Codes for the nuclear equation of state meta-modeling, neutron star EOS, TOV solver

J. Margueron

November 1, 2018

These notes describe the use of the different codes associated to the nuclear meta-modeling of dense matter [1, 2]. There are three different codes described here:

1. `metaEosMain.f95`: this code takes as inputs the empirical parameters and generates the dense matter EOS.

2. `nsEosMain.f95`: this code merges together a crust EOS with the meta-modeling for the core of the neutron star generated by `metaEosT0.f95`.

3. `tovMain.f95`: this code solves the TOV equations for a given EOS generated by `nsEosMain.f95`.

The interface between these codes is done in the main body of each of them. For instance `nsEosMain.f95` call the routines from `metaEosMain.f95` to generate the core EOS, and `tovMain.f95` calls the routines from `metaEosMain.f95` and `nsEosMain.f95`. They can therefore run independently from each other.

1 Meta-model for the nuclear equation of state

This part describes the code which generates the metaEOS, taking as input the values of the empirical parameters.

Compile: `$ bash compile-metaEos.sh`

Run: `$ ./metaEos.e {args}`

The arguments `{args}` can be:

- void: in this case, the code reads the default values defined in the file `metaEos.in`.

- the list of 13 values of the empirical parameters written as integers. See for instance `metaEos.in`.

It calls the main code `metaEosMain.f95` and the module `metaEosT0.f95`.

1.1 `metaEosMain.f95`

The code metaEOSMain.f95 is the main part of the code. It calls 3 subroutines:

- `call metaEos_read_inputs(coef)`,
- `call metaEos_compute_T0_SM_NM(coef)`,
- `call metaEos_compute_T0_beta(coef)`.

These subroutines are defined in the module `metaEosT0.f95`.

1.2 `metaEosT0Type.f95`

Define here the global parameters and the typed variables.

Note that you can choose the verbose mode (a lots of information on the standard output) or the non-verbose mode (information reduced to its minumum).

The global parameters are:

- `metaEos_neparam=15`: number of parameters of the model (=15). There are 13 empirical parameters plus `ikin` and `imuon`. The 13 empirical parameters are: E_{sat}, n_{sat}, K_{sat}, Q_{sat}, Z_{sat}, E_{sym}, L_{sym}, K_{sym}, Q_{sym}, Z_{sym}, m^*_{sat}, Δm^*_{sat}, b; and the two parameters control: `ikin=0(1)` empirical model without (with) kinetic energy, `imuon=0(1)` EOS without (with) muons.
- `metaEos_lverb=.true./.false.`: switch from verbose to non-verbose mode.

The typed variables are:

- `metaEos_Empirical`: `param(0:1,0:4)`, `nsat`, `ms`, `dm`, `xb`, `ikin`, `imuon`.
- `metaEos_Coef`: `VCOEF(0:1,0:4)`, `nsat`, `ms`, `dm`, `xb`, `ikin`, `imuon`, ...

- `metaEos_Densities` :
- `metaEos_Baryons` :
- `metaEos_Leptons` :
- `metaEos_Photons` :
- `metaEos_All` :

1.3 `metaEosT0.f95`

The important subroutines are described here:

- `metaEos_read_inputs(coef)` : Reads the input in different ways, depending on the arguments `{args}` of the code. The arguments are integer numbers defined between 0 and ± 9999, according to the following conversion:
  E_{sat}: `emp%param(0,0) =-0.01 * eparam(1)` . Ex: 1600 (-16.00 MeV)
  n_{sat}: `emp%nsat = 0.0001 * eparam(2)` . Ex: 1600 (0.1600 fm^{-3})
  K_{sat}: `emp%param(0,2) = 0.1 * eparam(3)` . Ex: 2350 (235.0 MeV)
  Q_{sat}: `emp%param(0,3) = 0.1 * eparam(4)` . Ex: 5000 (500.0 MeV)
  Z_{sat}: `emp%param(0,4) = 1.0 * eparam(5)` . Ex: -5000 (-5000 MeV)
  E_{sym}: `emp%param(1,0) = 0.01 * eparam(6)` . Ex: 3000 (30.00 MeV)
  L_{sym} `emp%param(1,1) = 0.01 * eparam(7)` . Ex: 4000 (40.00 MeV)
  K_{sym} `emp%param(1,2) = 0.1 * eparam(8)` . Ex: -1000 (-100.0 MeV)
  Q_{sym} `emp%param(1,3) = 1.0 * eparam(9)` . Ex: 1000 (1000 MeV)
  Z_{sym} `emp%param(1,4) = 1.0 * eparam(10)` . Ex: 7000 (7000 MeV)
  m^*_{sat} `emp%ms = 0.0001 * eparam(11)` . Ex: 7500 (0.7500)
  Δm^*_{sat} `emp%dm = 0.0001 * eparam(12)` . Ex: 1000 (0.1000)
  b: `emp%xb = 0.01 * eparam(13)` . Ex: 690 (6.90)
  In addition, there are two addition parameters `ikin=0/1` and `imuon=0/1` . Once the inputs are read: `call metaeos_setup_coeff(emp,coef)` to convert the empirical parameters in `emp` into the model coefficients in `coef` .

- `call metaeos_compute_T0_SM_NM(coef)` : compute the EOS in symmetric matter (SM) and neutron matter (NM). The lofical variable `iopt_beta=.false./.true.` optimizes (or not) the code for beta-equilibrium search.

  - `call metaeos_T0_densities(den,xd,xmuon,coef,eosDen)`: sets the densities of the particles for a given choice for `xd` and `xmuon=0`.
  - `call metaeos_T0_baryons(coef,eosDen,iopt_beta,eosb0)`: calculates the baryon contribution to the EOS. Note that the meta-model is defined here. For more details go to Refs. [1, 2].

  Print results on standard output.

- `call metaeos_compute_T0_beta(coef)`: compute the EOS at beta equilibrium, considering neutrons, protons, electrons and muons. For a faster code, muons can be neglected by setting `coef%imuon=0` in the input file.
  - if `coef%imuon=0`: `call metaeos_T0_beta_npe(den,xd,xmuon,coef,eosDen)`: calculates the EOS for n, p, and e matter (without μ).
  - if `coef%imuon=1`: `call metaeos_T0_beta_npemuon(den,xd,xmuon,coef,eosDen)`: calculates the EOS for n, p, e, and μ.
  - `call metaeos_T0_baryons(coef,eosDen,iopt_beta,eosb0)`: calculates the baryon contribution to the EOS. Note that the meta-model is defined here. For more details go to Refs. [1, 2].
  - `call metaeos_T0_leptons(eosDen,iopt_beta,eosl0)`: calculates the lepton contribution to the EOS (to be checked).
  - `call metaeos_T0_photons(eosDen,iopt_beta,eosp0)`: calculates the photon contribution to the EOS (to be done).
  - `call metaeos_T0_all(coef,eosDen,iopt_beta,eosb0,eosl0,eosp0,eosa0)`: sums up the contribution of all particles (to be checked).

2 Neutron star equation of state

This part describes the code which generates the neutron star EOS – connect the core EOS with a crust EOS – taking as input the values of the empirical parameters and the parameters for the crust EOS.

Compile: `$ bash compile-nsEos.sh`

Run: `$ ./nsEos.e {args}`

The arguments `{args}` can be:

- void or the list of 15 values of the empirical parameters written as integers: in this case, the parameters for the crust EOS are explicitly written in the subroutine. To modify them, edit the subroutine `nsEos_read_inputs` defined in `nsEosMod.f95`
- one argument defining the name of the input file, such as for instance `nsEos.in`, where the parameters of the crust EOS are defined.

It calls the main code `nsEosMain.f95` and the module `nsEosMod.f95`.

2.1 `nsEosMain.f95`

The code nsEOSMain.f95 is the main part of the code. It calls 4 subroutines:

- `call nsEos_read_inputs(nsEos_inp)`: reads the input parameters for the nsEos code,
- if `nsEOS_inp%core="meos"`: `call metaEos_read_inputs(coef_meta)`: reads the input parameters for the meta-model,
- if `nsEOS_inp%core="poly"`: `call polyEos_read_inputs(coef_poly)`: reads the input parameters for the piecewise polytrope,
- `call nsEos_compute_eos(coef_meta,coef_poly,nsEos_inp,nseos)`: join together the crust and the core EOS,
- `call nsEos_write_eos(nseos)`: write results in folder `nsEos-res/`.
- `call nsEos_write_table_eos(nseos,coef_meta)`: write table in folder `nsEos-res/` for NS merger models.

2.2 `nsEosType.f95`

Define here the global parameters and the typed variables.

Note that you can choose the verbose mode (a lots of information on the standard output) or the non-verbose mode (information reduced to its minumum).

The global parameters are:

- `nsEos_ncrust=200`: number of density point for the crust EOS.
- `nsEos_ncore=200`: number of density point for the core EOS.

- `nsEos_neos=nsEos_ncrust+nsEos_ncore` : number of density point for the crust+core EOS.
- `nsEos_lverb=.true./.false.` : switch from verbose to non-verbose mode.
- `nsEos_lfast=.true./.false.` : switch from fast to slower mode (fast to be completed).

The typed variables are:

- `nsEos_inputs` : input parameters.
- `nsEos_crust` : crust quantities.
- `nsEos_core` : core quantities.
- `nsEos_eos` : crust+core quantities.

Note that the `crust` , `core` and `eos` typed variables have each the following array: `EosLog(i,iden)` where

- $i = 1$: $\log(n_b)$, $10^{eos\%EosLog(1,iden)}$ is the baryon density in fm^{-3}.
- $i = 2$: $\log(\rho)$, $10^{eos\%EosLog(2,iden)}$ is the energy density in g.cm^{-3}.
- $i = 3$: $\log(P)$, $10^{eos\%EosLog(3,iden)}$ is the pressure in dyn.cm^{-2}.
- $i = 4$: $(v_c/c)^2$, $eos\%EosLog(4, iden)$ is the sound velocity.

2.3 `nsEosMod.f95`

The important subroutines are described here.

- `call nsEos_read_inputs(nsEos_inp)` : Reads the input in different ways, depending on the arguments `{args}` of the code. These parameters are explained hereafter.

  - `nsEos_inp%crust` : name of the file (stored in `nstar-EosCrut/`) where the crust EOS is given (table). it can be `"sly.dat"` or `"fps.dat"` , or ...

  - `nsEos_inp%core` : kind of EOS for the core. It can be the metaEOS (`"meos"`) or a polytropic form (not yet implemented).

  - `nsEos_inp%dobeta` : Type of equations to solve the β-equilibrium. By default keep it to be 3 (solves the chemical potential equation). The value 1 solves the approximate equation based on the symmetry energy.

  - `nsEos_inp%temp` : set it to 0.d0. This code does not works yet at finite temperature.

  - `nsEos_inp%den_core_min` : minimal density for the core EOS (in units of n_{sat}).

  - `nsEos_inp%den_core_max` : maximal density for the core EOS (in units of n_{sat}).

  - `nsEos_inp%den_core_step` : step density for the core EOS (in units of n_{sat}).

- `call metaEos_read_inputs(coef_meta)` : see Sec. 1.

- `call nsEos_compute_eos(coef_meta,coef_poly,nsEos_inp,nseos)` : This subroutine computes the metaEOS for the core equation of state, see Sec. 1, and matches it to the crust EOS. Notice that by default, the core EOS without muon is calculated here. to switch to an EOS with muons, comment line 524 and uncomment line 520. This subroutine calls the following subroutines:

  - `call nsEos_compute_crust(crust)` : store the crust EOS into `crust%EosLog(i,iden)` , where:

    * $i = 1$ stores $\log_{10} n$ in (fm^{-3}),
    * $i = 2$ stores $\log_{10} \rho$ in (g.cm^{-3}),
    * $i = 3$ stores $\log_{10} P$ in (dyn.cm^{-2}),
    * $i = 4$ stores $(v_s/c)^2$.

  - `call nsEos_compute_core_metaEos(core)` : store the core EOS from the metamodeling into `core%EosLog(i,iden)` . At the exit of the loop density, tells if stability/causality is fulfilled/violated. The max density (stored in `core%den_max`) is defined as the first density for which either one or the other of these conditions are violated.
  - `call nsEos_compute_core_polyEos(core)` : store the core EOS from the piecewise polytropes into `core%EosLog(i,iden)` (to be implemented).
  - `call nsEos_match_eos(eos)` : combine crust and core EOS into `eos%EosLog(i,iden)` .

- `call nsEos_write_eos(nseos)` : write the neutron star EOS into different files

  - `nsEos-res/eos.out` : the same data as the one used by the code TOV,
  - `nsEos-res/eos-lin.out` : a linear interpolation of the EOS – mostly for the high density core EOS,
  - `nsEos-res/eos-log.out` : a log interpolation of the EOS.

  These files store the results by columns in the following order: n in (fm^{-3}), ρ in (g.cm^{-3}), P in (dyn.cm^{-2}), and $(v_s/c)^2$.

- `call nsEos_write_table_eos(nseos,coef_meta)` : write the neutron star EOS into a table for the NS merger simulation:

  - `nsEos-res/table.out` : a log interpolation of the EOS as defined for the NS merger simulation

  This file stores the results by columns in the following order: density index i, n in (fm^{-3}), ρ in (g.cm^{-3}), P in (dyn.cm^{-2}), and $(v_s/c)^2$. The head of the file contains the meta-model parameters and some crust information.

3 Solver for the TOV equations

This part describes the code which solves the TOV equations, based on the neutron star equation of state previously generated.

Compile: `$ bash compile-tov.sh`

Run: `$ ./tov.e {args}`

The arguments `{args}` can be:

- void or the list of 15 values of the empirical parameters written as integers: in this case, the parameters for TOV solver are explicitly written in the subroutine. To modify them, edit the subroutine `tov_read_inputs` defined in `tovMod.f95`
- 2 arguments: these arguments define the name of the input files containing the parameters for the nsEos code and the TOV solver, for instance `nsEos.in tov.in`.

It calls the main code `tovMain.f95` and the module `tovMod.f95`.

3.1 `tovMain.f95`

The code tovMain.f95 is the main part of the code. It calls 4 subroutines:

- Build the EOS (same as in nsEosMain.f95):
  - `call nsEos_read_inputs(nsEos_inp)`: reads the input parameters for the nsEos code,
  - if `nsEOS_inp%core="meos"`: `call metaEos_read_inputs(coef_meta)`: reads the input parameters for the meta-model,
  - if `nsEOS_inp%core="poly"`: `call polyEos_read_inputs(coef_poly)`: reads the input parameters for the piecewise polytrope,
  - `call nsEos_compute_eos(coef_meta,coef_poly,nsEos_inp,nseos)`: join together the crust and the core EOS,
  - `call nsEos_write_eos(nseos)`: write results in folder `nsEos-res/`.
- Solves the TOV equations:
  - `tov_read_inputs(tov_inp)`: reads the input parameters for the tov code,
  - `tov_write_eos(tov_eos)`: write EOS in folder `tov-res/`,
  - stop here if `tov%do_tov="no"`, otherwise proceed.
  - if `tov%nbc≠0`: `tov_solve(tov_inp,tov_eos)`. Solves the TOV eqs for the central density defined in variable `tov%nbc`.

– if `tov%nbc=0` : `tov_solve_loopden(tov_inp,tov_eos)` . Solves the TOV eqs for a set of densities starting from 0.1 fm^{-3} up to the maximal density, with step `CST_nsat/30.0` .

3.2 `tovType.f95`

Define here the global parameters and the typed variables.

Note that you can choose the verbose mode (a lots of information on the standard output) or the non-verbose mode (information reduced to its minumum).

The global parameters are:

- `tov_nrad=10000` : number of points for the radial integration.
- `tov_neos=400` : number of density-points for the EOS.
- `tov_lverb=.true./.false.` : switch from verbose to non-verbose mode.
- `tov_lfast=.true./.false.` : switch from fast to slower mode (fast to be completed).

The typed variables are:

- `TY_TOV_inputs` : input parameters.
- `TY_TOV_eos` : crust+core quantities.
- `TY_TOV_outputs` : output results.
- `TY_TOV_outputs_rot` : output results for the slow rotation case.
- `TY_TOV_outputs_tidal` : output results for the tidal deformability.

3.3 `tovMod.f95`

The important subroutines are described here.

`call tov_read_inputs(tov_inp)` : Reads the input in different ways, depending on the arguments `{args}` of the code. These parameters are explained hereafter.
`tov_inp%do_tov` : ”yes” or ”no”. Decide to run or not the TOV solver,

after the construction of the EOS (ex: ”yes”).
`tov_inp%dradmax` , `tov_inp%dradmin` : Adaptative mesh in the radial integration of the TOV equations. maximal and minimal step in unit of the Schwarzschild radius for the sun (ex: 1.d-1, 3.d-4).
`tov_inp%brad` : maximal value in the radial integration, in cm (ex: 30.d5, 30 km).
`tov_inp%nbc` : central baryon density for the TOV solver, in fm^{-3}. If set to 0, then loop over different values for the central density (ex: 0).
`tov_inp%nbexit` : run TOV solver only if the metaEOS is stable and causal up to this density, defined in units of n_{sat} (ex: 2.5).
`tov_inp%pressMin` : minimal pressure below which the solver is stopped and where the radius is defined, in dyn/cm2 (ex:1.d10).
`tov_inp%aom0` : Ω_0 for the calculation of the moment of inertia (ex: 1.d-8).

References

[1] J. Margueron, R. Casali, and F. Gulminelli, Phys. Rev. C 96, 065805 (2018).

[2] J. Margueron, R. Casali, and F. Gulminelli, Phys. Rev. C 96, 065806 (2018).

[3] J. Piekarewicz and M. Centelles, Phys. Rev. C 79, 054311 (2009).