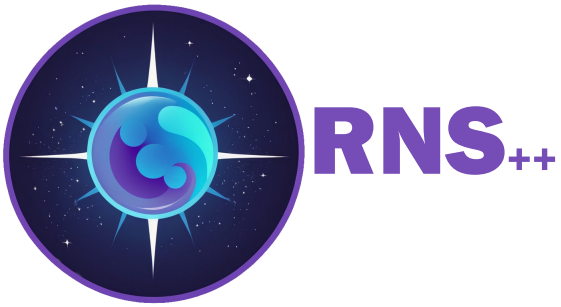

RNS$_{++}$
Technical Document

Lorenzo Cipriani

Contents

1 Introduction

The RNS code is a numerical tool designed to compute equilibrium models of rotating neutron stars in full general relativity. Originally based on the Komatsu–Eriguchi–Hachisu (KEH) scheme, the code solves the Einstein field equations under the assumptions of stationarity and axisymmetry. The code has been extended to incorporate differential rotation laws and multi-fluid systems (e.g., baryonic matter admixed with dark matter). In all cases, the goal is to obtain self-consistent solutions for the metric potentials and the matter distribution that satisfy both the gravitational field equations and the relativistic hydrostatic equilibrium.

1.1 Mathematical Model

In the RNS code the equilibrium configuration is described using quasi-isotropic coordinates, where the line element is written as

$$ds^2 = -e^{\gamma+\rho}\,dt^2 + e^{2\alpha}\,(dr^2 + r^2\,d\theta^2) + e^{\gamma-\rho}\,r^2\sin^2\theta\,(d\phi - \omega\,dt)^2. \tag{1}$$

Here, the metric potentials γ, ρ, α, and the frame-dragging potential ω are functions of the radial coordinate r and the polar angle θ. The proper circumferential radius is recovered via

$$R = r\,e^{(\gamma-\rho)/2}. \tag{2}$$

The neutron star is modelled as a perfect fluid with the energy-momentum tensor

$$T^{\mu\nu} = (\epsilon + P)\,u^\mu u^\nu + P\,g^{\mu\nu}, \tag{3}$$

where ϵ is the energy density, P is the pressure, and $u^\mu = \frac{e^{(\gamma+\rho)/2}}{\sqrt{1-v^2}}\,(1,0,0,\Omega)$ is the fluid's four-velocity. For barotropic fluids, the pressure is a unique function of the energy density, and the specific enthalpy is defined as $h = \frac{\epsilon+P}{\rho}$, with ρ being the rest-mass density.

In cases where additional components are considered (such as a dark-matter fluid), the total energy-momentum tensor is written as a sum (e.g., $T^{\mu\nu}_{\rm tot} = T^{\mu\nu}_{\rm BM} + T^{\mu\nu}_{\rm DM}$) with each fluid satisfying its own conservation law.

Having assumed stationarity and axisymmetry, the conservation of the energy-momentum tensor, $\nabla_\mu T^{\mu\nu} = 0$, leads to the relativistic Euler equation. The first integral of hydrostatic equilibrium can be written in the compact form

$$H - \ln u^t + \int_0^{\tilde{j}} \tilde{j}\,\frac{\mathrm{d}\Omega}{\mathrm{d}\tilde{j}}\mathrm{d}\tilde{j} = \text{constant}, \tag{4}$$

where H is the integrated enthalpy defined as

$$H(P) = \int_0^P \frac{dP'}{\epsilon(P') + P'}, \tag{5}$$

In Eq. (4), $j(\Omega) = u^t u_\phi$ is the gravitationally redshifted specific angular momentum and Ω the local angular velocity of the fluid. The relation $j(\Omega)$ specifies the rotational profile of the star: for uniformly rotating models, Ω is constant so the integral term vanishes, while for differentially rotating models this term encodes the chosen rotation law.

1.2 Differential rotation

Originally, the RNS code was built to handle uniform rotation; however, many astrophysical scenarios require differential rotation. One common choice is the "j-constant" (KEH) law, which imposes a relation such as

$$j(\Omega) = A^2(\Omega_c - \Omega), \tag{6}$$

with A a parameter that sets the length scale over which the angular velocity Ω declines from its central value Ω_c. Extensions of the code have implemented more versatile laws as the Uryù law, which parametrizes the angular velocity profile as

$$\Omega(j) = \Omega_c \frac{1 + \left(\frac{j}{A^2\Omega_c}\right)^p}{1 + \left(\frac{j}{B^2\Omega_c}\right)^{p+q}}, \tag{7}$$

where:

- Ω_c is the central angular velocity,
- A and B are length-scale parameters,
- p controls the decline of the angular velocity near the rotation axis,
- q sets the asymptotic, Keplerian fall-off,

In these formulations the integral in the first integral of the Euler equation can be computed either analytically (for selected choices of p and q) or numerically integrated.

Another possible choice is

$$\Omega(j) = \Omega_c \left[1 + \left(\frac{j}{B^2\Omega_c}\right)^p\right] \left[1 - \left(\frac{j}{A^2\Omega_c}\right)^\xi\right] \tag{8}$$

that more faithfully represents the angular profile in the post merger as recovered from numerical simulations.

1.3 Numerical Algorithm

The numerical algorithm implemented in the RNS code consists of the following steps:

1. **Initial Setup**: The computational domain is set up on a grid defined in quasi-isotropic coordinates, where radial and angular coordinates (i.e., $s = r/(r_e + r)$ and $\mu = \cos\theta$) are used to cover the star and its exterior. The equation of state is read from the table, if provided.

2. **Initial Guess:**
   The process begins by computing a nonrotating, spherically symmetric solution of the Tolman-Oppenheimer-Volkoff equations. This solution provides initial profiles for the metric potentials γ, ρ, α, ω and the matter variables (pressure, density, and enthalpy).

3. **Hydrostationary Equilibrium Calculation:**
   Using the first integral of the hydrostatic equilibrium (4), the code computes the fluid variables at each grid point. In the differential rotation case, this step involves evaluating first the integral term based on the chosen rotation law, ensuring that the local angular velocity distribution $\Omega(r, \theta)$ is consistent with the fluid's enthalpy and pressure profiles.

   Determination of Angular Velocity:
   The angular velocity Ω is not known a priori throughout the star. For the Uryù laws (7) and (8), the parameters are updated dynamically by solving for the scaling parameters (typically through prescribed ratios such as $\lambda_1 = \Omega_{\max}/\Omega_c$ and $\lambda_2 = \Omega_e/\Omega_c$) to match the desired rotation profile extracted from simulation data. The code determines:

   - The equatorial angular velocity Ω_e by equating the hydrostatic equilibrium integral at the equator and at the pole.
   - The central angular velocity Ω_c, as set by the rotation law.
   - At each grid point, the local value of Ω is obtained by solving the equation $j(\Omega) = u^t u_\phi$ (or its inverse, depending on the rotation law) using robust root-finding methods (e.g., Brent's algorithm).

4. **Update of Metric Potentials:**
   With the updated matter distribution, the Einstein field equations are recast into a set of elliptic equations using the KEH formulation. These equations are solved via Green's function techniques to update the metric potentials γ, ρ, α, and ω on the grid.

5. **Iterative Loop and Convergence Check:**
   The algorithm iterates over the following cycle:

   (a) Recompute the angular velocity distribution $\Omega(r, \theta)$.

   (b) Update fluid variables from the hydrostatic equilibrium integral.

   (c) Solve the metric equations to update the potentials.

   (d) Check convergence by monitoring changes in the equatorial radius r_e.

   Iteration continues until the changes fall below a prescribed tolerance.

6. **Computation of Global Quantities:**
   Once convergence is reached, integrated quantities are computed. Analytical expressions for these quantities are reported in Appendix A.

2 Implementation of the `main()` driver

The `main()` function serves as the high-level driver that glues together the core computational routines `sphere`, `spin` and `mass_radius` (the latter computes the macroscopic quantities of the model). It begins by including the necessary headers:

```
#include "equil.h"
#include "equil_util.h"
#include "nrutil.h"
#include "output.h"
#include "parfile.h"
```

and defining the principal data structures: `struct EOS` for equation-of-state parameters, `struct stellar_properties` for central densities and rotation flags, `struct evolution_variables` for the 2D metric and fluid fields, and `struct DiffRotParams` for the differential-rotation profile.

Next, default values are assigned for tolerances (`accuracy`, `cf`), dark-matter fraction (`fdm_target`), central energy density (`star_props.e_center`), and

differential-rotation constants. A `getopt` loop detects a `-c <config>` option: without it, the program writes a template `config.d` via `writeConfig(...)` and exits; with it, it reads back all parameters using `readConfig(...)`.

If the EOSs are tabulated, the code then calls `load_eos()` to read them. It then builds the angular grid $\{s_i, \mu_j\}$ with `make_grid(s_gp,mu)` and allocates the solver workspace arrays through `allocate_evolution_variables()`.

With initialization complete, the driver computes a non-rotating "seed" model by setting the central values via `make_center(...)` and invoking

```
sphere(s_gp, &eosBM, &star_props, &evolution_functions,
       &r_e_BM, &s_e_BM, &eosDM, &DM_props,
       &r_e_DM, &s_e_DM, &massBM, &massDM);
```

This yields the baseline equatorial radii and masses for both baryonic and dark components. In the special case $\mathbf{DM\_fraction} = 0$, the dark matter fields are explicitly zeroed before proceeding.

The core of the driver is an iteration loop (`while iter < MAXIT`) that adjusts either the axis ratios `r_ratio_BM`, `r_ratio_DM` or the DM central density to satisfy two convergence criteria: matching the target spin flattening and the target dark-matter mass fraction. Within each iteration, one or more calls to

```
spin(s_gp, mu, &eosBM, &star_props, &evolution_functions,
     accuracy, cf, r_ratio_BM, &r_e_BM, &Omega_BM,
     s_e_BM, &eosDM, &DM_props, r_ratio_DM,
     &r_e_DM, &Omega_DM, s_e_DM, verbose, &outOfiter,
     counter, &Omega_e_BM, &DiffRotBM, &Omega_e_DM,
     &DiffRotDM, zero);
```

brackets and refines the axis ratios until they lie within prescribed tolerances. Optionally, this update can be replaced by a root-finding subroutine (e.g. Newton–Raphson) that calls `spin` as a black-box. A built-in "parabola" weighting of the differential-rotation profile is applied in the sample to enhance stability, but users can remove or generalize this heuristic.

Once both spin and mass fraction targets are met (or `iter==MAXIT`), the driver computes all global diagnostics via

```
mass_radius(s_gp, mu, &eosBM, &evolution_functions,
            r_ratio_BM, r_e_BM, Omega_BM,
            &Mass_BM, &Mass_0_BM, &J_BM, &R_e_BM,
```

```
... , &Mass_DM, &Mass_0_DM, &J_DM,
&R_e_DM, ...);
```

which returns ADM mass, baryonic/rest mass, angular momentum, equatorial radius, Kepler frequency, kinetic/potential energies, and the final dark-matter fraction fdm = $M_{\rm DM}/(M_{\rm BM} + M_{\rm DM})$.

Finally, the results are printed to `stdout` for human inspection, written to a data file via `print_output(...)`, and, if enabled, exported into HDF5 2D snapshots and 1D profiles.

All along, the user may wrap the seed–spin–compute block in external loops over central density or EOS files to perform parameter sweeps, or replace the internal ratio updates with custom root-finding calls. By following this workflow, one obtains a modular, extensible driver where `sphere`, `spin` and `mass_radius` form the only building blocks users need to explore arbitrary regions of parameter space.

2.1 List of main drivers:

- **RNS_Diff_one**: Standard driver, computes one model for a specified r_ratios and energies.
- **RNS_Diff_Jc**: Computes a model at a fixed total angular momentum J_{tot}, with a fixed percentage of it in J_{DM}.
- **RNS_Diff_BM**: Computes a model at a fixed baryonic angular momentum J_{BM}, with a fixed percentage of J_{tot} in J_{DM}.

3 Description of the input file

The input file can be generated by running the main program without any additional command line arguments. This file will be created in the current directory and named "`config.d`". It contains all the parameters that can be passed to the algorithm, including their names, default values, and a brief description, as detailed below. To add extra parameters, the appropriate routines in the source file "`src/parfile.c`" should be modified.

Please note that the parameters are not validated for consistency at the start of the run. It is the user's responsibility to ensure that the input values are

correct. Parameters whose name contains a $\mathcal{X}$ are present for both the BM and DM and are set substituting it with the appropriate label.

- **id_file** (String): Path to a .h5 file containing the initial data for the run, useful to skip repeated restarts.
- **EoS_$\mathcal{X}$_type** (Integer): Accepted Values: 0 →polytropic, 1 →tabulated. Description: Type of EoS for BM.
- **EoS_$\mathcal{X}$_file** (String): Description: Path to the EoS file, required only if **EoS_$\mathcal{X}$_type** = 1.
- **EoS_$\mathcal{X}$_N** (Float): Accepted Values: $\mathbb{R}_+$. Description: Expontent for polytropic EoS, $P \propto \rho^{(1+1/N)}$, required if **EoS_$\mathcal{X}$_type** = 0.
- **accuracy** (Float): Accepted Values: < 1. Description: Accuracy goal for the simulation.
- **$\mathcal{X}$_central_energy** (Float): Accepted Values: $\mathbb{R}_+$. Description: Central energy density for $\mathcal{X}$. Its use depends on the actual implementation of the main program (see Sec. 2). It is expressed in g cm^{-3} $\times$ 10^{-15} (e.g., to input a central energy density of 2×10^{15}g cm^{-3} the parameter **$\mathcal{X}$_central_energy** must be set equal to 2).
- **DM_particle_mass** (Float): Accepted Values: $\mathbb{R}_+$. Description: Mass of the DM particle in MeV.
- **DM_fraction** (Float): Accepted Values: < 1. Description: Target dark matter fraction. If set to zero, the code will solve the TOV equations for some small value of **DM_central_energy** and then set all DM fields to zero.
- **DM_fraction_tol** (Float): Accepted Values: < 1. Description: Tolerance in determination of the target dark matter fraction.
- **$\mathcal{X}$_rotation_type** (Integer): Type of rotation for $\mathcal{X}$. Accepted Values: 0 →Uniform, 1 →J constant, 2 →Uryu 8, 3 →Uryu Extended. Description: Type of rotation for $\mathcal{X}$.
- **$\mathcal{X}$_A** (Float): Accepted Values: $\mathbb{R}_+$. Description: Uryu parameter A. Required if **$\mathcal{X}$_rotation_type** = 1, 2, 3.
- **$\mathcal{X}$_B** (Float): Accepted Values: $\mathbb{R}_+$. Description: Uryu parameter B. Required if **$\mathcal{X}$_rotation_type** = 2, 3.

- $\mathcal{X}$**_lambda1** (Float): Accepted Values: > 1. Description: Set the ratio $\Omega_{\rm equator}/\Omega_{\rm central}$. This should be automatically satisfied by valid models. Required if $\mathcal{X}$**_rotation_type** = 2, 3.
- $\mathcal{X}$**_lambda2** (Float): Accepted Values: $\mathbb{R}_+$. Description: Set the ratio $\Omega_{\rm max}/\Omega_{\rm central}$. This is imposed during the model convergence. Required if $\mathcal{X}$**_rotation_type** = 2, 3.
- $\mathcal{X}$**_p** (Float): Accepted Values: $\mathbb{R}_+$. Description: Uryu parameter p. Required if BM_rotation_type = 3.
- $\mathcal{X}$**_csi** (Float): Accepted Values: $\mathbb{R}_+$. Description: Uryu parameter csi. Required if BM_rotation_type = 3.
- **counter_rotation** (Integer): Accepted Values: 0 →Corotating configuration, 1 →Counter-rotating configuration. Description: Set whether DM counter-rotates with respect to the BM.
- **r_ratio_**$\mathcal{X}$ (Float): Accepted Values: < 1. Description: Desired ratio of polar and equatorial radius for $\mathcal{X}$. Actual usage depends on implementation of the main driver.
- **r_step** (Float): Accepted Values: $<$ 1- Description: Step-size used in the algorithm to reach desired **r_ratio_**$\mathcal{X}$. Actual usage depends on implementation of the main driver.
- **1Doutput** (Integer): Accepted Values: 0 →no, 1 →yes. Description: Enable 1D output.
- **2Doutput** (Integer): Accepted Values: 0 →no, 1 →yes. Description: Enable 2D output.
- **output_name** (String): Name of the .dat and .h5 output files. Default is auto. Do not specify the extension!
- **verbose** (Integer): Accepted Values: 0-4. Description: Enable different levels of verbosity.

Note that to compute a model without the dark matter fluid, it is needed to set the following parameters:

- **DM_fraction** = 0

The code will automatically set the following:

- **DM_central_energy** = 1e-3;
- **DM_rotation_type** = 0;
- **r_ratio_DM** = 1;

Then it will compute the initial TOV solution for 2 fluids, set all matter fields to zero and skip all computations related to the second fluid.

4 Description of output

The output file module is designed to manage simulation data output, supporting both plain text and HDF5 formats. This section provides a thorough description of its components and functionality.

4.1 Text File Output

The module defines routines for creating and writing formatted text files containing simulation results. The main functions include:

- **File Initialization:** A function is provided to open a file for writing. It attempts to create a new file and, upon failure (e.g., if the file cannot be opened), it terminates the program after reporting an error. At the time of file creation, a header line is written. This header lists all the output fields such as central energy densities, equatorial and polar radii, masses normalized by the solar mass, various angular velocities, and additional astrophysical parameters.
- **Data Output:** A dedicated function outputs the simulation data in a tabulated format. It employs a strict numerical format (using high-precision formatting specifiers) to maintain uniformity of output across all columns. Variables are carefully scaled and converted as necessary before being written. The list of saved quantities is as follows:
    1. **BM_e_center** – Central energy density of the baryonic component.
    2. **R_e_BM (km)** – Equatorial radius of the baryonic matter, in kilometers.
    3. **R_p_BM (km)** – Polar radius of the baryonic matter, in kilometers.

4. **Mass_BM ($M_\odot$)** – Gravitational mass of the baryonic matter, in solar masses.
5. **Mass_0_BM ($M_\odot$)** – Rest mass (baryonic mass) of the baryonic component, in solar masses.
6. **DM_e_center** – Central energy density of the dark matter component.
7. **R_e_DM (km)** – Equatorial radius of the dark matter component, in kilometers.
8. **R_p_DM (km)** – Polar radius of the dark matter component, in kilometers.
9. **Mass_DM ($M_\odot$)** – Gravitational mass of the dark matter component, in solar masses.
10. **Mass_0_DM ($M_\odot$)** – Rest mass of the dark matter component, in solar masses.
11. **Mtot ($M_\odot$)** – Total gravitational mass of the system (BM + DM), in solar masses.
12. **fdm** – Fraction of the total mass composed of dark matter, defined as $M_{\rm DM}/M_{\rm tot}$.
13. **Omega_BM (Hz)** – Angular velocity of the baryonic matter, in hertz. Meaningful only for the Uniform rotation profile.
14. **Omega_DM (Hz)** – Angular velocity of the dark matter, in hertz. Meaningful only for the Uniform rotation profile.
15. **J_BM**/$M_{\rm BM}^2$ – Dimensionless spin parameter of the baryonic component.
16. **J_DM**/$M_{\rm DM}^2$ – Dimensionless spin parameter of the dark matter component.
17. **r_ratio_BM** – Axis ratio (polar to equatorial) of the baryonic component.
18. **r_ratio_DM** – Axis ratio of the dark matter component.
19. **Omega_K_BM (Hz)** – Keplerian (mass-shedding) angular velocity of the baryonic matter, in hertz.

20. **Omega_K_DM (Hz)** – Keplerian angular velocity of the dark matter component, in hertz.
21. **T_BM** – Rotational kinetic energy of the baryonic component.
22. **T_DM** – Rotational kinetic energy of the dark matter component.
23. **W_BM** – Gravitational potential energy of the baryonic component.
24. **W_DM** – Gravitational potential energy of the dark matter component.

- **Closing the File:** Finally, a utility function ensures that the file is properly closed once all writing operations are complete, ensuring that all data is flushed and the file is in a consistent state.

4.2 HDF5 Data Handling

The code also integrates the HDF5 library for managing more complex data storage, which involves both grid data and metadata. Two primary routines are responsible for saving and reading data:

- **Saving Simulation Data:**
  - The HDF5 save routine first creates a new file using default properties.
  - It then constructs an array of attributes that capture essential simulation parameters. These attributes include information about the equation of state, stellar structure properties (such as central energy densities), and rotation parameters.
  - In addition to attributes, the function writes out grid data (for spatial coordinates) as well as two-dimensional arrays that represent various physical quantities (for example, potential, energy density, pressure, and angular velocities).
  - Each dataset is accompanied by descriptive metadata which annotates the physical meaning of the stored variable.
- **Reading Simulation Data:**
  - The corresponding read routine opens an existing HDF5 file in read-only mode.

  - It retrieves all stored attributes to restore simulation parameters, including converting string representations of differential rotation types into their internal enumerated form.
  - Grid point arrays and datasets are then read into memory, reconstructing two-dimensional simulation data structures that are used for subsequent analysis.

4.3 Profile Data Export

An auxiliary function is included to generate radial profiles for selected simulation variables. Its operation involves:

- Checking for an existing file and safely removing it if necessary.
- Creating a two-column output where the first column represents the radial coordinate (derived from the simulation grid) and the second contains the corresponding variable value.
- Iterating over the grid resolution to output data with consistent precision.

A Analytical expressions of global quantities

Given $r_e = \max\left(r_e^{BM}, r_e^{DM}\right)$, the differentials are:

$$\begin{cases} \mathrm{d}r &= r_e \frac{\mathrm{d}s}{(1-s)^2} \\ \mathrm{d}\theta &= -\frac{\mathrm{d}\mu}{\sin\theta} \end{cases}$$

For i = BM, DM:

- Circumferential radius:

$$R_e^i = r_e^i \ \mathrm{e}^{\frac{\gamma_e^i - \rho_e^i}{2}}$$

$$\gamma_e^i = \gamma(s = s_e^i, \mu = 0), \quad \rho_e^i = \rho(s = s_e^i, \mu = 0)$$

- Gravitational mass:

$$\begin{aligned} M_i &= \int_0^\infty \mathrm{d}r \int_0^\pi \mathrm{d}\theta \int_0^{2\pi} \mathrm{d}\phi \left(-2T^0_{i\ 0} + T^\mu_{i\ \mu}\right)\sqrt{-g} \\ &= 4\pi \int_0^1 r_e \frac{\mathrm{d}s}{(1-s)^2}\left(r_e\frac{s}{1-s}\right)^2 \int_0^1 \mathrm{d}\mu\ \mathrm{e}^{2\alpha+\gamma}\left[\frac{e_i + P_i}{1-v_i^2}\times \right. \\ &\qquad \left. \left(1 + v_i^2 + 2\omega r_e \frac{s}{1-s} v_i \sqrt{1-\mu^2}\ \mathrm{e}^{-\rho}\right) + 2P_i\right] \end{aligned} \tag{9}$$

- Rest mass:

$$M_i^0 = 4\pi \int r_e \frac{\mathrm{d}s}{(1-s)^2}\left(r_e\frac{s}{1-s}\right)^2 \int \mathrm{d}\mu\ \mathrm{e}^{2\alpha + \frac{\gamma-\rho}{2}} \frac{\rho_i^0}{\sqrt{1-v_i^2}}$$

$$\rho_i^0 = (e_i + P_i)\ \mathrm{e}^{-h_i} \text{ or interpolation from table}$$

- Angular momentum:

$$J_i = 4\pi \int r_e \frac{\mathrm{d}s}{(1-s)^2}\left(r_e\frac{s}{1-s}\right)^3 \int \mathrm{d}\mu\ \mathrm{e}^{2\alpha+\gamma-\rho}(e_i + P_i)\frac{v_i}{1-v_i^2}\sqrt{1-\mu^2}$$

- Rotational kinetic energy:

$$T_i = \frac{1}{2} J_i * \Omega_i$$

- Gravitational binding energy:

$$W_i = M_i^p - M_i + T_i$$

where

$$M_i^p = 4\pi \int r_e \frac{\mathrm{d}s}{(1-s)^2} \left(r_e \frac{s}{1-s} \right)^2 \int \mathrm{d}\mu \ \mathrm{e}^{2\alpha + \frac{\gamma-\rho}{2}} \frac{e_i}{\sqrt{1-v_i^2}}$$
$$\equiv \int\int\int \mathrm{d}\phi \mathrm{d}r \mathrm{d}\theta \sqrt{g} W e_i$$

- Velocity of co-rotating and counter-rotating particles with respect to ZAMO

$$v_\pm = \frac{\pm\sqrt{v} + r_e s^2 \ \mathrm{e}^{-\rho} \partial_s \omega}{2 + (1-s)(\partial_s \gamma - \partial_s \rho)}$$
$$\sqrt{v} = r_e^2 s^4 \ \mathrm{e}^{-2\rho} \left(\partial_s \omega\right)^2 + 2(1-s)\left[\partial_s \gamma + \partial_s \rho - (\partial_s \rho)^2\right]$$

- Kepler angular velocity:

$$\Omega_K = \omega_e + \frac{v_e^K}{r_e^{\mathrm{BM}}} \ \mathrm{e}^{\rho_e}$$

Defining

$$\mathcal{Y} = \frac{s_{e\mathrm{BM}}^2 \ \partial_s \omega|_{s_e}}{2 + (\partial_s \gamma|_{s_e} - \partial_s \rho|_{s_e}) s_{e\mathrm{BM}} (1 - s_{e\mathrm{BM}})} r_e \ \mathrm{e}^{-\rho}$$

we have

$$v_e^K = \mathcal{Y} + \sqrt{\mathcal{Y} \frac{(1 - s_{e\mathrm{BM}})(\partial_s \gamma|_{s_e} + \partial_s \rho|_{s_e})}{\partial_s \omega|_{s_e}} + \mathcal{Y}^2}$$