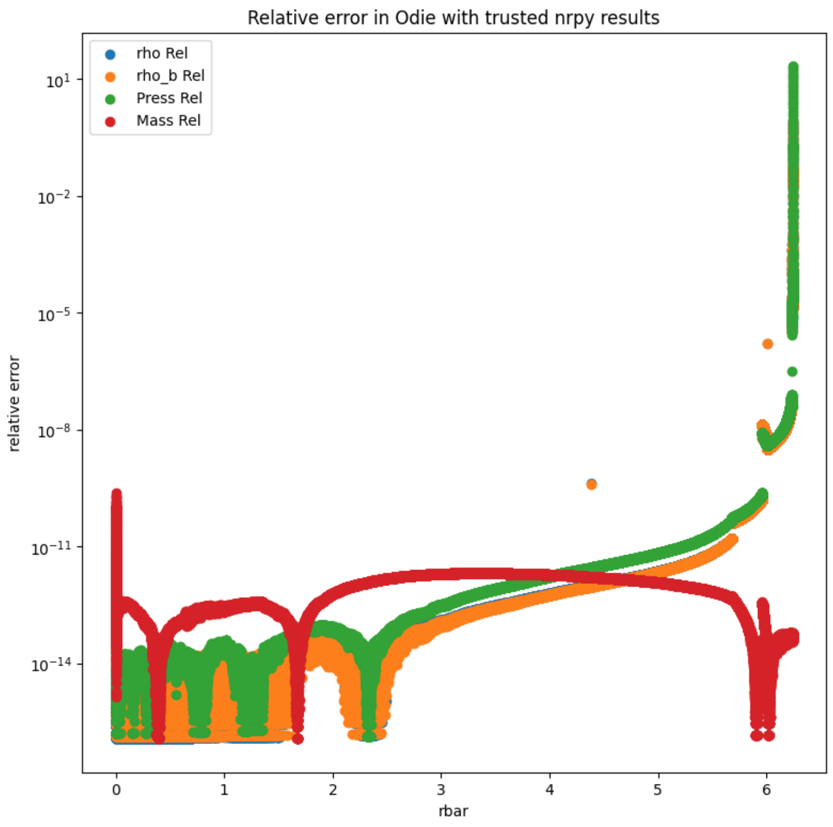

Figure 2: Relative error between `NRml_TOV` and `NRPy` TOV solutions. Most error around 10^{-13}

5.3 Tabulated EOS Solve

Please see the tabulated template for tables and values used for this test.

The TOV solution agrees with the values from the `NRPy` TOV solution to around 10^{-8} in relative error among all values. This is error is a bit higher than the previous two EOS types since the adaptive methods used in `NRml_TOV` is more closely aligned with the GSL solver (as OdieGM is meant to be GSL's drop-in replacement[1]), while the `NRPy` solver uses the scipy integrator.[3] Because these adaptive methods will differ, interpolation error occurs, leading to a little higher error.

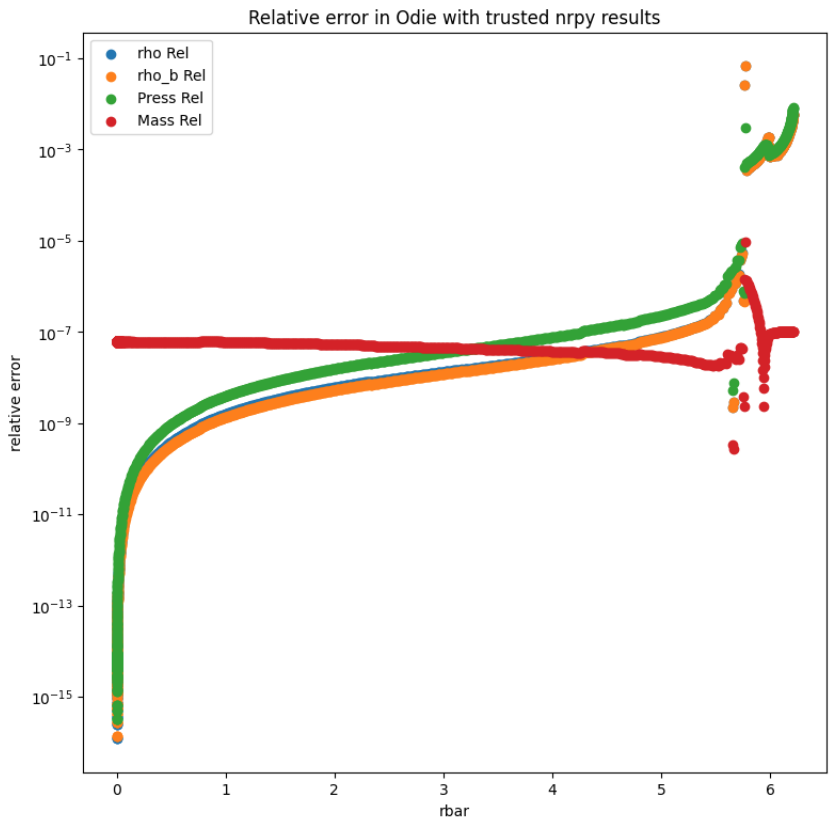

Figure 3: Relative error between `NRml_TOV` and `NRPy` TOV solutions. Most error around 10^{-8}

As a final note, you may notice a spike in error near the surface of the star. This is to be expected, because at the surface, pressure and densities go to zero. Based on the earlier definition of relative error, this will cause the error to blow up, even if the absolute error is substantially small.

6 Acknowledgements

I would like to acknowledge the `OdieGM` ODE solver by Gabriel Steward, as it was the foundation of `NRml_TOV`. `OdieGM` was made to be a drop-in replacement to the `GSL` solver. Each of the files were originally code from `OdieGM`, just edited to handle the TOV equations specifically for several different EOS types.[1]

I also want to acknowledge the `GRHayL` library. `GRHayL` was used to handle all of the EOS initializations and calculations in the code.[2]

References

[1] G. Steward. Github repository: https://github.com/GMBlackjack/OdieGM

[2] S. Cupp, L. Werneck, T. Pierre Jacques, Z. Etienne. In Prep (6/24). Github repository: https://github.com/GRHayL/GRHayL

[3] Z. Etienne. Github repository: https://github.com/zachetienne/nrpytutorial

[4] R. C. Tolman, Phys. Rev. **55**, 364 (1939).

[5] J. R. Oppenheimer and G. Volkoff, Physical Review **55**, 374 (1939).