QLIMR Module

MUSES Meeting 2023

Carlos A. Conde O.*, Hung Tan* and Nicolás Yunes*

*Department of Physics
University of Illinois at Urbana-Champaign
Illinois Center for the Advanced Studies of the Universe

muses | QLIMR

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Introduction: *What do we want to solve?*

Metric ansatz based on the Hartle-Thorne procedure to generate slowly rotating neutron stars [1, 3]

$$ds^2 = -e^{\nu}[1+2h_2]\,dt^2 + e^{\lambda}\left[1+\frac{2m_2}{r-2M}\right]dr^2 + r^2\,(1+2k_2)\left\{d\theta^2 + \sin^2\theta\,[d\phi - (\Omega-\omega_1)\,dt]^2\right\} \quad (1)$$

Introduction: *What do we want to solve?*

Metric ansatz based on the Hartle-Thorne procedure to generate slowly rotating neutron stars [1, 3]

$$ds^2 = -e^{\nu}[1+2h_2]\,dt^2 + e^{\lambda}\left[1+\frac{2m_2}{r-2M}\right]dr^2 + r^2\,(1+2k_2)\left\{d\theta^2 + \sin^2\theta\,[d\phi - (\Omega-\omega_1)\,dt]^2\right\} \quad (1)$$

TOV equations: $\mathcal{O}(1)$

$$\frac{dm}{dr} = 4\pi r^2\varepsilon \quad ; \quad \frac{dp}{dr} = -(\varepsilon+p)\frac{m+4\pi r^3 p}{r(r-2m)} \quad ; \quad \frac{d\nu}{dr} = 2\frac{m+4\pi r^3 p}{r(r-2m)} \quad ; \quad p = p(\varepsilon) \quad (2)$$

Using Lindblom's method with $dh = dp/(\varepsilon+p)$ these equations can be rewritten as [2]

$$\frac{dm}{dh} = -\frac{4\pi\varepsilon r^3(r-2m)}{m+4\pi r^3 p} \quad ; \quad \frac{dr}{dh} = -\frac{r(r-2m)}{m+4\pi r^3 p} \quad ; \quad \frac{d\nu}{dr} = -2 \quad ; \quad p(h)\,,\,\varepsilon(h) \quad (3)$$

Introduction: *What do we want to solve?*

First Order: $\mathcal{O}(\Omega)$

$$\frac{d^2\omega_1}{dr^2} + 4\frac{1-\pi r^2(\varepsilon+p)e^{\lambda}}{r}\frac{d\omega_1}{dr} - 16\pi(\varepsilon+p)e^{\lambda}\omega_1 = 0 \tag{4}$$

Introduction: *What do we want to solve?*

First Order: $\mathcal{O}(\Omega)$

$$\frac{d^2\omega_1}{dr^2} + 4\frac{1-\pi r^2(\varepsilon+p)e^{\lambda}}{r}\frac{d\omega_1}{dr} - 16\pi(\varepsilon+p)e^{\lambda}\omega_1 = 0 \tag{4}$$

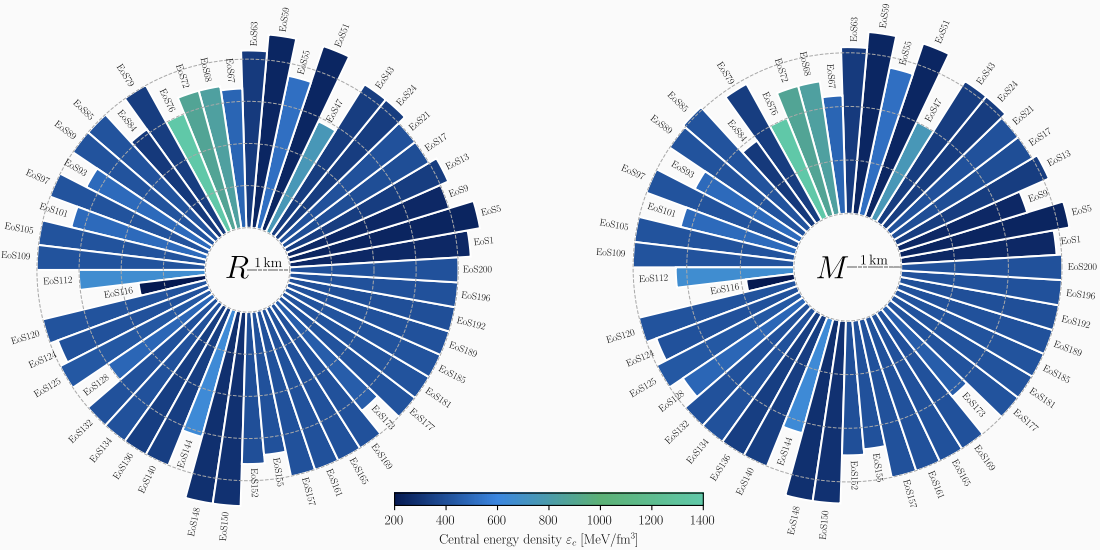
Second Order: $\mathcal{O}(\Omega^2)$

$$m_2 = -Re^{-\lambda}h_2 + \frac{1}{6}R^4 e^{-(\nu+\lambda)}\left[Re^{-\lambda}\omega_1'^2 + 16\pi R\omega_1^2(\varepsilon+p)\right]$$

$$k_2' = -h_2' + \frac{R-3M-4\pi pR^3}{R^2}e^{\lambda}h_2 + \frac{R-M+4\pi pR^3}{R^3}e^{2\lambda}m_2 \tag{5}$$

$$h_2' = -\frac{R-M+4\pi pR^3}{R}e^{\lambda}k_2' + \frac{3-4\pi(\varepsilon+p)R^2}{R}e^{\lambda}h_2 + \frac{2}{R}e^{\lambda}k_2 + \frac{1+8\pi pR^2}{R^2}e^{2\lambda}m_2 + \frac{R^3}{12}e^{-\nu}\omega_1'^2$$

Past Work: *Accuracy and precision*



Code development: *Inputs/Outputs*

Equation of state data table in *.dat* format

Energy density [MeV/fm^3] | Pressure [MeV/fm^3]

Variable	Description
eos_name	Equation of state name
r_start	Initial integration radius [km]
initial_epsilon	Initial ε_c [MeV/fm^3]
final_epsilon	Final ε_c [MeV/fm^3]
number_epsilons	Number of initial epsilons
resolution_in_R	Resolution in radius [km]
resolution_in_M	Resolution in mass [Msun]
compute_inertia	Moment of inertia (1 or 0)
compute_quadrupole	Quadrupole moment (1 or 0)
compute_love	Tidal love number (1 or 0)

Code development: *Inputs/Outputs*

Equation of state data table in *.dat* format

Energy density [MeV/fm^3] | Pressure [MeV/fm^3]

Variable	Description
eos_name	Equation of state name
r_start	Initial integration radius [km]
initial_epsilon	Initial ε_c [MeV/fm^3]
final_epsilon	Final ε_c [MeV/fm^3]
number_epsilons	Number of initial epsilons
resolution_in_R	Resolution in radius [km]
resolution_in_M	Resolution in mass [Msun]
compute_inertia	Moment of inertia (1 or 0)
compute_quadrupole	Quadrupole moment (1 or 0)
compute_love	Tidal love number (1 or 0)

Variable	Description
NS_eps	Central energy density
NS_R	Radius [km]
NS_M	Total Mass [$M_\odot$]
NS_I	Moment of inertia [-]
NS_L	Tidal Love number [-]
NS_Q	Quadrupole Moment [-]
r	Radial distance [km]
eps	Energy density [MeV/fm^3]
p	Pressure [MeV/fm^3]
m	Enclosed mass [$M_\odot$]
nu	Metric function $\mathcal{O}(1)$
omega1	Metric function $\mathcal{O}(\Omega)$
h2	Metric function $\mathcal{O}(\Omega^2)$
k2	Metric function $\mathcal{O}(\Omega^2)$

Code development: *.yaml files and adaptors*

```yaml
- E_dens: 295.1484
  P: 19.30304
  T: 0.0
  delta0: 0.0
  deltaF: 0.0
  mu_B: 1000.0
  mu_Q: 0.0
  mu_S: 0.0
  omega0: 70.0
  omegaF: 37.06624
  phi0: -50.0
  phiF: -4.20017
  polyakov0: 0.0
  polyakovF: 0.0
  rho0: 0.0
  rhoF: 0.0
  sigma0: -27.0
  sigmaF: -42.73479
  zeta0:-90.0
  zetaF:-97.15133
- ...
```

```
295.14840 19.30304
296.63980 19.61824
298.12770 19.93492
299.61220 20.25309
301.09320 20.57274
302.57110 20.89386
304.04570 21.21645
305.51730 21.54050
306.98590 21.86601
308.45160 22.19298
309.91440 22.52140
311.37450 22.85127
312.83190 23.18258
314.28670 23.51533
315.73900 23.84951
317.18880 24.18513
318.63620 24.52217
320.08130 24.86064
321.52420 25.20052
322.96480 25.54182
...
```

```
180.000 14.7756 0.226011
182.577 14.5810 0.232493
185.153 14.4015 0.239081
187.730 14.2359 0.245769
190.306 14.0828 0.252551
192.883 13.9418 0.259393
195.459 13.8145 0.266130
198.036 13.6992 0.272764
200.612 13.5937 0.279333
205.765 13.4059 0.292433
210.918 13.2411 0.305742
216.071 13.0928 0.319573
221.224 12.9566 0.334236
226.378 12.8299 0.350044
231.531 12.7113 0.367306
236.684 12.6003 0.386331
241.837 12.4968 0.407425
252.143 12.3146 0.457291
257.296 12.2408 0.485659
262.449 12.1806 0.515850
...
```

```yaml
- NS_eps: 180.000
  NS_R: 14.7756
  NS_M: 0.226011
- NS_eps: 182.577
  NS_R: 14.5810
  NS_M: 0.232493
- NS_eps: 185.153
  NS_R: 14.4015
  NS_M: 0.239081
- NS_eps: 190.306
  NS_R: 14.0828
  NS_M: 0.252551
- NS_eps: 192.883
  NS_R: 13.9418
  NS_M: 0.259393
- NS_eps: 195.459
  NS_R: 13.8145
  NS_M: 0.266130
- NS_eps: 198.036
  NS_R: 13.6992
  ...
```

Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

If $|R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\mathrm{user})}$

or $|M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\mathrm{user})}$

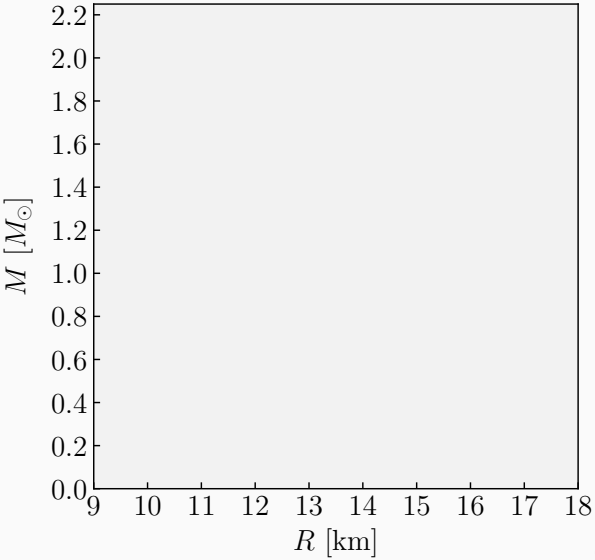
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



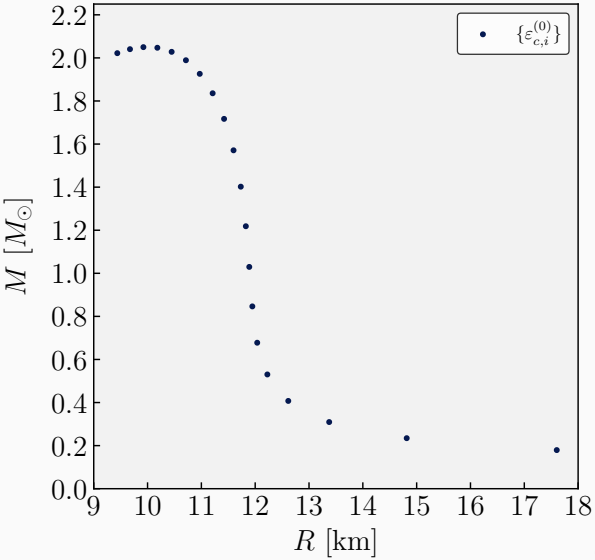
Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.
2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.
3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

   If $|R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\text{(user)}}$

   or $|M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\text{(user)}}$

   Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.
4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.
5) Stack A and B sets.
6) Sort with respect to ε_c.
7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

If $|R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\mathrm{user})}$

or $|M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\mathrm{user})}$

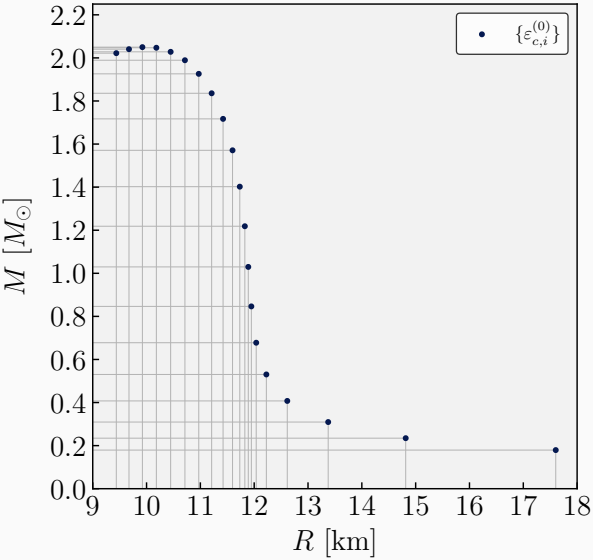
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

$$\text{If} \quad |R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\text{user})}$$

$$\text{or} \quad |M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\text{user})}$$

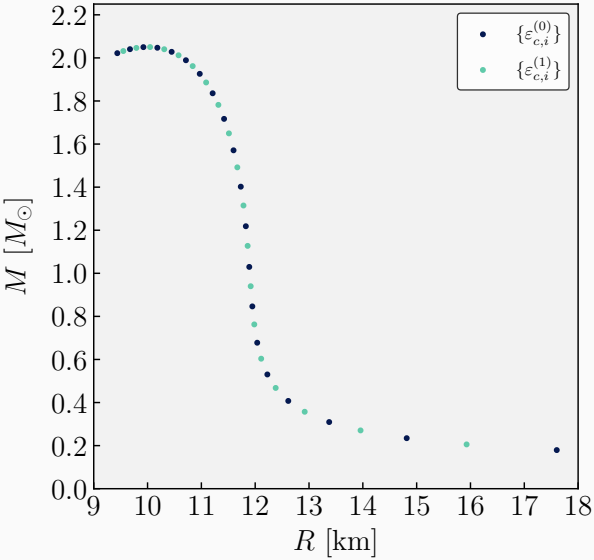
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

$$\text{If} \quad |R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\text{user})}$$

$$\text{or} \quad |M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\text{user})}$$

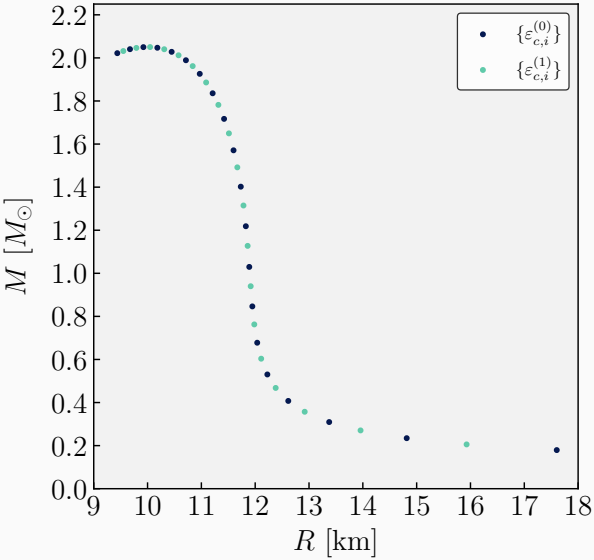
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

$$\text{If} \quad |R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\text{user})}$$

$$\text{or} \quad |M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\text{user})}$$

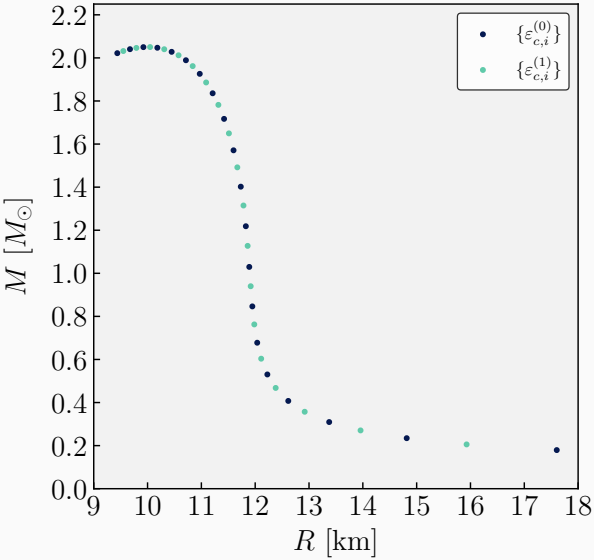
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

$$\text{If} \quad |R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\text{user})}$$

$$\text{or} \quad |M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\text{user})}$$

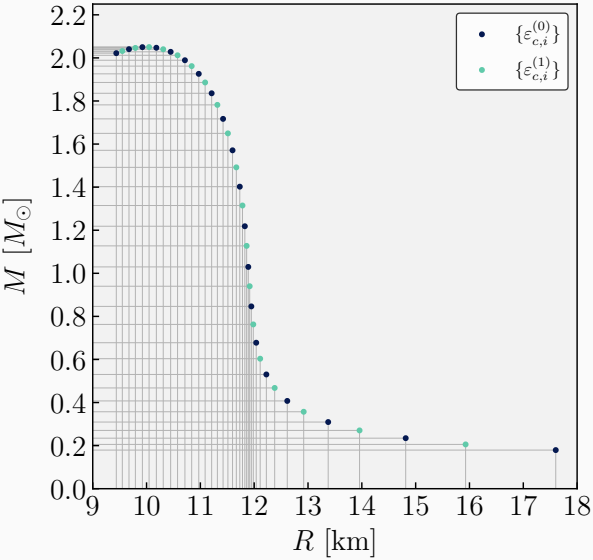
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

$$\text{If} \quad |R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\text{user})}$$

$$\text{or} \quad |M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\text{user})}$$

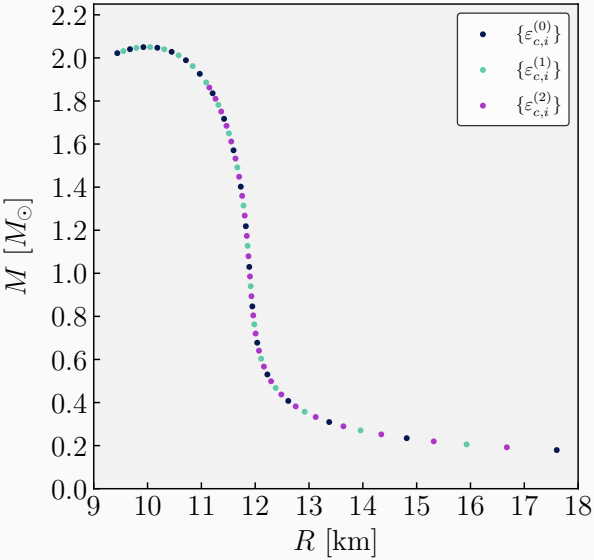
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

$$\text{If} \quad |R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\text{user})}$$

$$\text{or} \quad |M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\text{user})}$$

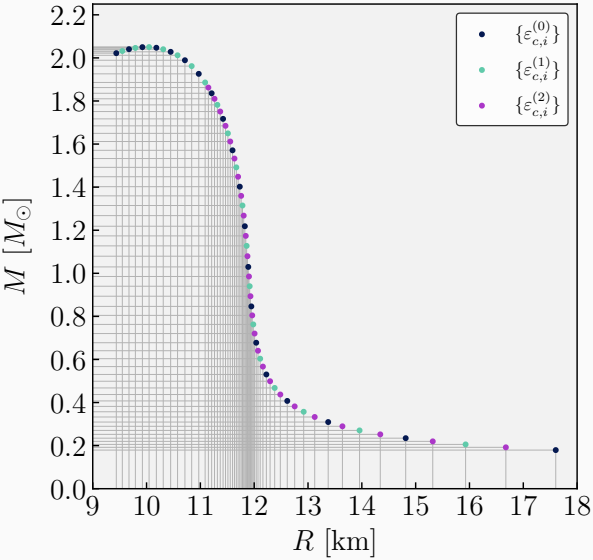
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

$$\text{If} \quad |R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\text{user})}$$

$$\text{or} \quad |M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\text{user})}$$

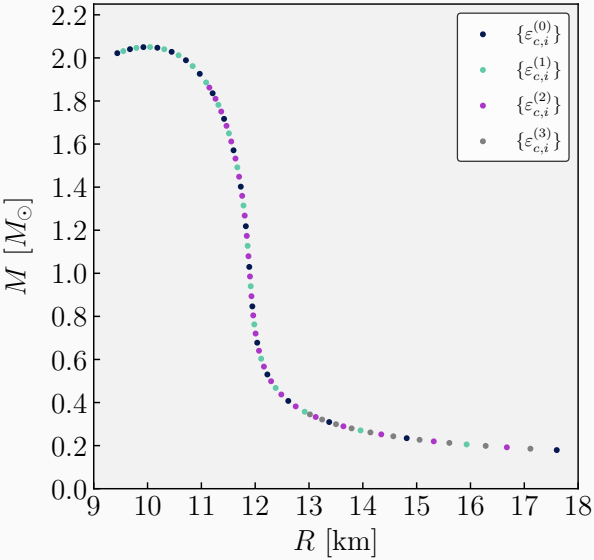
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *How does it work?*

1) Create an initial set $\{\varepsilon_{c,i}^{(0)}\}$.

2) Get observables $A = \{\varepsilon_{c,i}^{(0)}, R_{c,i}^{(0)}, M_{c,i}^{(0)}, ...\}$.

3) Check ΔR and ΔM and create $\{\varepsilon_{c,i}^{(1)}\}$ as:

$$\text{If} \quad |R(\varepsilon_{c,i+1}^{(0)}) - R(\varepsilon_{c,i}^{(0)})| > \Delta R^{\,(\text{user})}$$

$$\text{or} \quad |M(\varepsilon_{c,i+1}^{(0)}) - M(\varepsilon_{c,i}^{(0)})| > \Delta M^{\,(\text{user})}$$

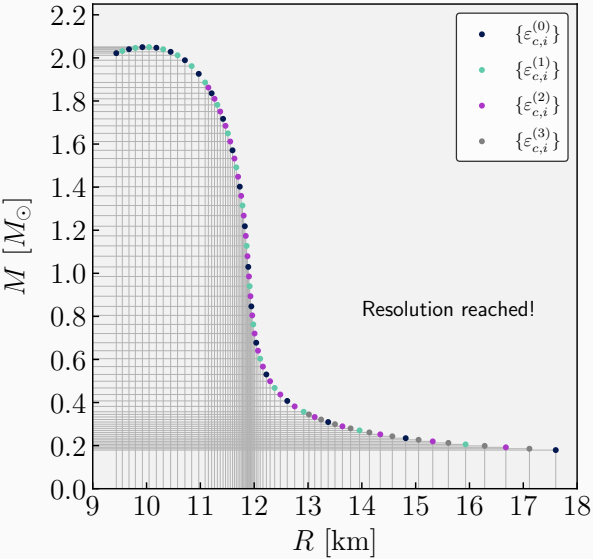
Then, $\varepsilon_{c,i}^{(1)} = (\varepsilon_{c,i+1}^{(0)} + \varepsilon_{c,i}^{(0)})/2$.

4) Get observables $B = \{\varepsilon_{c,i}^{(1)}, R_{c,i}^{(1)}, M_{c,i}^{(1)}, ...\}$.

5) Stack A and B sets.

6) Sort with respect to ε_c.

7) Repeat 3) - 6) until resolution is reached.



Code development: *Parallelization Strategy*

We use OpenMP library to parallelize the code. For each set j given by $\{\varepsilon_{c,i}^{(j)}\}$ each thread computes all requested observables.

T	ε_c [MeV/fm^3]	R [km]	M [M$_\odot$]	$\bar{I}$ [−]	$\bar{\lambda}$ [−]	$\bar{Q}$ [−]
T4	450.000000000000	11.738931318059	1.122726321261	16.596211781695	1171.90702190071	7.7199147067570
T2	350.000000000000	11.805581630043	0.788768009861	29.493662320058	7907.83655530857	13.541561068899
T6	550.000000000000	11.638637146876	1.392890083969	11.589811479256	308.627478502279	5.1005561024365
T3	400.000000000000	11.769751613092	0.962330844807	21.355592973080	2779.59357482180	10.006673956968
T1	300.000000000000	11.900411579063	0.608803557535	44.850072581395	28588.1555522672	19.297172752463
T5	500.000000000000	11.696254488416	1.266588439648	13.595409112033	568.854565643023	6.1774520489820

Check resolution, stack and sort functions are executed in serial.

Code development: *How to run it?*

To **clone** QLIMR repository

```
$ git clone
  https://gitlab.com/nsf-muses
  /ns-qlimr/qlimr.git
```

To **build** docker module *image*

```
$ docker build -t qlimr:dev .
```

To **run** docker container

```
$ docker run -it --rm --name qlimr
  -v $(pwd)/src:/home/qlimr/src
  -v $(pwd)/input:/home/qlimr/input
  -v $(pwd)/output:/home/qlimr/output
  qlimr:dev qlimr
```

Code development: *How to run it?*

To **clone** QLIMR repository

```
$ git clone
  https://gitlab.com/nsf-muses
  /ns-qlimr/qlimr.git
```

To **build** docker module *image*

```
$ docker build -t qlimr:dev .
```

To **run** docker container

```
$ docker run -it --rm --name qlimr
  -v $(pwd)/src:/home/qlimr/src
  -v $(pwd)/input:/home/qlimr/input
  -v $(pwd)/output:/home/qlimr/output
  qlimr:dev qlimr
```

Running module locally

To run the module locally the following C++ **libraries** need to be installed:

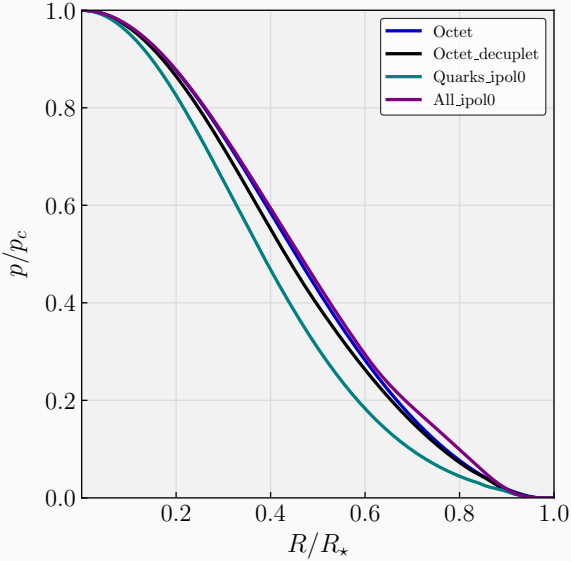
GSL, yaml and OpenMP.

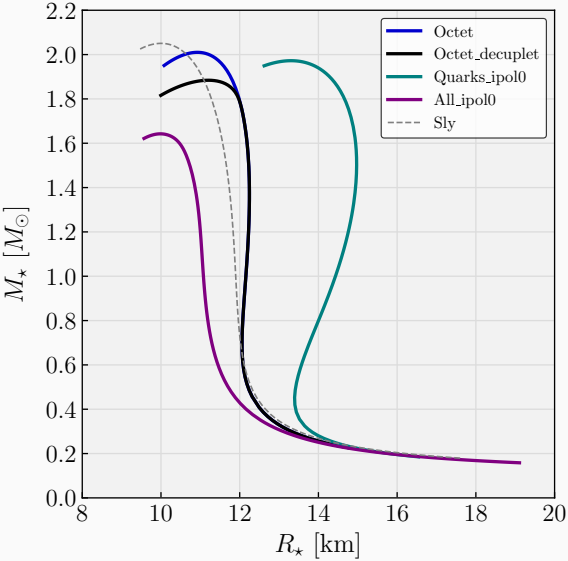
After cloning repository go to `src` folder. Then, to run only the source code execute

```
$ make redo
$ make run
```

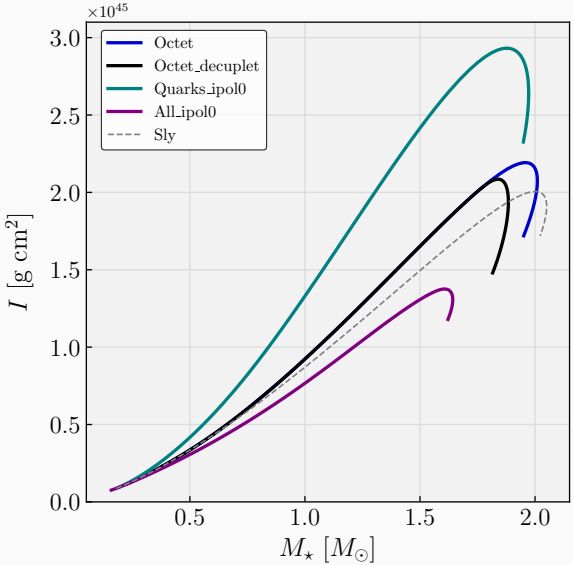
To see outputs open `output` folder.
To change inputs modify *config.yaml* file.
To run other EoS add *eos_name.dat* file inside `input` folder.

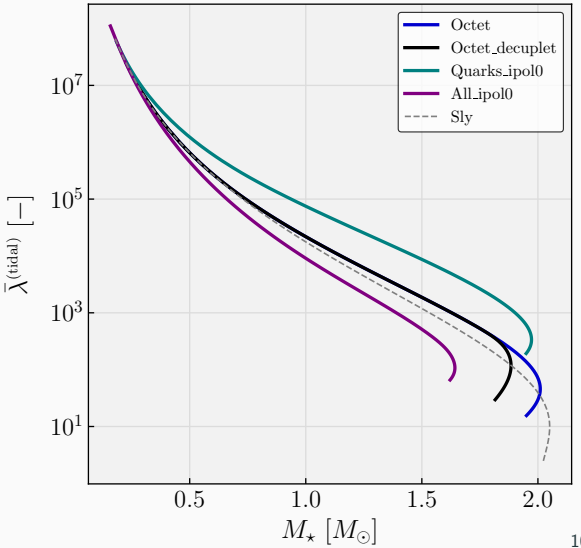
Results: *Plots*



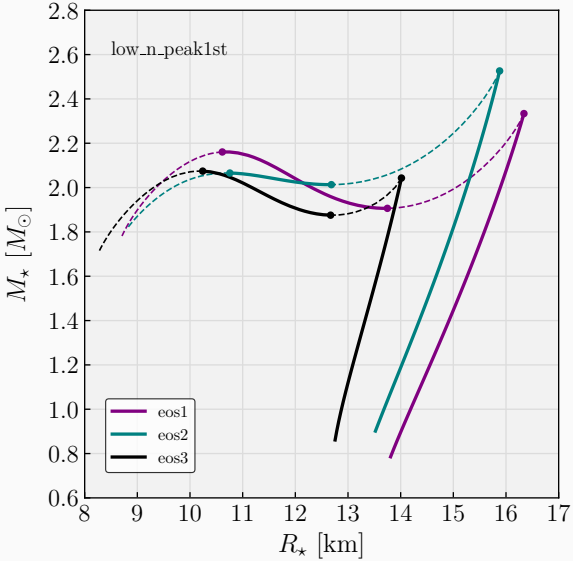


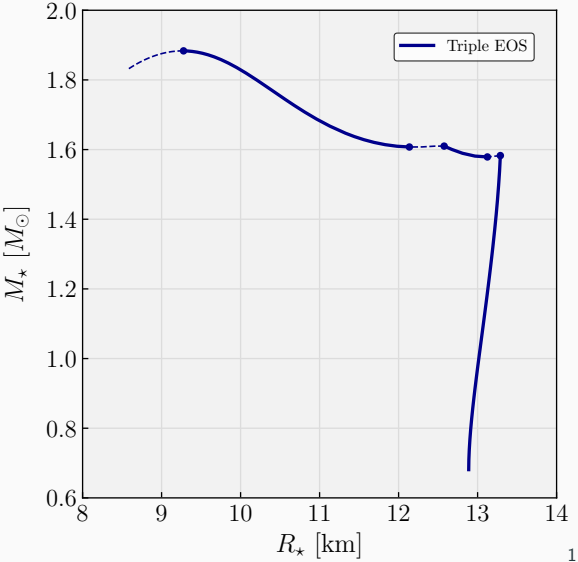
Results: *Plots*





Results: *Plots*





Future Work: *What remains to be done...*

Calculation tasks

- Output local functions into *.csv* files
- Solve $l = 0$ mode equations at $\mathcal{O}(\Omega^2)$ to find the shape of the star and the corrections to the mass and radius.
- Compute binary love relations $\Lambda_a(\Lambda_s)$.
- Implement local maxima and minima finder for MR curve in the C++ code.
- Complete Q calculation.
- Finish EoS matching crust function.
- Test module for EoS with phase transitions.

Future Work: *What remains to be done...*

Calculation tasks

- Output local functions into *.csv* files
- Solve $l = 0$ mode equations at $\mathcal{O}(\Omega^2)$ to find the shape of the star and the corrections to the mass and radius.
- Compute binary love relations $\Lambda_a(\Lambda_s)$.
- Implement local maxima and minima finder for MR curve in the C++ code.
- Complete Q calculation.
- Finish EoS matching crust function.
- Test module for EoS with phase transitions.

Cyberinfrastructure tasks

- Integrate module into the calculation engine (CE) workflow.
- Implement merge input function between *OpenAPI_Specifications.yaml* and *config.yaml*.
- Create a .sh script QLIMR workflow including python adaptors.
- Run CMF+QLIMR within the CE.
- Finish parallelization details.
- Store local functions output into a SQLite data base.

Conclusions

- For a given EoS table, the module is able to compute the mass M, the radius R, the moment of inertia I, the tidal love number $\bar{\lambda}$ and their corresponding local functions.
- The code is currently working in a docker container and in the campus cluster.
- Local tests with CMF module using *.yaml* files and python adaptors were successful.
- Parallelization strategy was tested to compute M and R.
- Integration of the module to the calculation engine is still in progress.

References

J. B. Hartle.
Slowly rotating relativistic stars. i. equations of structure.
The Astrophysical Journal, 150:1005, 1967.

L. Lindblom.
Determining the nuclear equation of state from neutron-star masses and radii.
The Astrophysical Journal, 398:569–573, 1992.

K. Yagi and N. Yunes.
I-love-q relations in neutron stars and their applications to astrophysics, gravitational waves, and fundamental physics.
Physical Review D, 88(2):023009, 2013.