The TDHF Code Sky3D
Online Documentation

J. A. Maruhn*

Institut für Theoretische Physik, Goethe-Universität, Max-von-Laue-Str. 1, 60438 Frankfurt am Main, Germany

P.-G. Reinhard

Institut für Theoretische Physik II, Universität Erlangen-Nürnberg, Staudtstrasse 7, 91058 Erlangen, Germany

P. D. Stevenson

Department of Physics, University of Surrey, Guildford, Surrey, GU2 7XH, United Kingdom

A. S. Umar

Department of Physics and Astronomy, Vanderbilt University, Nashville, Tennessee 37235, USA

Abstract

The nuclear mean-field model based on Skyrme forces or related density functionals has found widespread application to the description of nuclear ground states, collective vibrational excitations, and heavy-ion collisions. The code Sky3D solves the static or dynamic equations on a three-dimensional Cartesian mesh with isolated or periodic boundary conditions and no further symmetry assumptions. Pairing can be included in the BCS approximation for the static case. The code is implemented with a view to allow easy modifications for including additional physics or special analysis of the results.

Keywords: Hartree-Fock; BCS; Density-functional theory; Skyrme energy functional; Giant Resonances; Heavy-Ion collisions.

PROGRAM SUMMARY

Title: The TDHF Code Sky3D
Authors: J. A. Maruhn, P.-G. Reinhard, P. D. Stevenson, and A. S. Umar.
Program Title: Sky3D
Journal Reference:
Catalogue identifier:

*Corresponding author. *Phone: +49-69-79847873*
Email addresses: maruhn@th.physik.uni-frankfurt.de (J. A. Maruhn), Paul-Gerhard.Reinhard@physik.uni-erlangen.de (P.-G. Reinhard), p.stevenson@surrey.ac.uk (P. D. Stevenson), sait.a.umar@Vanderbilt.Edu (A. S. Umar)

Licensing provisions: none
Programming language: Fortran 90. The `OpenMP` version requires a relatively recent compiler; it was found to work using `gfortran 4.6.2` or later and the Intel compiler version 12 or later.
Computer: All computers with a Fortran compiler supporting at least Fortran 90.
Operating system: All operating systems with such a compiler. Some of the Makefiles and scripts depend on a Unix-like system and need modification under Windows.
RAM: 1 GB
Number of processors used: no built-in limit, runs under both OpenMP and MPI
Keywords: Nuclear theory, Mean-field models, Nuclear reactions
Classification: 17.16 Theoretical Methods - General, 17.22 Hartree-Fock Calculations, 17.23 Fission and Fusion Processes
External routines/libraries: LAPACK, FFTW3

Nature of problem: The time-dependent Hartree-Fock equations can be used to simulate nuclear vibrations and collisions between nuclei for low energies. This code implements the equations based on a Skyrme energy functional and also allows the determination of the ground-state structure of nuclei through the static version of the equations. For the case of vibrations the principal aim is to calculate the excitation spectra by Fourier-analyzing the time dependence of suitable observables. In collisions, the formation of a neck between nuclei, the dissipation of energy from collective motion, processes like charge transfer and the approach to fusion are of principal interest.

Solution method: The nucleonic wave function spinors are represented on a three-dimensional Cartesian mesh with no further symmetry restrictions. The boundary conditions are always periodic for the wave functions, while the Coulomb potential can also be calculated for an isolated charge distribution. All spatial derivatives are evaluated using the finite Fourier transform method. The code solves the static Hartree-Fock equations with a damped gradient iteration method and the time-dependent Hartree-Fock equations with an expansion of the time-development operator. Any number of initial nuclei can be placed into the mesh in with arbitrary positions and initial velocities.

Restrictions: The reliability of the mean-field approximation is limited by the absence of hard nucleon-nucleon collisions. This limits the scope of applications to collision energies about a few MeV per nucleon above the Coulomb barrier and to relatively short interaction times. Similarly, some of the missing time-odd terms in the implementation of the Skyrme interaction may restrict the applications to even-even nuclei.

Unusual features:
The possibility of periodic boundary conditions and the highly flexible initialization make the code also suitable for astrophysical nuclear-matter applications.

Running time: The running time depends strongly on the size of the grid, the number of nucleons, and the duration of the collision. For a single-processor PC-type computer it can vary between a few minutes and weeks.

Contents

1. Introduction

The vast majority of microscopic models of many-body systems rely on a description in terms of the single-particle (s.p.) wave functions. Among them, self-consistent mean-field models (SCMF) automatically generate the optimal one-body potentials corresponding to the s.p. wave functions. A rigorous SCMF is the Hartree-Fock theory (HF) where the s.p. wave functions are determined variationally for a given two-body interaction [1, 2]. A more practical approach is provided by the Density Functional Theory (DFT), which incorporates the involved many-body effects into effective interactions, or effective energy-density functionals. This is a very efficient and successful scheme, widely used in electronic systems [3]. Straightforward HF is unsuitable for nuclei because the free-space two-nucleon force contains a strong short-range repulsion requiring renormalization in the nuclear medium. For this reason, nuclear SCMFs necessarily employ effective interactions or functionals although they often carry the label HF as, e.g, in the Skyrme Hartree-Fock (SHF) method. There are relativistic as well as non-relativistic approaches. For an extensive review see [4].

The description of dynamical processes is even more demanding than the modeling of structure. SCMFs are also the first method of choice in this domain. The natural extension of HF is time-dependent HF (TDHF) which was proposed as early as 1930 in [5]. Earlier applications were restricted to the linearized regime covering small amplitude motion, see, e. g., [6]. Large scale TDHF calculations became possible in the last few decades with the increasing computing capacities. Again, as in the static case, true TDHF calculations make sense only for electronic systems and even there they are very rare. The overwhelming majority of dynamical SCMF calculations employ, in fact, time-dependent DFT (TDDFT). In electronic systems, this amounts to the time-dependent local density approximation (TDLDA) [3], which is widely used in atoms, molecules, and solids; for examples in nanoparticles see, e. g., [7]. Dynamical SCMFs in nuclei also stay at the level of TDLDA even if they are often named TDHF which happens particularly for dynamical calculations using the Skyrme energy functional. Nuclear TDHF started about forty years ago [8] and has developed since then into a powerful and versatile tool for simulating a great variety of dynamical scenarios. Earlier applications were based mainly on non-relativistic TDHF using the effective Skyrme energy functional [9, 10]. Due to higher numerical demands, relativistic calculations appeared somewhat later [11], but have developed meanwhile equally well to a widely used tool [12, 13].

In this paper, we present a code for TDHF calculations on the basis of the non-relativistic Skyrme energy functional. The code uses a fully three dimensional (3D) representation of wave functions and fields on a Cartesian grid in coordinate space. There are no symmetry restrictions and the full Skyrme energy functional is used including the spin-orbit and most important time-odd terms. Such fully-fledged 3D calculations became possible only over the last decade with the steadily increasing computing capabilities. In fact, early TDHF studies all used restricted representations, axial symmetry and/or reflection symmetries. This limited the possible applications. TDHF experienced a revival during the last ten years when unrestricted 3D calculations became possible. There are several groups performing large scale TDHF studies for various scenarios of nuclear dynamics, see, e.g., [14–16]. Aside from these studies of nuclear collisions a principal application has been to collective vibrations, e.g., [17–20]. In the linear regime TDHF leads to fully self-consistent RPA, for which though, unlike TDHF, often additional approximations like the neglect of the Coulomb potential or the spin-orbit terms in the residual interaction are made. TDHF can also be used to investigate non-linearities of nuclear vibrations.

For a recent review of vibrational and collisional applications see [21]. Such calculations have clearly outgrown the developmental stage. It is an appropriate time to give a broader public access to a 3D TDHF code. This is the goal of this paper. Skyrme HF covers such a broad range of physical phenomena and is relatively involved that efficient computational treatment of 3D simulations requires elaborate numerical methods. We shall make an effort to explain the many necessary ingredients in a comprehensive, and yet compact, manner.

Most recent Skyrme density functionals contain terms such as fractional powers of the density that cannot be related to a two- or three-body interaction. In that sense, the present code solves the TDDFT rather than TDHF equations. Nevertheless, we prefer to keep the name TDHF since it is associated historically with this large field of nuclear reaction theory.

Although the code may be run as it is and many innovative applications are possible, we also intend it to be used for exploring new ideas that need a basic HF and TDHF algorithm, which is implemented in a transparent modern style of programming and extensively documented (see the accompanying online documentation), allowing for relatively easy modification. A guide to some possible extensions is given in Sect. 11.

Some recent developments in TDHF which require more extensive modification but might be implemented on the basis of this code include:

- The extraction of nucleus-nucleus potentials with and without dynamical effects. Approaches include density-constrained TDHF (DC-TDHF) method [22], the density-dependent TDHF (DD-TDHF) approach [23], and the frozen HF method [24].
- The extraction of (multi-)nucleon transfer probabilities using particle number projection techniques [25].
- The use of novel spatial distributions to excite low-lying dipole states [26].
- The incorporation of fluctuations of one-body observables using the Balian-Vénéroni variational principle [27–29].
- The inclusion of dynamical pairing correlations at the TDHF+BCS or TDHFB levels [30–32]
- The inclusion of collision terms with the Extended TDHF or the time-dependent density matrix approaches [33, 34].
- The generalization of the static HF code to odd particles by the inclusion of the time-odd terms in the Skyrme interaction [35–37].

2. General purpose and structure

2.1. Intended applications

The code Sky3D solves the static Hartree-Fock as well as the time-dependent Hartree-Fock (TDHF) equations for interactions of Skyrme-force type in a general three-dimensional geometry. No symmetries of any kind are assumed, so that the code can be used for a wide variety of applications in nuclear structure, collective excitations, and nuclear reactions; of course within the limitations of mean-field theory.

2.2. Specific model implemented

The code in the presented version contains a useful selection of terms in the Skyrme force but by no means all terms that have been included in some recent works. It should still be useful, because (1) for many interesting applications the interest is semi-quantitative so that a Skyrme force fitted with the latest models is not necessary — usually a selection of forces is desired to look at the variability of results, but not a high-accuracy fit of data; (2) the code is written in such a way that additional terms can be added easily. The coding corresponds one-to-one to the analytic formulas in most places except where efficiency demands reordering the calculations.

2.2.1. The single-particle basis

In a mean field theory one seeks to describe the many-body system exclusively in terms of a set of single-particle wave functions ψ_α with fractional occupation amplitudes v_α, i.e.

$$\{\psi_\alpha, v_\alpha, \alpha = 1, ..., \Omega\} \tag{1a}$$

where Ω denotes the size of the active s.p. space. The occupation amplitude can take values continuously in the interval $[0, 1]$. The complementary non-occupation amplitude is $u_\alpha = \sqrt{1 - v_\alpha^2}$. A formal definition of the BCS mean-field state reads

$$|\Phi\rangle = \prod_{\alpha>0} \left(u_\alpha + v_\alpha \hat{a}_\alpha^+ \hat{a}_{\bar{\alpha}}^+\right)|0\rangle \tag{1b}$$

where $|0\rangle$ is the vacuum state, $\hat{a}_\alpha^+$ the generator of a Fermion in state ψ_α, and $\bar{\alpha}$ the time reverse partner to state α. We will use variation of the BCS amplitudes v_α only in the static part of even-even nuclei where the time reverse partner is unambiguously defined. In fact, the pairing calculation assumes that the paired states have exactly the same spatial density distribution.

The time-dependent calculation technically can be run with pairing included. This is done by keeping the pairing occupation probabilities fixed during the time evolution; the pairing between states can simply be taken over from the static calculation, although time-reversal invariance is lost for boosted nuclei. The conservation of total energy *excluding the pairing energy* is not impaired for the case of constant occupation, while the pairing energy is not even computed.

It should be noted, however, that once the wave functions change dynamically, this approach is not correct as the occupation probabilities will also change with time. In this case the TDHF-Bogolyubov equations should be used. In addition, the pairing energies are not computed at all. Using this code as is with pairing included in the time-dependent case might be useful for schematic or exploratory studies but extreme caution is advised when interpreting such results.

2.2.2. Local densities and currents

The Skyrme-energy-density functional is defined in terms of only a few local densities and currents. These are the *time-even* fields

$$\begin{aligned}
\rho_q(\vec{r}) &= \sum_{\alpha\in q}\sum_{s} v_\alpha^2 |\psi_\alpha(\vec{r},s)|^2 && \text{density} \\
\vec{J}_q(\vec{r}) &= -\mathrm{i}\sum_{\alpha\in q}\sum_{ss'} v_\alpha^2 \psi_\alpha^*(\vec{r},s)\nabla\times\vec{\sigma}_{ss'}\psi_\alpha(\vec{r},s') && \text{spin-orbit dens.} \\
\tau_q(\vec{r}) &= \sum_{\alpha\in q}\sum_{s} v_\alpha^2 |\nabla\psi_\alpha(\vec{r},s)|^2 && \text{kinetic density,}
\end{aligned} \tag{2}$$

the *time-odd* fields

$$\begin{aligned}
\vec{s}_q(\vec{r}) &= \sum_{\alpha\in q}\sum_{ss'} v_\alpha^2 \psi_\alpha^*(\vec{r},s)\vec{\sigma}_{ss'}\psi_\alpha(\vec{r},s') && \text{spin density} \\
\vec{j}_q(\vec{r}) &= \Im m\left\{\sum_{\alpha\in q}\sum_{s} v_\alpha^2 \psi_\alpha^*(\vec{r},s)\nabla\psi_\alpha(\vec{r},s)\right\} && \text{current density,}
\end{aligned} \tag{3}$$

and a field with undefined time parity:

$$\xi_q(\vec{r}) = \sum_{\alpha\in q}\sum_{s} u_\alpha v_\alpha \psi_{\overline{\alpha}}(\vec{r},s)\psi_\alpha(\vec{r},s) \qquad \text{pairing density} \tag{4}$$

where q labels the nucleon species with $q = p$ for protons and $q = n$ for neutrons. A local density/current without q index stands for the total quantity, e.g. $\rho = \rho_p + \rho_n$ is the total density, and similarly for the other densities/currents. The variable s indicates the two spinor components of the wave functions.

2.2.3. The energy-density functional

The mean-field equations solved in the code are based on the widely used Skyrme energy functional. For recent reviews see [4, 38]. The functional at the level at which it is used here can be written as

$$E_{\text{tot}} = T + (E_0 + E_1 + E_2 + E_3 + E_{\text{ls}}) + E_{\text{Coulomb}} + E_{\text{pair}} + E_{\text{corr}}, \tag{5a}$$

where the parentheses were used to group the terms arising from the Skyrme force. The various terms read in detail (all densities and currents defined in Section 2.2.2 are understood to depend on $\vec{r}$)

- T: the total kinetic energy calculated as

$$T = \sum_q \frac{\hbar^2}{2m_q}\int \mathrm{d}^3r\,\tau_q \tag{5b}$$

  with τ_q the isospin-specific kinetic density of Eq. (2).

- E_0: The b_0 and b_0'-dependent part is

$$E_0 = \int \mathrm{d}^3r \left(\frac{b_0}{2}\rho^2 - \frac{b_0'}{2}\sum_q \rho_q^2\right). \tag{5c}$$

- E_1: kinetic terms containing the coefficients b_1 and b_1':

$$E_1 = \int \mathrm{d}^3r \left(b_1[\rho\tau - \vec{j}^2] - b_1' \sum_q [\rho_q\tau_q - \vec{j}_q{}^2] \right) . \tag{5d}$$

- E_2: terms containing the coefficients b_2 and b_2'. They involve the Laplacians of the densities.

$$E_2 = \int \mathrm{d}^3r \left(-\frac{b_2}{2}\rho\Delta\rho + \frac{b_2'}{2} \sum_q \rho_q\Delta\rho_q \right) . \tag{5e}$$

- E_3: The many-body contribution is given by

$$E_3 = \int \mathrm{d}^3r \left(\frac{b_3}{3}\rho^{\alpha+2} - \frac{b_3'}{3}\rho^{\alpha} \sum_q \rho_q^2 \right) . \tag{5f}$$

- E_{ls}: the spin-orbit energy

$$\begin{aligned} E_{\mathrm{ls}} = \int \mathrm{d}^3r \Big(& -b_4[\rho\nabla\cdot\vec{J} + \vec{s}\cdot(\nabla\times\vec{j})] \\ & - b_4' \sum_q [\rho_q\nabla\cdot\vec{J}_q + \vec{s}_q\cdot(\nabla\times\vec{j}_q)] \Big) \end{aligned} \tag{5g}$$

- E_{C}: the Coulomb energy. It consists of the standard expression for a charge distribution in its own field (Hartree term) plus the exchange term in the Slater approximation [39]. The formula is

$$E_{\mathrm{C}} = \frac{e^2}{2} \int \mathrm{d}^3r \mathrm{d}^3r' \frac{\rho_p(\vec{r})\rho_p(\vec{r'})}{|\vec{r} - \vec{r'}|} - \int \mathrm{d}^3r \frac{3e^2}{4} \left(\frac{3}{\pi}\right)^{\frac{1}{3}} \rho_p^{4/3} \tag{5h}$$

where e is the elementary charge with $e^2 = 1.43989$ MeV·fm.

- E_{pair}: the pairing energy. It consists of a contact pairing interaction involving the pairing densities ξ_q augmented by an optional density dependence. The formula is

$$E_{\mathrm{pair}} = \frac{1}{4} \sum_{q\in\{p,n\}} V_{\mathrm{pair,q}} \int \mathrm{d}^3r |\xi_q|^2 \left[1 - \frac{\rho}{\rho_{0,\mathrm{pair}}} \right] . \tag{5i}$$

It contains a continuous switch, the parameter $\rho_{0,\mathrm{pair}}$. A pure δ-interaction (DI), also called volume pairing, is recovered for $\rho_{0,\mathrm{pair}} \longrightarrow \infty$. The general case is the density dependent δ-interaction (DDDI). A typical value near matter equilibrium density $\rho_{0,\mathrm{pair}} = 0.16$ fm^{-3} concentrates pairing to the surface. The most flexible choice is to consider $\rho_{0,\mathrm{pair}}$ as an additional free parameter. Actual adjustments with this option deliver a form of the pairing functional which stays in between the extremes of volume and surface pairing [40]. The implementation in the code is discussed in Section 5.15.

The term $E_{\rm corr}$ stands for all additional corrections from correlations beyond the mean field that might be added. Most calculations include at least the center-of-mass correction $E_{\rm cm}$. For deformed nuclei this should be augmented by a rotational correction and for soft nuclei by correlations from all low-energy quadrupole motions [41]. So far, these correlations are usually added a posteriori after static calculations. This procedure is associated with setting the switch `zpe=1` which is the standard option adopted here. A fully variational treatment and a dynamical propagation of the c.m. correction is extremely involved and usually not considered. The other strategy is to modify the nucleon mass by $m \longrightarrow m - m/A$ and to include this simplified correction in the variational treatment thus avoiding the a posteriori correction. This way is chosen in a couple of traditional parametrizations, e.g., in SkM* [42]. We keep this option in case of static calculations for consistency and associate it with `zpe=0`.

Using a center-of-mass correction is desirable for static calculations, it is disputable for vibrational excitations, and runs fully into inconsistencies in collisions and fragmentation as it employs only the total mass number A and cannot account for the masses of the fragments. Therefore all dynamical runs are done with

$$E_{\rm cm} = 0 \quad . \tag{6}$$

An inconsistency may occur if `zpe=0` was used in the static calculation providing the input for the dynamical run because the setting $m \longrightarrow m - m/A$ is not used in dynamics. In order to safely suppress the c.m. correction in statics and dynamics, we introduce an additional switch `turnoff_zpe` in the input, which turns off the zero-point energy correction irrespective of what value is given to `zpe` in the force definition. This allows the use of forces with `zpe=0` also for collisions. It should be pointed out also that there are newer Skyrme parametrizations like Sly4d [43, 44] and UNEDF [45] that are fitted without any center-of-mass correction and are thus specially intended for collision calculations.

For some further considerations on the c.m. correction in TDHF see [**?**].

The functional in the above form contains the minimal number of terms which are needed to guarantee Galilean invariance [35, 38] and so to allow performance of TDHF calculations which respect all basic conservation laws. We ignore the tensor spin-orbit terms and spin-spin couplings [4, 38, 46]. These may be important for magnetic excitations [38] and odd nuclei [47] which are, however, not the prime focus of TDHF studies.

2.2.4. Force coefficients

The above formulation in terms of the Skyrme energy functional introduces the force parameters b_0, b'_0, ... b'_4 naturally as the factors in front of each contribution in the terms (5c-5g). Traditionally, the functional is deduced from a Skyrme force which is a density-dependent, zero-range interaction [48]. The t and x coefficients in this Skyrme-force definition are related to the b coefficients in the functional definition as

$$\begin{aligned}
b_0 &= t_0 \left(1 + \tfrac{1}{2}x_0\right) \\
b'_0 &= t_0 \left(\tfrac{1}{2} + x_0\right) \\
b_1 &= \tfrac{1}{4}\left[t_1 \left(1 + \tfrac{1}{2}x_1\right) + t_2 \left(1 + \tfrac{1}{2}x_2\right)\right] \\
b'_1 &= \tfrac{1}{4}\left[t_1 \left(\tfrac{1}{2} + x_1\right) - t_2 \left(\tfrac{1}{2} + x_2\right)\right] \\
b_2 &= \tfrac{1}{8}\left[3t_1 \left(1 + \tfrac{1}{2}x_1\right) - t_2 \left(1 + \tfrac{1}{2}x_2\right)\right] \\
b'_2 &= \tfrac{1}{8}\left[3t_1 \left(\tfrac{1}{2} + x_1\right) + t_2 \left(\tfrac{1}{2} + x_2\right)\right] \\
b_3 &= \tfrac{1}{4}t_3 \left(1 + \tfrac{1}{2}x_3\right) \\
b'_3 &= \tfrac{1}{4}t_3 \left(\tfrac{1}{2} + x_3\right) \\
b_4 &= \tfrac{1}{2}t_4
\end{aligned} \tag{7}$$

The coefficient b'_4 is usually fixed as $b'_4 = \frac{1}{2}t_4$ for most traditional Skyrme forces. More recent variants of Skyrme forces (SkI3 etc.) handle it as a separate free parameter [49]. In addition to the b and b' parameters, there is the power coefficient in the (originally) three-body term, which is usually called α, but in the code is referred to as `power`. The input of the force to the code is done in terms of the t_i, x_i parameters, see Section 5.7.

2.2.5. The single-particle Hamiltonian

The mean-field Hamiltonian $\hat{h}$ is derived from the energy functional of Section 2.2.3 by variation $\partial_{\psi^*_\alpha} E = \hat{h}\psi_\alpha$. It reads in detail

$$\begin{aligned}
\hat{h}_q &= U_q(\vec{r}) - \nabla \cdot [B_q(\vec{r})\nabla] + \mathrm{i}\vec{W}_q \cdot (\vec{\sigma} \times \nabla) + \vec{S}_q \cdot \vec{\sigma} \\
&\quad - \frac{\mathrm{i}}{2}\left[(\nabla \cdot \vec{A}_q) + 2\vec{A}_q \cdot \nabla\right] ,
\end{aligned} \tag{8a}$$

with $q \in \{p, n\}$ as usual distinguishing the different Hamiltonians for protons and neutrons. For the protons the Coulomb potential is also added. The first term is the local part of the mean field, which acts on the wave functions like a local potential. It is defined as

$$\begin{aligned}
U_q &= b_0\rho - b'_0\rho_q + b_1\tau - b'_1\tau_q - b_2\Delta\rho + b'_2\Delta\rho_q \\
&\quad + b_3\tfrac{\alpha+2}{3}\,\rho^{\alpha+1} - b'_3\tfrac{2}{3}\,\rho^\alpha\rho_q - b'_3\tfrac{\alpha}{3}\,\rho^{\alpha-1}\sum_{q'}\rho^2_{q'} \\
&\quad - b_4\nabla \cdot \vec{J} - b'_4\nabla \cdot \vec{J}_q \,.
\end{aligned} \tag{8b}$$

Next comes the "effective mass", which replaces the standard $\frac{\hbar^2}{2m}$ factor by the isospin and space-dependent expression

$$B_q = \frac{\hbar^2}{2m_q} + b_1\rho - b'_1\rho_q. \tag{8c}$$

Note that the Skyrme force definitions contain the first term (nucleon mass) as a parameter which varies slightly from parametrization to parametrization and may be different for protons and neutrons. The spin-orbit potential is

$$\vec{W}_q = b_4\nabla\rho + b'_4\nabla\rho_q \quad . \tag{8d}$$

The above three terms involve the time-even densities. Dynamical effects come into play with the next terms which include the time-odd contributions from current and spin-density:

$$\vec{A}_q = -2b_1\vec{\jmath} + 2b_1'\vec{\jmath}_q - b_4\nabla\times\vec{s} - b_4'\nabla\times\vec{s}_q. \,, \tag{8e}$$

$$\vec{S}_q = -b_4\nabla\times\vec{\jmath} - b_4'\nabla\times\vec{\jmath}_q \,. \tag{8f}$$

2.3. Coupling to external fields

For the dynamic case, the system can also be coupled to an external excitation field, to study collective response such as in giant resonances. The present code only implements a very simple case, since it is expected that most serious applications will need modifications, which are quite easy to incorporate.

The external field is introduced as a time-dependent, local operator

$$\hat{h}_q \longrightarrow \hat{h}_q + U_{q,\mathrm{ext}}(\vec{r},t) \quad , \quad U_{q,\mathrm{ext}} = \eta\, f(t)\, F_q(\vec{r}) \,, \tag{9a}$$

where $f(t)$ carries the temporal profile of the excitation mechanism, $F_q(\vec{r})$ is some local operator, and η tunes the overall strength. The spatial distribution $F_q(\vec{r})$, is allowed to be different for the two isospins q. Typical examples are isoscalar and isovector multipole operators as, e.g., the isoscalar quadrupole $F_q(\vec{r}) = 2z^2 - x^2 - y^2$.

The prefactor η is a strength parameter which allows scanning different excitation strengths easily while keeping the temporal and spatial profiles the same. It should be noted that the absolute magnitude of the perturbing potential by itself usually has little direct meaning. What counts is the excitation energy caused by the perturbation and subsequently the magnitude of vibrations in the observables (such as the time-dependent quadrupole moment). An exception is, e. g., the simulation of the close approach of another nucleus that stays external to the computational grid, where the potential is uniquely defined.

One important point remains to be noted concerning the spatial profile $F_q(\vec{r})$. This can be illustrated by the quadrupole operator. Let us assume an instant where $A > 0$ and $f(t) > 0$. For then the operator $\propto 2z^2 - x^2 - y^2$ leads to a perturbing potential U_{ext} which is binding in z-direction but asymptotically unbound in the x- and y-directions. This can cause unphysical effects in case of large strengths and/or numerical boxes. For this reason it is useful to have a cut-off by a Woods-Saxon like function according to [50]

$$F_q(\vec{r}) \to \frac{F_q(\vec{r})}{1 + e^{(r-r_0)/\Delta r}} \,, \tag{9b}$$

where r_0 and Δr are parameters describing a transition region sufficiently outside the nucleus, but also sufficiently small to maintain binding.

Another problem associated with the external field is that in general it will not be periodic but instead have discontinuities on the boundary when crossing into the neighboring cell. If damping is sufficiently strong, the field may be practically zero on the boundary and thus become periodic. Another solution for this problem is to make the field explicitly periodic by replacing the coordinates with periodic substitutes. The exact formulation depends on the specific field used; for the above-mentioned quadrupole operator, which depends only on the squares of the coordinates, e. g., substituting

$$x^2 \to \sin^2(\pi x/x_L) \,, \tag{9c}$$

with $x_L = \mathtt{nx}\,\Delta x$ the period interval, will provide the proper behavior as the sine squared has a period of π and there is no unphysical decrease of this function near the boundary. Of course the analogous transformation has to be applied to y and z.

The time dependence $f(t)$ is modeled as a short pulse centered around some excitation frequency ω. The code allows a choice between two pulse envelopes:

1. A Gaussian of the form
$$f(t) = \exp\left(-(t-\tau_0)^2/\Delta\tau^2\right)\cos(\omega(\tau-\tau_0)) \ , \tag{9d}$$
with peak time τ_0 and width $\Delta\tau$.
2. A cosine squared function defined via
$$f(t) = \cos\left(\frac{\pi}{2}\left(\frac{t-\tau_0}{\Delta\tau}\right)^2\right)\theta\left(\Delta\tau - |t-\tau_0|\right)\cos(\omega(\tau-\tau_0)) \ , \tag{9e}$$
which is confined to the intervals $t \in (\tau_0 - \Delta\tau, \tau_0 + \Delta\tau)$. This envelope is also characterized by a peak time of τ_0 and width $\Delta\tau$.

Broad envelopes provide a high frequency resolution and so concentrate the excitation around the driving frequency ω. Short pulses lose frequency selectivity and excite a broad band of frequencies.

The extreme case is an infinitely short pulse, $\Delta\tau \longrightarrow 0$. It amounts eventually to an instantaneous boost of the initial wave functions which can be expressed as a phase factor according to
$$\psi_k(\vec{r}, s, t{=}0) = \psi_{k,0}(\vec{r}, s)\,\exp\left(-\mathrm{i}\eta F_q(\vec{r})\right) \ , \tag{10}$$
where $\psi_{k,0}$ stands for the stationary wave function before boost. This instantaneous boost, being infinitely short, is insensitive to the problem of asymptotically unstable potentials and allows the use of a driving field F_q without damping (9b) which, in turn, simplifies spectral analysis (see Section 2.5.3).

The effect of the boost (10) can be understood by virtue of the Madelung decomposition of a complex wave function $\phi(\vec{r}) = \chi(\vec{r})\exp(\mathrm{i}S(\vec{r}))$ with χ and S being real. A straightforward calculation leads to the probability flow density as $\vec{j} = \hbar\chi^2\nabla S/m$. Dividing by the density χ^2 then produces the "probability-flow velocity" $\vec{v} = \hbar\nabla S/m$. An illustrative example is the plane wave where χ =constant and $S = \vec{k}\cdot\vec{r}$ which yields the correct result $\vec{v} = \hbar\vec{k}/m$. In the present case, assuming that the static wave functions themselves can be written as real functions, we get an initial velocity $\vec{v} = -\nabla F_q$, i. e., just in the direction of the classical force resulting from the "velocity potential" $F_q(\mathbf{r})$.

2.4. Static Hartree-Fock

2.4.1. The coupled mean-field and BCS equations

The stationary equations are obtained variationally. Variation with respect to the single-particle wave functions ψ_α yields the mean field equations [2, 51]
$$\hat{h}\psi_\alpha = \varepsilon_\alpha\psi_\alpha \ , \tag{11a}$$

where $\hat{h}$ is the mean-field Hamiltonian (8a) and ε_α is the single-particle energy of state α. Simultaneous variation of ψ_α together with the occupation amplitude v_α yields the Hartree-Fock-Bogolyubov equations [4, 51, 52] which complicate Eq. (11a) by non-diagonal terms on the right-hand-side [53]. We employ here the BCS approximation which exploits time-reversal symmetry where each single-particle state has a time reversed partner $\psi_\alpha \leftrightarrow \psi_{\overline{\alpha}}$ as was already implied in the pairing densities ξ_q in Eq. (4). Each pair of time-conjugate states is associated with an occupation amplitude v_α. These are determined by the BCS equation $\left(\varepsilon_\alpha - \epsilon_{\mathrm{F},q_\alpha}\right)\left(u_\alpha^2 - v_\alpha^2\right) = \Delta_\alpha u_\alpha v_\alpha$ which can be resolved to a closed expression for the occupation amplitudes as

$$v_\alpha^2 = \frac{1}{2}\left(1 - \frac{\varepsilon_\alpha - \epsilon_{\mathrm{F},q_\alpha}}{\sqrt{(\varepsilon_\alpha - \epsilon_{\mathrm{F},q_\alpha})^2 + \Delta_\alpha^2}}\right) \quad , \tag{11b}$$

$$\Delta_\alpha = \frac{1}{2}V_{\mathrm{pair,q_\alpha}} \int \mathrm{d}^3r\, \psi_\alpha^+ \psi_\alpha \xi_{q_\alpha} \left[1 - \frac{\rho}{\rho_{0,\mathrm{pair}}}\right] \quad , \tag{11c}$$

$$\epsilon_{\mathrm{F},q} \; : \; \sum_{\alpha \in q} v_\alpha^2 = N_q \quad . \tag{11d}$$

q_α denotes the nucleon type to which state α belongs, $\alpha \in q$ means all states of type q, and N_q is the number of nucleons of type q (identified as $N_p = Z$ and $N_n = N$). The Fermi energies $\epsilon_{\mathrm{F},q_\alpha}$ serve to regulate the average particle number to the required values N_q. Here, the space of pairing-active states is just the space of states actually included in the calculation. The results of BCS pairing depend slightly on the size of the active space [51, 52]. We recommend using about

$$N_q + \frac{5}{3}N_q^{2/3}$$

single-nucleon states, which comes closest to the dynamical pairing space of Ref. [54].

2.4.2. Iterative solution

The coupled mean-field and BCS equations (11) are solved iteratively. The wave functions are iterated with a gradient step which is accelerated by kinetic-energy damping [55, 56]

$$\psi_\alpha^{(n+1)} = \mathcal{O}\left\{\psi_\alpha^{(n)} - \frac{\delta}{\hat{T} + E_0}\left(\hat{h}^{(n)} - \langle\psi_\alpha^{(n)}|\hat{h}^{(n)}|\psi_\alpha^{(n)}\rangle\right)\psi_\alpha^{(n)}\right\} \tag{12}$$

where $\hat{T} = \hat{p}^2/(2m)$ is the operator of kinetic energy, $\mathcal{O}$ means orthonormalization of the whole set of new wave functions, and the upper index indicates the iteration number. Note that this sort of kinetic-energy damping is particularly suited for the fast Fourier techniques that we use in the present code. The damped gradient step has two numerical parameters, the step size δ and the damping regulator E_0. The latter should be chosen typically of the order of the depth of the local potential U_q. In practice, we find $E_0 = 100$ MeV a safe choice. The step size is of order of $\delta = 0.1...0.8$. Larger values yield faster iteration, but can run more easily into pathological conditions. The optimum choice depends somewhat on the Skyrme parametrization. Those with effective mass $m^*/m \approx 1$ allow larger δ values. Low m^*/m may require a reduction in the step size.

After each such wave function step, the BCS equations (11b–11d) are solved with $\varepsilon_\alpha = \langle\psi_\alpha|\hat{h}|\psi_\alpha\rangle$, the densities are updated (Eqs. 2-4), and new mean fields computed (8a-8f). This then provides the starting point for the next iteration. The process is continued until sufficient convergence is achieved. We consider as the convergence criterion the average energy variance, or fluctuation, of the single particle states

$$\overline{\Delta\varepsilon} = \sqrt{\frac{\sum_\alpha \Delta\varepsilon_\alpha^2}{\sum_\alpha 1}} \quad , \tag{13a}$$

$$\Delta\varepsilon_\alpha^2 = \langle\psi_\alpha|\hat{h}^2|\psi_\alpha\rangle - \varepsilon_\alpha^2 \quad , \tag{13b}$$

$$\varepsilon_\alpha = \langle\psi_\alpha|\hat{h}|\psi_\alpha\rangle \quad . \tag{13c}$$

The single particle energy ε_α is defined here as an expectation value. It finally becomes an eigenvalue in Eq. (11a) if the iteration has converged to $\Delta\varepsilon_\alpha \approx 0$. Vanishing total variance $\overline{\Delta\varepsilon}$ signals that we have reached minimum energy, i.e. a solution of the mean-field plus BCS equations. One has to be aware, however, that this may be only a local minimum (isomeric state). It requires experience to judge whether one has found the absolute energy minimum. In case of doubt, one should redo a couple of static iterations from very different initial configurations.

This raises the question of how to initialize the iteration. We take as a starting point the wave functions of the deformed harmonic oscillator (see point 1 in Section 2.8). These are characterized by $\vec{n} = (n_x, n_y, n_z)$, the number of nodes in each direction. We stack the wave functions in order of increasing oscillator energy $\epsilon_\alpha^{(0)} = \hbar\omega_x n_x + \hbar\omega_y n_y + \hbar\omega_z n_z$ and stop if the desired number of states is reached. The deformation of the initializing oscillator influences the initial state in two ways: first, through the deformation of the basis wave functions as such, and second, through the energy ordering of the $\epsilon_\alpha^{(0)}$ and corresponding sequence of levels built. Variation of initial conditions means basically a variation of the oscillator deformation. For example, the iteration will most probably end up in a prolate minimum if the initial state was sufficiently prolate, and in an oblate minimum after an oblate initial state. It depends on the nucleus which one is the absolute minimum.

2.5. TDHF

2.5.1. The time-dependent mean-field equations

The TDHF equations are determined from the variation of the action

$$S = \int \mathrm{d}t \left[E[\{\psi_\alpha\}] - \sum_\alpha \langle\psi_\alpha|\mathrm{i}\partial_t|\psi_\alpha\rangle \right] \quad ,$$

with respect to the wave functions ψ_α^+ where the energy is given as in Eq. (5a-5h) [2]. This yields the TDHF equation

$$\mathrm{i}\partial_t\psi_\alpha = \hat{h}\psi_\alpha \; , \tag{14}$$

where $\hat{h}$ is, again, the mean-field Hamiltonian (8a). For simplicity, we are not considering variation of the occupation amplitude in the time-dependent case. The occupation amplitudes obtained from static iteration are kept frozen during time evolution. For studies of mean-field flow at moderate excitations (heavy-ion collisions, giant resonances) this approximation is legitimate. However, a study of truly low energy dynamics in the range of a few MeV (soft vibrations, fission) requires a full time-dependent Hartree-Fock-Bogolyubov treatment and should not be undertaken with the present code.

2.5.2. Time development algorithm

The TDHF equation (14) can be formally resolved into an integral equation as

$$|\psi_\alpha(t+\Delta t)\rangle = \hat{U}(t,t+\Delta t)|\psi_\alpha(t)\rangle \tag{15a}$$

$$\hat{U}(t,t+\Delta t) = \hat{\mathcal{T}}\exp\left(-\frac{\mathrm{i}}{\hbar}\int_t^{t+\Delta t}\hat{h}(t')\,\mathrm{d}t'\right) , \tag{15b}$$

where $\hat{U}$ is the time-evolution operator and $\hat{\mathcal{T}}$ the time-ordering operation. This time evolution is unitary, thus conserving orthonormalization of the single-particle wave functions, and it conserves the total energy (5a) provided that there are no time-dependent external fields. To convert this involved operator into an efficiently computable but also sufficiently accurate form a predictor-corrector strategy is used:

1. In a first step (predictor), we determine the single-particle Hamiltonian at midtime $\hat{h}(t+\Delta t/2)$. To that end, a trial step by Δt

$$\tilde{\psi}_\alpha = \exp\left(-\tfrac{\mathrm{i}}{\hbar}\hat{h}(t)\,\Delta t\right)\psi_\alpha(t) \tag{16}$$

   is performed using the mean field $\hat{h}(t)$ at initial time t. The density $\tilde{\rho}$ and similarly also other densities and currents at $t+\Delta t$ are accumulated from the wave functions $\tilde{\psi}_\alpha$, which can be discarded immediately, so that it is not necessary to store a complete second set of wave functions. They are used to compute an estimate for the densities at half step $\rho_{\mathrm{pre}} = (\rho(t)+\tilde{\rho})/2$ and analogously for the other densities and currents. These are then used to calculate the estimated Hamiltonian $\hat{h}_{\mathrm{pre}}$ at $t+\Delta t/2$ according to Eq. (8a-8f).

2. This is used to perform the corrector step to advance the wave functions to the end of the time step (again with frozen Hamiltonian, but now $\hat{h}_{\mathrm{pre}}$)

$$\psi_\alpha(t+\Delta t) = \exp\left(-\tfrac{\mathrm{i}}{\hbar}\hat{h}_{\mathrm{pre}}\,\Delta t\right)\psi_\alpha(t) . \tag{17}$$

3. In both cases the operator exponential is evaluated by a Taylor series expansion up to order m:

$$\exp\left(-\tfrac{\mathrm{i}}{\hbar}\hat{h}\,\Delta t\right)\psi \approx \sum_{n=0}^{m}\frac{(-\mathrm{i}\Delta t)^n}{\hbar^n\,n!}\hat{h}^n\psi , \tag{18}$$

   where $\hat{h}$ is the actual mean field in step (16), or (17) respectively. $\hat{h}^n\psi$ is computed in straightforward manner by successive application of the mean field Hamiltonian, i.e. $\hat{h}^n\psi = \underbrace{\hat{h}(...(\hat{h}}_{n\text{times}}\psi)...)$.

The Taylor expansion spoils strict unitarity of the exponential $\exp\left(-\tfrac{\mathrm{i}}{\hbar}\hat{h}\,\Delta t\right)$ and energy conservation. We turn this flaw into an advantage and use norm conservation as well as energy conservation (if it applies) as counter-check of the quality of the step along the propagation. The reliability depends, of course, on a proper choice of the numerical parameters in this step which are the step size Δt and the order of the Taylor expansion m. The step size is limited by the maximum possible kinetic energy and by the typical time scales of changes in the mean field $\hat{h}$. The maximum kinetic energy, in turn, depends on the grid spacing as $\propto \Delta x^{-2}$. A choice of $\Delta t = 0.1-0.2$fm/c is applicable in connection with $\Delta x = 0.7 - 1$/fm. For the order of Taylor expansion, one needs at least $m = 4$. Although there are formal reasons for $m = 4$ [57], in practice $m > 4$ may also be used, but choosing $m > 6$ is not so efficient for the values of Δt considered here.

2.5.3. Collective excitations

Giant resonances are prominent excitation modes of nuclei. Best known is probably the isovector giant dipole resonance, but there are many more modes depending on isospin and angular momentum. The typical resonance energies lie in a region from 10 to 30 MeV where the present TDHF code with frozen occupation numbers is applicable because the relevant energy range lies far above the pairing gap (1–2 MeV). The generation of these modes is particularly simple within the present TDHF treatment. One first produces a stationary state as outlined in Section 2.4 and then triggers the excitation by a time-dependent external field as described in Section 2.3. A broad pulse allows triggering particular excitation energies. An infinitely short pulse amounts to an instantaneous boost.

The boost is a generic excitation of a system which gives the same weight to all frequencies. It is thus ideally suited for analyzing in an unbiased manner the excitation spectra of a system. This, in turn, allows a thorough spectral analysis. To obtain the spectral distribution of isovector dipole strength, one applies a boost with small strength η and $F_q = \hat{D} \propto r^1 Y_{10}\tau_z$ the isovector dipole operator. The Slater determinant $|\Phi(t)\rangle$ is propagated in TDHF for a sufficiently long time while recording the dipole moment $D(t) = \langle\Phi(t)|\hat{D}|\Phi(t)\rangle$. The dipole strength is finally extracted from the Fourier transform $\tilde{D}(\omega)$ as $S_D(\omega) = \Im\left\{\tilde{D}(\omega)\right\}/\eta$. Note that this is valid only in the linear case, i. e., if the amplitude of the vibration if proportional to the boost velocity [59, 60]. This should be checked in the calculations.

The straightforward Fourier transform leads to artifacts if the dipole signal has not fully died out at the end of the simulation time. In the general case, some filtering is necessary to suppress artifacts from cutting the signal at a certain final time [58]. In practice, it is most convenient to use filtering in the time domain by damping the signal $D(t)$ towards the final time. A robust choice is

$$D(t) \quad \longrightarrow \quad D_{\mathrm{fil}}(t) = D(t)\cos\left(\frac{\pi}{2}\frac{t}{t_{\mathrm{final}}}\right)^{n_{\mathrm{fil}}} \tag{19}$$

where t_{final} is the final time of the simulation. This guarantees that the effective signal D_{fil} vanishes at the end of the interval. The $\cos^n$ profile switches off gently and leaves as much as possible from the relevant signal at early times. The parameter n_{fil} determines the strengths of filtering. Value of order of 4–6 are recommended to suppress the artifacts safely. For a detailed description of this spectral analysis see [59]. For typical applications in nuclear physics see [60]. It is to be noted that the code does not include this final step of spectral analysis. The time dependent signals are printed on the protocol files `monopoles.res`, `dipoles.res`, and `quadrupoles.res` (see Section 5.17.3). It is left to the user to perform the final steps towards a spectral distribution. A word is in order about t_{final}. It determines the resolution of the spectral analysis. The corresponding energy bins are given by $\delta E_{\text{exc}} = \hbar\pi/t_{\text{final}}$. Windowing effectively reduces the time span in which relevant information is contained and roughly doubles the relevant δE_{exc}. For example, to obtain a spectral resolution of 1 MeV, one needs to simulate up to about 1200 fm/c.

Although excitation spectra are one of the most basic properties of the system, there are many other dynamical features of interest. The multipole signals in the time domain (printed in the protocol files) are as such interesting quantities. One can have, e.g., a look at cross-talk between the multipole channels. It is particularly interesting to study excitation dynamics for varying excitation strength η, from the regime of linear response (small η) deep into the non-linear regime. It is inefficient to perform a full three-dimensional TDHF calculation to obtain linear-regime excitation spectra for spherical nuclei. This is better done in a dedicated RPA calculation on a spherical basis (see, e.g., [61]) for which there exist an overwhelming multitude of codes. The realm of TDHF calculations of nuclear excitations are spectra in deformed systems, stability analysis of exotic configurations, and in particular non-linear dynamics.

There are many more details worth looking at. One may check the densities and currents to visualize the flow pattern associated with a mode. A most elaborate analysis deals with a phase-space picture of nuclear dynamics by virtue of the Wigner transformation [62]. The code allows saving all ingredients needed for such elaborate analysis in dedicated output files, see Section 8.5. It is left to the user to work out the further steps of the analysis.

A serious problem that can occur in collective excitation studies is the evaporation of particles at higher excitation energies. Some of the single-particle wave functions become unbound and move towards the boundaries for the computational box with a non-negligible part of their probability distributions. Since the code assumes periodic boundary conditions for the wave functions (for details see Sect 2.7.4), effectively the particles reenter the box from the opposite side and can interact with the nucleus again, leading to a spurious revival of the excitation [20, 63]. This effect can be reduced in several ways; for a brief discussion see Sect. 2.7.4.

2.5.4. Nuclear reactions

Collisions of nuclei are a prime application of nuclear TDHF. They were, in fact, the major motivation for its realization [10, 64]. The present code is designed to initialize such collision scenarios in a most flexible manner. We start by explaining the simplest case of a collision of two nuclei. First, we prepare the ground states of the two nuclei as explained in Section 2.4.2. The static solutions are centered around the origin $\vec{r} = 0$ of their initial grid. The static wave functions $\psi_{\alpha,I}^{(\text{stat})}$ where $I = 1, 2$ labels the two nuclei are shifted to new centers $\vec{R}_I$ where the distance $|\vec{R}_2 - \vec{R}_1|$ should be sufficiently large that the nuclei have negligible overlap and negligible Coulomb distortion from the other nucleus (the latter condition usually only loosely fulfilled). The shifted wave functions $\psi_{\alpha,I}^{(\text{stat})}(\vec{r} - \vec{R}_I, s)$ are obtained by interpolation on the grid. The interpolation is done by transforming to momentum space, applying translation factors, and going back. It is obvious that the collisions need a larger numerical box than the static Hartree-Fock calculations. Thus, we may compute the static wave functions on a smaller box since we are shifting and interpolating the result anyway for dynamical initialization.

It should be noted that the nuclei may be placed at arbitrary positions in the new grid. It is highly recommended, however, to displace them by an integer number of grid points from the original position, since otherwise the interpolation may lead to some degree of excitation. In practice, if the center of mass in the static calculation was at the origin, the new position should of the form (m_x `dx`, m_y `dy`, m_z `dz`) with integer factors m_x etc. If for some reason this is not desired, accuracy can be restored by placing the nucleus *alone* in the larger grid and running this as input for a static calculation with a sufficient number of timesteps before using the resulting wave functions as input for the dynamic one.

At this point we have the nuclei resting at a safe distance. To set them in motion we need to give each nucleus a momentum $\vec{P}_1 = -\vec{P}_2$ (note that the total momentum of the combined system still vanishes). Consequently, the initial configuration is given by the Slater state built from the shifted and boosted single-particle wave functions (see fragment initialization, point 2 in Section 2.8)

$$\begin{aligned}\psi_{\alpha,1}(\vec{r},s;t\!=\!0) &= e^{\mathrm{i}\vec{p}_1\cdot\vec{r}}\psi^{(\mathrm{stat})}_{\alpha,1}(\vec{r}-\vec{R}_1,s), \quad \vec{p}_1 = \tfrac{\vec{P}_1}{A_1} \quad ,\\ \psi_{\alpha,2}(\vec{r},s;t\!=\!0) &= e^{\mathrm{i}\vec{p}_2\cdot\vec{r}}\psi^{(\mathrm{stat})}_{\alpha,2}(\vec{r}-\vec{R}_2,s), \quad \vec{p}_2 = \tfrac{\vec{P}_2}{A_2} \quad .\end{aligned} \tag{20}$$

The distance between the nuclei is large, but inevitably finite. This may induce minor violations of orthonormality. Thus the full set of wave functions (20) is orthonormalized as a final step of initial preparation.

The occupation amplitudes $v_{\alpha,I}$ are taken over from the static solution and frozen along the dynamical evolution. In fact, most of the collision studies will be principally to explore the dynamical features in the regimes of fusion and inelastic collisions. It is then recommended to use the conceptually simplest and most robust strategy, namely to start from simple Slater states (not BCS states) for the two nuclei. This means that in most cases the static solution is calculated without pairing, fixing $v_\alpha = 1$ and including just as many states as there are nucleons.

The above example deals with two initial fragments. The code is more flexible than that. It allows an initial state composed from several fragments. The strategy remains the same as for the binary system. It is simply repeated for each new fragment. For details see Section 5.9.

The time evolution is performed as outlined in Section 2.5.2. It requires some effort to visualize the complex dynamics which emerges in collisions. A rough picture is, again, provided by the multipole moments. The quadrupole moment, e.g., can serve as a measure of stretching of the total system. Small values indicate a compound nucleus while asymptotically growing values signal fragmentation. One may want, particularly in case of collisions, more detailed pictures of the flow as, e.g., snapshots of the density of current distributions and, ultimately, a full phase space picture [62]. Material for that can be output on demand, as detailed in Section 8.5. Again, we leave it to the user to extract the wanted information and to prepare it for visualization.

In nuclear collisions at higher energies the emission of particles can also cause problems; in those cases one better uses absorbing boundary conditions. See Sect. 2.7.4 for details.

2.6. Observables

It was already mentioned in Sections 2.5.3 and 2.5.4 which observables may be used to analyze nuclear dynamics. We here briefly summarize the observables computed and output in the code and indicate how further observables may be extracted. Basic features of the description by the Skyrme energy-density functional are, of course, energy and densities.

2.6.1. Multipole moments

The gross features of the density distribution are well characterized by the multipole moments. The most important moment is the center of mass (c.m.)

$$\vec{R}^{(\mathrm{type})} = \frac{\int \mathrm{d}^3r\,\vec{r}\,\rho^{(\mathrm{type})}(\vec{r})}{A} \quad , \tag{21a}$$

where $A = \int \mathrm{d}^3r\,\rho(\vec{r})$ is the total mass number and "type" can refer to proton c.m. from ρ_p, neutron c.m. from ρ_n, isoscalar or total c.m. from the total density $\rho = \rho_p + \rho_n \equiv \rho_{T=0}$, or isovector moment related to the isovector density $\rho_{T=1} = \frac{N}{A}\rho_p - \frac{Z}{A}\rho_n$. The definition of $\vec{R}^{(\mathrm{type})}$ directly employs the Cartesian coordinate r_i. The same holds for the Cartesian quadrupole tensor

$$\mathcal{Q}_{kl}^{(\mathrm{type})} = \int \mathrm{d}^3r \left(3(r_k - R_k)(r_l - R_l) - \delta_{kl}\sum_i (r_i - R_i)^2\right)\rho^{(\mathrm{type})}(\vec{r}) \quad , \tag{21b}$$

again for the various types as discussed above. The matrix $\mathcal{Q}_{kl}$ is not invariant under rotations of the coordinate frame. There is a preferred coordinate system: the system of principle axes. It is obtained by diagonalizing $\mathcal{Q}_{kl}$. The quadrupole matrix in the principle-axis frame thus has only three non-vanishing entries Q_{xx}, Q_{yy}, and Q_{zz} together with the trace condition $Q_{xx} + Q_{yy} + Q_{zz} = 0$.

For the quadrupole case the spherical moments defined as

$$Q_{2m}^{(\mathrm{type})} = \int \mathrm{d}^3r\, r^2 Y_{2m}\,\rho^{(\mathrm{type})}(\vec{r} - \vec{R}) \; , \tag{21c}$$

are also useful, where $r = |\vec{r}|$ and Y_{2m} are the spherical harmonics. They are often expressed as dimensionless quadrupole moments

$$a_m = \frac{4\pi}{5}\frac{Q_{2m}}{AR^2} \quad , \tag{21d}$$

with $R = r_0 A^{1/3}$ a fixed radius derived from the total mass number A. This, again, could be defined for any "type", but is used, in practice, mainly for the isoscalar moments.

The dimensionless moments have the advantage of being free of an overall scale which was removed by the denominator AR^2. They allow characterization of the shape of the nucleus. However, the general a_m are not invariant under rotations of the coordinate frame. We obtain a unique characterization by transforming to the principle-axis system. These are defined by the conditions $a_{\pm 1} = 0$ and $a_2 = a_{-2}$. There remain only two shape parameters a_0 and a_2. These are often reexpressed as total deformation β and triaxiality γ, often called Bohr-Mottelson parameters, through

$$\beta = \sqrt{a_0^2 + 2a_2^2} \quad , \quad \gamma = \mathrm{atan}\left(\frac{\sqrt{2}\,a_2}{a_0}\right) \quad . \tag{21e}$$

Triaxiality γ is handled like an angle. It can, in principle, take all values between 0^o and 360^o, but physically relevant parameters stay in the $0\ldots60^\circ$ range. The other sectors correspond to equivalent configurations [51].

The monopole moment just corresponds to the total particle number, so to describe monopole vibrations usually the r.m.s. radii or their squares are employed. The r.m.s. radii are defined as

$$r_{\mathrm{rms}}^{(\mathrm{type})} = \sqrt{\frac{\int \mathrm{d}^3r\,(\vec{r}-\vec{R})^2\,\rho^{(\mathrm{type})}(\vec{r})}{\int \mathrm{d}^3r\,\rho^{(\mathrm{type})}(\vec{r})}} \tag{21f}$$

where “type” can be proton, neutron, or total. The isovector variant does not make sense here.

We supply printouts of all the above variants of multipole moments to allow a most flexible analysis. For the reason given above the r.m.s. radii are output in a file called `monopolesfile`.

2.6.2. Alternative way to evaluate the total energy

The key observable is the total energy E_{tot}. It is computed as given in Eqs. (5a-5h). More detailed energy observables are provided by the s.p. energies (13c). These can also be used to compute the total energy. The traditional HF scheme deals with pure two-body interactions and exploits that to simplify [51]

$$E_{\mathrm{tot,HF}} = \frac{1}{2}\sum_\alpha (t_\alpha + \epsilon_\alpha) \tag{22}$$

where $t_\alpha = \langle\psi_\alpha|\hat{T}|\psi_\alpha\rangle$ is the s.p. kinetic energy. This is possible because

$$\epsilon_\alpha = t_\alpha + u_\alpha \quad , \quad u_\alpha = \sum_\beta [v_{\alpha\beta\alpha\beta} - v_{\alpha\beta\beta\alpha}] - \tfrac{1}{2}v_\alpha = \epsilon_\alpha - t_\alpha \quad ,$$

where u_α is the s.p. mean-field potential energy and v the two-body interaction, and

$$E_{\mathrm{tot,HF}} = \sum_\alpha t_\alpha + \frac{1}{2}\sum_{\alpha\beta}[v_{\alpha\beta\alpha\beta} - v_{\alpha\beta\beta\alpha}] \quad .$$

The Skyrme force does not simply have this two-body structure. Still the total energy is very often computed along the strategy of Eq. (22). However, the density dependence requires augmenting this recipe by a rearrangement energy which accounts for a contribution missing in the simple recipe (22). The extension to the Skyrme energy thus reads

$$
\begin{aligned}
E_{\mathrm{tot,HF}} &= \frac{1}{2}\sum_\alpha (t_\alpha + \epsilon_\alpha) + E_{3,\mathrm{corr}} + E_{\mathrm{C,corr}} \quad , && (23\mathrm{a})\\
E_{3,\mathrm{corr}} &= \int \mathrm{d}^3 r \frac{\alpha}{6}\rho^\alpha \left[b_3\rho^2 - b_3'(\rho_p^2+\rho_n^2)\right] \quad , && (23\mathrm{b})\\
E_{\mathrm{C,corr}} &= \frac{1}{4}\left(\frac{3}{\pi}\right)^{1/3} \int \mathrm{d}^3 r\, \rho_{pr}^{4/3} \quad . && (23\mathrm{c})
\end{aligned}
$$

In the code the total energy is computed both ways, from the straightforward Skyrme energy (5a) as well as from the above recipe (23), as described in Section 5.5.1. Numerically these values are close but not identical.

2.7. Discretization

2.7.1. Data types

In the module `Params` a type `db` is defined for 12-digit accuracy and on all present machines should amount to double precision, i. e., `REAL(db)` and `COMPLEX(db)` are actually `REAL(8)` and `COMPLEX(8)` for most compilers. Keeping the symbolic type throughout of course makes the code more flexible for future hardware. Using single precision is not recommended.

Since the external libraries are based on C or Fortran-77 coding, special care has to be taken in this respect. The FFTW3 library stores its plans in 8-byte integers, which in the modules `Coulomb` and `Fourier` are defined using the type `C_LONG` from the system-supplied module `ISO_C_BINDING`. If this is not available, they may be defined as `INTEGER(8)` or if that also causes problems, `DOUBLE PRECISION`, which certainly corresponds to at least 8 bytes. For the `LAPACK` routines, double-precision real and complex variables are necessary, which we also define using “`db''`. If `db` should ever be changed in such a way that `REAL(db)` no longer corresponds to 8 bytes, different `LAPACK` routines must be selected.

Note that data conversion needs some care. If `a` and `b` are double precision, `CMPLX(a,b)` is not; according to the standard it returns the default accuracy, which on many machines will still be single precision. Thus that for example in `EXP(CMPLX(a,b))` the exponential would be evaluated in single precision. That is why in the code the expression `CMPLX(a,b,db)` is used consistently; this is also safe for future changes in accuracy.

The other conversion functions used are safe. `AIMAG` is generic and `REAL` reproduces the accuracy of (only) a *complex argument.*

2.7.2. Grid definition

All wave functions and fields are defined on a three-dimensional regular Cartesian grid of `nx` by `ny` by `nz` grid points. **`nx`, `ny`, and `nz` must be even numbers**. The physical spacing between the points is given as `dx`, `dy`, and `dz` (in fm). In principle these could be different for the three directions, but since this will lead to a loss of accuracy it is highly recommended to give the same value to all of them. A typical range is 0.5–1.0 fm.

The grid is automatically arranged in such a way that in each direction the same number of grid points are located on both sides of the origin. This means that the three-dimensional origin is in the center of a cubic cell and has the advantage that exact parity properties for the wave functions can be maintained. The coordinate values for e. g., the x-direction are thus:

$$-\frac{\mathtt{nx}-1}{2}\mathtt{dx},\quad -\frac{\mathtt{nx}-1}{2}\mathtt{dx}+\mathtt{dx},\quad \ldots\quad -\frac{\mathtt{dx}}{2},\\ +\frac{\mathtt{dx}}{2},\quad \ldots\quad \frac{\mathtt{nx}-1}{2}\mathtt{dx}. \tag{24}$$

The corresponding values are available in the arrays `x(nx)`, `y(ny)`, and `z(nz)`.

2.7.3. Derivatives

The computation of the kinetic densities and currents and the application of the mean-field Hamiltonian require first and second derivatives at several places in the code. We define them in Fourier space. For simplicity, the strategy is explained here for one dimension. The generalization to 3D is obvious.

The `nx` discrete grid points x_ν in coordinate space are related to the same number of grid points k_n in Fourier space (physically equivalent to momentum space) as

$$x_\nu = \left(-\frac{\mathtt{nx}-1}{2}+\nu\right)\mathtt{dx} \quad ,\nu=1,...,\mathtt{nx} \quad , \tag{25a}$$

$$k_n = (n-1)\mathtt{dk}, \quad n=1,\ldots \mathtt{nx}/2 \quad , \tag{25b}$$

$$k_n = (n-\mathtt{nx}-1)\,\mathtt{dk}, \quad n=\mathtt{nx}/2+1,\ldots,\mathtt{nx} \quad ,$$

$$\mathtt{dk} = \frac{2\pi}{\mathtt{nx}\cdot\mathtt{dx}} \quad . \tag{25c}$$

Note the particular indexing for the k-values. In principle, the values $k_n = (n-1)\mathtt{dk}$ for all n are equivalent for the Fourier transform, but for the second half of this range the negative k-values should be chosen because of their smaller magnitude. For the Fourier expansion, $k=-\mathtt{dk}$ and $k=(\mathtt{nx}-1)\mathtt{dk}$ are equivalent because of periodicity in k-space.

A function $f(x_\nu)$ in coordinate space is connected to a function $\tilde{f}(k_n)$ in Fourier space by

$$\tilde{f}(k_n) = \sum_{\nu=1}^{\mathtt{nx}} \exp\left(-\mathrm{i}k_n x_\nu\right) f(x_\nu) \quad , \tag{25d}$$

$$f(x_\nu) = \frac{1}{\mathtt{nx}}\sum_{n=1}^{\mathtt{nx}} \exp\left(\mathrm{i}k_n x_\nu\right)\tilde{f}(k_n) \tag{25e}$$

This complex Fourier representation implies that the function f is periodic with $f(x+\mathtt{dx}\cdot\mathtt{nx}) = f(x)$. The appropriate integration scheme is the trapezoidal rule which complies with the above summations adding up all terms with equal weight. The derivatives of the exponential basis functions are

$$\frac{\mathrm{d}^m}{\mathrm{d}x^m}\exp\left(\mathrm{i}k_n x\right) = (\mathrm{i}k_n)^m \exp\left(\mathrm{i}k_n x\right) \quad . \tag{26}$$

Computation of the mth derivative thus becomes a trivial multiplication by $(\mathrm{i}k_n)^m$ in Fourier space. Time critical derivatives are best evaluated in Fourier space using the fast Fourier transformation (FFT). To that end a forward transform (25d) is performed, then the values $\tilde{f}(k_n)$ are multiplied by $(\mathrm{i}k_n)^m$ as given in Eq. (26) and finally transformed $(\mathrm{i}k_n)^m\tilde{f}(k_n)$ back to coordinate space by the transformation (25e). This strategy is coded in the subroutines `cdervx`, `cdervy`, and `cdervz` contained in module `Levels`. It is used for derivatives of wave functions provided the switch `TFFT` is set.

For coding purposes, it is often useful to perform derivatives as a matrix operation directly in coordinate space. The derivative matrices are built by evaluating the double summation of forward and backward transform ahead of time. For the mth derivative this reads

$$\begin{aligned}
f^{(m)}(x_\nu) &= \frac{1}{\mathtt{nx}}\sum_n \exp\left(ik_n x_\nu\right)(\mathrm{i}k_n)^m \sum_{\nu'} \exp\left(-\mathrm{i}k_n x_{\nu'}\right) f(x_{\nu'}) \\
&= \sum_{\nu'} \underbrace{\frac{1}{\mathtt{nx}}\sum_n \exp\left(\mathrm{i}k_n x_\nu\right)(\mathrm{i}k_n)^m \exp\left(-\mathrm{i}k_n x_{\nu'}\right)}_{D^{(m)}_{\nu\nu'}} f(x_{\nu'}) \quad .
\end{aligned}$$

From here, the detailed handling depends on the order of derivative. The k_n run over the values $k_n = 0, \pm\mathtt{dk}, \pm2\mathtt{dk}, ..., (\mathtt{nx}/2-1)\mathtt{dk}, +\mathtt{dk}\,\mathtt{nx}/2$. Here the index ordering given in Eq. (25b) does not matter as the index is summed over. Note that the first and the last value come alone while all others come in pairs of $\pm$ partners. These pairwise terms can be combined into a sine function for $n=1$ and a cosine for $n=2$. The derivative matrices thus read in detail

$$\begin{aligned}
D^{(1)}_{\nu\nu'} =& -\frac{2\mathtt{dk}}{\mathtt{nx}}\sum_{n=1}^{\mathtt{nx}/2-1} n\sin(k_n\mathtt{dx}(\nu-\nu')) \\
& -\frac{\mathtt{dk}}{\mathtt{nx}}\frac{\mathtt{nx}}{2}\sin((\nu-\nu')\mathtt{dk}\,\mathtt{nx}/2) \quad , && \text{(27a)}\\
D^{(2)}_{\nu\nu'} =& -\frac{2\mathtt{dk}^2}{\mathtt{nx}}\sum_{n=1}^{\mathtt{nx}/2-1} n^2\cos(k_n\mathtt{dx}(\nu-\nu')) \\
& -\frac{\mathtt{dk}^2}{\mathtt{nx}}\left(\frac{\mathtt{nx}}{2}\right)^2\cos((\nu-\nu')\mathtt{dk}\,\mathtt{nx}/2) \quad . && \text{(27b)}
\end{aligned}$$

A word is in order about the first derivative. The upper point in the k-grid, $\mathtt{dk}\,\mathtt{nx}/2$, is ambiguous. Exploiting periodicity, it could be equally well $-\mathtt{dk}\,\mathtt{nx}/2$. In order, to deal with a $\pm k$ symmetric derivative we have anti-symmetrized this last point. The price for this is a slight violation of hermiticity which, however, should be very small as we anyway should not have significant wave-function contributions at the upper edge of the k-grid.

The derivative matrices $D^{(m)}_{\nu\nu'}$ can be prepared ahead of time and are then at disposal for any derivative in the course of the program. Actually, the matrices for the first derivative are prepared in routine `sder` and for the second derivative in routine `sder2`, both contained in module `Grids`. These routines are applied to generate the derivative matrices `derv1x`, `derv1y`, `derv1z`, `derv2x`, `derv2y`, and `derv2z`, for first and second derivatives in the x, y and z directions.

The matrix formulation of the derivatives is used in the code in two ways: on the one hand, the derivatives of the real-valued densities, currents, and mean-field components are always calculated using these derivative matrices, because they are real and the Fourier transform method would require converting them to complex values (using special Fourier techniques for real arrays is in principle possible but has not been worked out yet). In addition the user can switch to using the matrix method everywhere, which may give a slight speed advantage for small grid dimensions.

2.7.4. Boundary conditions

The code uses a periodic Fourier transform to calculate derivatives. This is valid only with *periodic boundary conditions*. Thus in principle the wave functions and potentials are assumed to be repeated periodically in each Cartesian direction. Because of the short range of the nuclear force, this is not a serious problem in most cases; at higher energies, however, as mentioned in Sect. 2.5.3 and 2.5.4 the emission of low-density material from the nuclei can interfere with the dynamics in the neighboring box and cause problems in the conservation of energy and angular momentum; for a detailed discussion see [65]. This is aggravated by the fact that even with periodic boundary conditions periodicity is truly fulfilled only for the wave functions and mean-field components. Since the vector $\vec{r}$ itself is not periodic but jumps at the boundary, operators such as the orbital angular momentum are not periodic.

Several ways to solve, or reduce, the problem have been brought up. The most obvious and conceptually simplest approach is to increase the size of the numerical box. This is, however, not an option in 3D calculations as the expense grows cubically with the box length. Very recently, a multigrid method has been proposed [66] which renders the use of enlarged boxes feasible (although still at the boundaries of present days computer capabilities). Perfect removal of escaping particles is achieved by radiating or exact boundary conditions [67–70] which, again, are not yet practicable in 3D calculations. Robust and efficient are absorbing boundary layers using an especially tailored imaginary potential [20] or by applying a mask function during time evolution [71]. The latter technique is particularly easy to implement and has been widely used in the past. Its robustness and efficiency allow developing advanced analyzing techniques on the grid as, e.g., the computation of kinetic-energy spectra and angular distributions of the emitted particles [72]. Those who are not afraid of a little bit of coding can easily implement the mask-function technique for absorbing boundary conditions into the present code. A detailed description and discussion of this approach and the proper choice of numerical parameters is found in [63].

On the other hand, for the Coulomb field with its long range periodicity would be clearly wrong. Therefore a computation of the Coulomb potential for the boundary condition of an isolated charge distribution is implemented in addition to the periodic one; see the manual for the module `Coulomb`. This is selected by the logical input variable `periodic` **and applies only to the Coulomb potential**.

2.7.5. Wave function storage

Module `Levels` handles the single-particle wave functions and associated quantities.

The principal array for the wave function is called `psi`, which is of type `COMPLEX(db)`. Its dimension is `(nx,ny,nz,2,nstloc)`, where the first three indices naturally refer to the spatial position. The 4th index corresponds to spin: index 1 refers to spin up and 2 to spin down, quantization being along the z-direction.

The last index numbers the wave functions. If the code is run on a single node, the value is `nstmax`, the total number of single-particle wave functions. They are divided up into neutron and proton states, with the index range given by `npmin` and `npsi`. The sub-ranges are:

- `npmin(1)`...`npsi(1)` : the neutron states,
- `npmin(2)`...`npsi(2)` : the proton states.

In the present code `npmin(1)=1` and `npsi(2)=nstmax`.

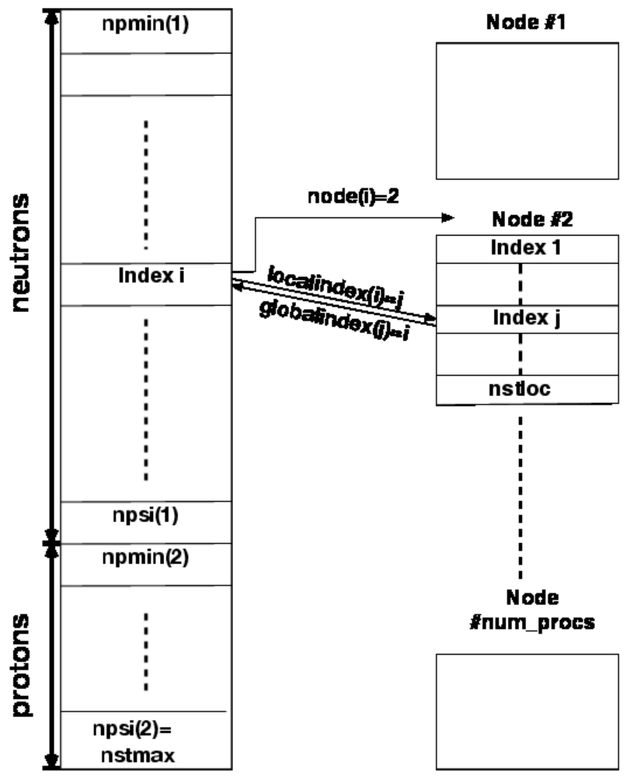

Figure 1: Storage arrangement of the single-particle wave functions. On the left the case for single-processor or `OpenMP` is shown, which for the case of distributed memory under `MPI` is mapped to the individual processors as shown on the right. Note that each node will have its own values of `nstloc` and `globalindex`.

If the code is run in parallel (MPI) on several nodes, only `nstloc` single-particle wave functions are stored on a given node, where `nstloc` may vary. Pointers are then defined to indicate the relationship between the local index and that in the global array of wave functions. For details see the section on parallelization; the general layout is given in Fig. 1.

There are a number of arrays containing the physical properties of the wave functions, such as the single-particle energy. The names start with `sp_` and they are defined in module `Levels`. They are not split up in the parallel case, but on each node only the pertinent index positions are used.

2.7.6. Densities and currents

The various densities necessary for constructing the mean field are actually kept in separate arrays and can be output onto data files for later analysis (see subroutine `write_densities`). The dimensioning is `(nx,ny,nz,2)` with the last index referring to isospin for scalar densities, so that `rho(:,:,:,1)` is the neutron density and `rho(:,:,:,2)` the proton density. For vector densities there is an additional index with values 1 to 3 for the Cartesian direction, thus `sdens(nx,ny,nz,3,2)` containing the spin density in each direction for neutrons and protons.

Since it is often not necessary to keep the neutron and proton contributions separate, subroutine `write_densities` has the option of adding them up before output.

2.8. Initialization

A particular strength of the code is its flexible initialization. There are essentially three types of initialization, which can be selected through the input variable `nof`:

1. **Harmonic oscillator:** `nof=0`: this is applicable only to static calculations. The initial wave functions are generated from harmonic oscillator states with initial radii `radinx`, `radiny`, and `radinz` in the three directions. It is advisable to choose the three radii different to avoid being kept in a symmetric configuration for non-spherical nuclei. Note that this is a very simple initialization and has some defects; for example, the initial deformation is controlled more by the occupation of the oscillator states than by the radius parameters. This should eventually be replaced by, e. g., Nilsson wave functions.
   For this case the type of nucleus is determined by the input numbers `nneut` and `nprot` giving the number of neutrons and protons, while `npsi` can be used to add some unoccupied states (this sometimes leads to faster convergence).
2. **Fragment initialization:** `nof>0`: wave functions for a number `nof` of fragments are read in and positioned in the grid at certain positions. The wave functions are read from files produced by the static code with the file names given by the input `filename`, they are positioned at center-of-mass positions `fcent` and given an initial velocity controlled by `fboost`. The code determines the number of wave functions needed from these data files and also checks the agreement of Skyrme force and grid used. This initialization is used, e.g., for nuclear reactions (see Section 2.5.4). For details see the input description in Section 5.9.
   The number of fragments read in is arbitrary, but there are two special cases:
   - for `nof=1` a single fragment is read in. This can be useful for initializing with static wave functions to study collective vibrations in a nucleus using the TDHF mode.
   - for `nof=2` a special initialization can be done where the initial velocities are not given directly but computed from a center-of-mass energy `ecm` and an impact parameter `b`.

3. **User initialization:** a user-supplied routine `user_init` can be employed to set up the wave functions in any desired way. The only condition is that the index ranges etc. are set up correctly and the wave function array `psi` is filled with the proper values. It was found useful, e. g., to use initial Gaussians distributed in various geometric patterns for α-cluster studies.

2.9. Restarting a calculation

Sometimes it is necessary to continue a calculation that was not run to the desired completion because of a machine failure or because the number of iterations or time steps was set too low. In such cases the last wave function file with name `wffile`, which is generated at regular intervals of `mrest` iterations or time steps, can be used to initialize a continuation. The program handles this in a simple fashion: if the logical variable `trestart` is input as `TRUE`, it sets up an initialization with one fragment (read from the initialization file) placed at the origin and with zero velocity. The only other modifications to the regular setup are then to take the initial iteration number and time from that file instead of starting at zero, as well as suppressing some unneeded initialization steps.

Restarting works for both static and dynamic calculations. To continue a calculation that was stopped because the desired number of iterations or time steps was reached, a new limit for these should be provided in the input.

This flexible restart makes it possible to use a different grid for the continuation in the sense that the grid spacings must agree, but the new grid can be larger than the old one.

2.10. Accuracy considerations

The grid representation and solution methods introduced above depend on several numerical parameters. Their proper choice is crucial for the accuracy and speed of the calculations. In this Section, we want to briefly address the dependence on numerical parameters. An extensive discussion of grid representations and static iteration is found in [56].

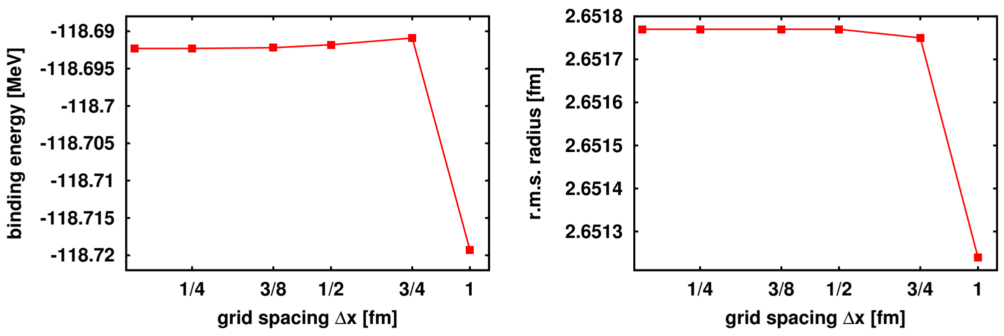

Figure 2: Binding energy (left) and r.m.s. radius (right) of ^{16}O computed for the force SkI3 drawn as functions of grid spacing $\Delta x = \Delta y = \Delta z$. A logarithmic scale us used for Δx. The number of grid points has been chosen to keep the box size constant at $N_x \Delta x = 24$ fm.

Figure 2 shows the sensitivity with respect to the grid spacings Δx, Δy, Δz. The trend is the same for both observables, energy and radius: The results have very high

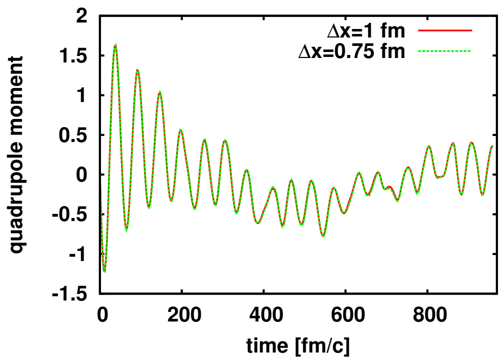

Figure 3: Time evolution of the quadrupole momentum for two different grid spacings $\Delta x = \Delta y = \Delta z$ and constant box size of 24 fm. The test case is ^{16}O excited by an instantaneous boost computed with the force SkI3.

quality and change very little up to $\Delta x = 0.75$ fm. They quickly degrade above that spacing. But even at $\Delta x = 1$ fm, we still find an acceptable quality which suffices for most applications, particularly for large scale explorations. If high accuracy matters, $\Delta x \approx 0.75$ fm should be chosen; not much is gained by going to even finer gridding. This holds for ground states and moderate excitations. High excitations and fast collisions may require a finer mesh. Note that the maximum representable kinetic energy is $E_{\text{kin,max}} = (\hbar^2/2m)(\pi/\Delta x)^2$, which amounts to about 200 MeV for $\Delta x = 1$ fm. The actual energies of interest should stay far below this limit. It is an instructive exercise to study uniform center-of-mass motion at various velocities to explore the limits of a given representation.

The number of grid points $N_x = N_y = N_z$ in the tests of Figure 2 were chosen such that the box size was the same in all cases. The actual choice of N_x depends sensitively on the system, its size and separation energy. As a rule of thumb, the density decreases asymptotically as $\rho \propto \exp\left(-2\sqrt{2m\varepsilon_N}r/\hbar\right)$ where ε_N is the single particle energy of the least bound state. One should aim for at least $\rho < 10^{-8}\,\text{fm}^{-3}$ at the boundaries.

Figure 3 explores the effect of grid spacing for dynamics. Two different spacings are compared for a quadrupole oscillation following an instantaneous quadrupole boost. Practically no difference can be seen for the “safe choice” $\Delta x = 0.75$ fm and the robust choice $\Delta x = 1$ fm. Dynamical applications, oscillations and collisions, are in general less demanding and can be performed very well with $\Delta x = 1$ fm. This is pleasing as dynamical calculations are usually much more costly than purely static ones.

There are two parameters regulating the static iteration according to Eq. (12), the damping energy `e0inv` and the step size `x0dmp`. `e0inv` should correspond to the depth of the binding potential. The overall step size `x0dmp` can be of order of one if `e0inv` is well chosen. Nuclear binding is very similar all over the chart of nuclei. This allows to develop one safe choice for nearly all cases. We recommend `e0inv`$\approx$ 100 MeV together with `x0dmp`$\approx 1/2$, reducing the latter slightly if convergence problems appear. Of course, a few percent in iteration speed might be gained by fine-tuning these parameters for a given case, but this is not worth the effort unless large scale surveys for a given class of nuclei and forces are planned.

The time stepping using the exponential propagator has the two parameters, step

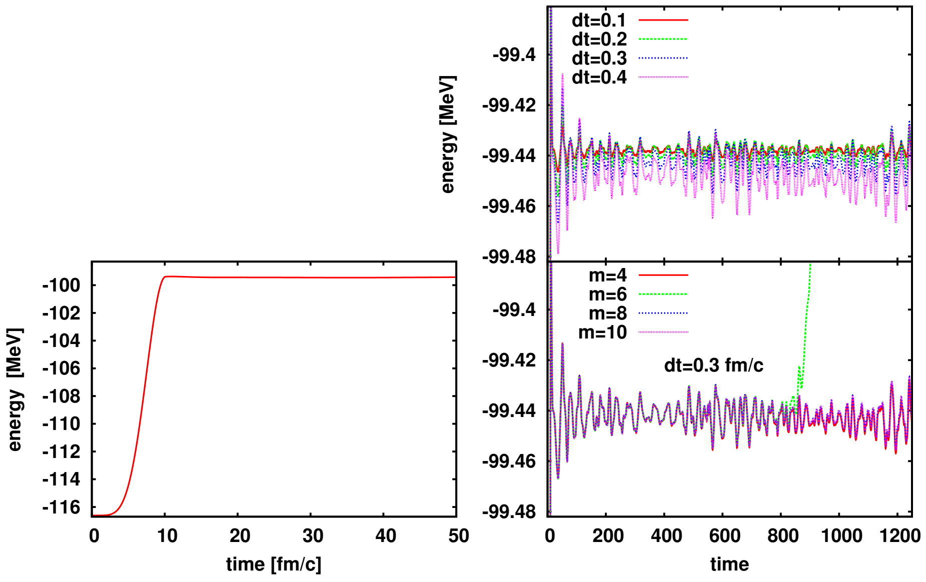

Figure 4: Time evolution of the binding energy of ^{16}O after an excitation by a soft $\cos^2$ pulse of width 20 fm/c computed for the force SV-bas [40] with a grid spacing $\Delta x = 0.75$ fm and box size of 24 fm. Results are shown for different sizes of time step `dt` and different order of Taylor expansion `m` of the exponential evolution. The left panel shows the full evolution from ground state energy to the excited energy. The right panels concentrate on the times after the pulse is over in a narrower energy range relevant for this excitation.

size `dt` and order `mxp`$= m$ of the Taylor expansion (18) of the exponential. Intuitively, one expects that small `dt` and large `mxp` improve the quality of the step. An efficient stepping scheme, however, looks for the largest `dt` and smallest `mxp` which still provide acceptable and stable results. It is hard to give general rules as good working values for the parameters depend on all details of the actual calculation: gridding, nuclei involved, excitation energy, and kind of excitation.

Figure 4 demonstrates the dependence of a typical dynamical evolution on these time-stepping parameters. We consider a time interval up to 1260 fm/c which is a long time for heavy-ion collisions and just sufficient for a spectral analysis of oscillations [60]. The excitation is done by a soft $\sin^2$ pulse of finite extension in time. The energy increases during the initial excitation phase, as can be seen from the left panel in the figure. After the external pulse is over, energy conservation holds, which is nicely seen at plotting resolution in the left panel. Normalization should be conserved at all times. Both conservation laws serve as tests for the time step. Norm is conserved up to at least six digits for all cases and times shown in Figure 4. The energy is more critical. The right panels show the energy in a small window around the final energy after the excitation phase is over. The right lower panel shows a variation of the Taylor order m for fixed time step. The most prominent effect is the sudden turn to catastrophic failure for $m = 6$. In fact, propagation by approximate exponential evolution explodes sooner or later in all cases. The art is to extend the stable interval by a proper choice of the stepping parameters. It is plausible that the cases with $m > 6$ maintain stability longer because the exponential is better approximated. It is surprising that $m = 4$ is also stable over the whole time interval. There seem to be subtle cancellations of error going on. Considering the stable signals, we see very little differences between the cases. One may generally be happy with low m. It is mainly stability demands which could call for larger m. Note that this is not a generic result. Stability for a given test case should be checked once in a while and particularly before launching larger surveys.

The right upper panel in Figure 4 shows results for different `dt` (as we have seen, the m values are not important as long as we achieve stable results). Here we see a clear dependence on the step size. The energies remain constant in the average. But there are energy fluctuations and these depend sensitively on `dt`. Smaller `dt` yields smaller fluctuations. As far as one can read off from the figure, the amplitude of the fluctuations shrink $\propto \texttt{dt}^2$. It depends on the intended analysis to which level of precision the time evolution should be driven. A value of `dt`$\approx$ 0.4 fm/c will be acceptable in most cases because the average trend remains far smaller than the fluctuations. Here also it must be emphasized that this is not a generic number. Forces with lower effective mass (SV-bas has $m^*/m = 0.9$) are more demanding and usually require smaller `dt`. On the other hand, running propagation without the spin-orbit term allows even larger time steps because the spin-orbit potential is the most critical piece in the mean-field Hamiltonian. The mix of $\mathbf{p}$ and $\mathbf{r}$ imposes high demands on the numerical representations. We again strongly recommend running a few tests when switching forces or excitation schemes.

3. Code structure

The code is completely modularized to provide as large a degree of encapsulation as possible in order to ease modification. Most modules read their operating parameters from an associated `NAMELIST` and have their local initialization routines. In addition, a modern style of programming is used that employs a minimum number of local variables and streamlined array calculations that make the code lines very close to the physical equations being solved.

Here we give a brief overview over the modules and their purpose. The source files containing the modules have the same names, but all in lower case, with an extension of `.f90`. The higher-level modules are:

Main program: It calls initialization routines, sets up the initial wave functions using either harmonic oscillator states or reading wave functions of static Hartree-Fock solutions from given input files (module `Fragments`). It then calls either `statichf` from module `Static` or `dynamichf` from module `Dynamic` to run the calculation.

Static: This contains the code for the static iterations `statichf` and the subroutine `sinfo` to generate output of the results.

Dynamic: runs the dynamic calculation in `dynamichf` and generates output in `tinfo`. Also controls the inclusion of an external excitation implemented in module `External`.

Densities: calculates the densities and current densities by summing over the single-particle states.

Meanfield: contains the central physics calculation: the computation of the components of the mean field (subroutine `skyrme`) and the application of the single-particle Hamiltonian to a wave function (subroutine `hpsi`).

Coulomb: calculation of the Coulomb potential.

Energies: calculation of the total energies and its various contributions.

External: calculation of the action of an external potential or initial collective boost of the wave functions.

Pairs: Implementation of the pairing correlations in the BCS approximation.

Moment: calculation of moments and deformation parameters for the bulk density.

Twobody: attempts to divide up the system into two separated nuclei and to calculate their properties and relative motion.

The lower-level supporting modules are:

Params: general parameters used throughout the code.

Forces: defines parameters of the Skyrme force and the pairing interaction and constructs them according to input.

Grids: defines everything associated with the numerical grid and sets it up.

Levels: definition of the single-particle wave functions and elementary operations on them such as derivatives.

Fragments: controls the reading of static wave functions from precomputed data and setting them up in the grid.

Inout: contains the subroutines for I/O of wave functions and densities.

Trivial: defines some very basic operations on wave functions and densities.

Fourier: sets up the transform plans for the `FFTW3` package to calculate Fourier transforms of wave functions and densities.

Parallel: This comes in two versions. The source file `parallel.f90` contains the routines to handle `MPI` message passing, while `sequential.f90` sets up essentially dummy replacements for sequential or `OpenMP` mode.

User: contains a sample user initialization code which can be used as a template for more complicated setups.

4. Parallelization

For both OpenMP [73] and MPI [74] the code can be run in parallel mode. Parallelization for the static mode works in OpenMP but not in MPI: the reason is in the orthogonalization step which is not easily amenable for distributed-memory parallel computation. This should be worked on in the future.

MPI and OpenMP can be used jointly if there are computing nodes with multiple processors.

In both cases parallelization is done over the wave functions. The code applies the time-development operator or the gradient iteration, which use the fixed set of mean-field components, to each wave function, and this can naturally be parallelized. Computing the mean fields and densities by summing up over single-particle wave functions is also easily parallelizable.

The library `FFTW3` [75] itself can also run on multiple processors in parallel. This can be used in addition to OpenMP or MPI, but was found to be helpful only in the sequential version of the code.

4.1. OpenMP

The application of the subroutine `tstep` for propagating one wave function for one time step, and of `add_densities` for adding one wave function's contribution to the mean fields is done in parallel loops. The only complicating factor is that the densities, being accumulated in several subsets, must be kept separate using the `REDUCTION(+)` clause of OpenMP. The summation cannot be done internally in `add_densities`, because for the half time step the wave functions are immediately discarded after adding their contribution to the densities to avoid having to store the full set at half time. Thus only the combined `tstep-add_densities` loop should be parallelized.

The OpenMP program version can be compiled using the appropriate compiler option. A separate `Makefile.openmp` is provided which just contains the `-fopenmp` option for the GNU compiler. The number of parallel threads is not set by the code: the user should set the environment variable `OMP_NUM_THREADS` to the desired number.

The user should also check compatibility of the locally installed tt LAPACK/BLAS libraries with `OpenMP`.

4.2. MPI

In principle MPI uses the same technique as OpenMP, parallelizing over wave functions. In this case, however, each node contains only a fraction of the wave functions. This has several consequences:

1. The time-stepping of the wave functions can be done independently on each node, but requires that the densities are broadcast to all nodes after each half or full time step by summing up partial densities from the nodes in subroutine `collect_densities`.
2. The other calculation that uses wave functions directly is that of the single-particle properties. These are calculated on each node for the wave functions present on that node and then collected using subroutine `collect_sp_properties`.
3. Only one node must be allowed to produce output. This is regulated by choosing node #0 and setting the flag `wflag` to `.TRUE.` on that node.

4. The saving of the wave functions is done in the following way: Node zero writes a header file containing the job information on file `wffile`, then each node writes a separate file `wwfile.001`, `wwfile.002`, and so on up to the number of nodes. This avoids having to collect the wave functions on one node.
5. Using these parallel output files as fragment initialization or restart files is handled so flexibly that they can be read into a different nodal configuration or even a sequential run.

The MPI version needs the appropriate compiler and linker calls for the system used. The sequential or OpenMP versions are obtained simply by linking with `sequential.f90` instead of `parallel.f90`, which replaces the MPI calls with a set of dummy routines and sets up the descriptor arrays for the wave function allocation in a trivial way.

The `Makefile.mpi` shows the procedure; in practice systems differ considerably and the user should look up the compilation commands for his particular system.

5. Program unit descriptions

5.1. Program `Main3D`

This is the main program that organizes the reading of the input and funnels the calculation into the correct subroutines.

It consists of a number of simple steps. They are:

- **Step 1:** initialize `MPI` in case of an `MPI` parallel job. Start reading from standard input beginning with namelist `files` for any changes in the file names.
- **Step 2:** read the definition of the force to be used and set it up (for its parameters see Section 2.2.3).
- **Step 3:** read namelist `main`, which contains the overall controlling parameters. The choice of `imode` is used to set up `tstatic` and `tdynamic` as logical variables. In addition, if this is a restart, the number of fragments is set to one. The properties of this one fragment are then filled in in subroutine `getin_fragments`.
- **Step 4:** read the grid definition (see Section 2.7.2). With the dimensions known, all grid-based arrays (but not the wave functions) can be allocated and the initialization of the `FFTW` system can be done.
- **Step 5:** the appropriate namelist is read for the static or dynamic case, defining some parameters in those modules.
- **Step 6:** determine wave function numbers. The way this is done depends on the choice of `nof`. For positive `nof`, the fragment files are consulted to find out the properties of each and add up the numbers. For `nof=0` the numbers are taken from namelist `static`, which was read before. If `nof<0`, they are also given in namelist `static`, but the wave functions will be replaced by user-calculated ones.
- **Step 7:** now that the numbers are known, the wave function distribution over nodes is computed or set to trivial for a sequential calculation, and the the arrays related with wave functions are allocated.
- **Step 8:** the initial values of the wave functions are calculated. For `nof>0` the wave functions are read from the fragment files and inserted into the proper positions (see Sections 2.5.4 and 5.9). For `nof=0` the routine `harmosc` in module `Static` is called to calculate harmonic-oscillator wave function, and for `nof<0` the routine `init_user` does an arbitrary user initialization.
- **Step 9:** the mean-field arrays are zeroed and the Coulomb solver is initialized.
- **Step 10:** the calculation branches into either static or dynamic mode.

  In the static calculation, `init_static` just prints some information and sets up damping (see Section 2.4.2) and `statichf` does the real calculation.

  In the dynamic case, the subroutine `dynamichf` does all the work.
- **Step 11:** finally the `MPI` system is terminated.

5.2. Module `Coulomb`

5.2.1. Purpose

This module offers the subroutine `poisson`, which calculates the Coulomb potential from the proton density. The two boundary conditions allowed are distinguished by the value of the variable `periodic`. If it is true, the mesh is assumed to be periodic in all three directions and the Poisson equation is solved using the $1/k^2$ Green's function in momentum space and assuming a homogeneous negative background annulling the total charge (jellium approximation). Otherwise the boundary condition is for an isolated charge distribution. In this case the potential is calculated by the method of doubling the grid in all directions with zero density, folding with the $1/r$-potential in momentum space and then restricting to the physical region [76].

5.2.2. Details on the Coulomb solver for an isolated system

The Coulomb solver follows the ideas of [76]. We summarize here the way it is realized in the code without detailed proofs.

Two grids are used. We will call them here $\mathcal{G}_1$ and $\mathcal{G}_2$. The grid $\mathcal{G}_1$ is our standard working grid as given in eq. (25). We repeat here the essentials

$$\begin{aligned}
&\mathcal{G}_1 : \nu_x = 1,...,\mathtt{nx}\ ,\ \nu_y = 1,...,\mathtt{ny}\ ,\ \nu_z = 1,...,\mathtt{nz}\ ;\\
&\quad\text{coordinate spacing: } \mathtt{dx},\mathtt{dy},\mathtt{dz};\\
&\quad\text{momentum spacing: } \mathtt{dk}_x = \frac{2\pi}{\mathtt{nx}\cdot\mathtt{dx}}, \mathtt{dk}_y = \frac{2\pi}{\mathtt{ny}\cdot\mathtt{dy}}, \mathtt{dk}_z = \frac{2\pi}{\mathtt{nz}\cdot\mathtt{dz}};
\end{aligned}$$

The second grid amounts to a doubled box in each direction, thus

$$\begin{aligned}
&\mathcal{G}_2 : \nu_x = 1,...,2\mathtt{nx}\ ,\ \nu_y = 1,...,2\mathtt{ny}\ ,\ \nu_z = 1,...,2\mathtt{nz}\ ;\\
&\quad x_{\nu_x} = (\nu_x - \mathtt{nx})\mathtt{dx}\ ,\ y_{\nu_y} = (\nu_y - \mathtt{ny})\mathtt{dy}\ ,\ z_{\nu_z} = (\nu_z - \mathtt{nz})\mathtt{dz}\ ;\\
&\quad\text{momentum spacing: } \mathtt{dk}_x = \frac{2\pi}{2\mathtt{nx}\cdot\mathtt{dx}}, \mathtt{dk}_y = \frac{2\pi}{2\mathtt{ny}\cdot\mathtt{dy}}, \mathtt{dk}_z = \frac{2\pi}{2\mathtt{nz}\cdot\mathtt{dz}};
\end{aligned}$$

The momenta k_{n_i} in $\mathcal{G}_2$ are arranged around $k = 0$ similar as in $\mathcal{G}_1$, but with half spacing $\mathtt{dk}_i$. For simplicity, we will use compact notation $\mathbf{r}_{\boldsymbol{\nu}} = \left(x_{\nu_x}, y_{\nu_y}, z_{\nu_z}\right)$ and similarly for $\mathbf{k}_{\mathbf{n}}$.

First, the Coulomb solver has to be initialized by defining the appropriate Greens function which is done on $\mathcal{G}_2$. The two necessary steps are:

1. Prepare the Greens function in r space as

$$G(\mathbf{r}_{\boldsymbol{\nu}}) = \frac{e^2}{|\mathbf{r}_{\boldsymbol{\nu}}|} \quad .$$

   Special consideration has to be given to the singularity of the $1/r$ potential. In the discretized version, the value at $\vec{r} = 0$ should be replaced by the average of $1/r$ over a cuboid of size $\Delta x \times \Delta y \times \Delta z$. For equal spacing in all three directions this would lead to a value of $2.38/\Delta x$. Practical experimentation, however, showed that the underlying assumption of a point charge can be improved by some smeared-out density for nuclear applications; a value of $2.84/\Delta x$ was found to be optimal. We use the expression

$$G(0) = \frac{2.84\sqrt{3}}{\sqrt{\mathtt{dx}^2 + \mathtt{dy}^2 + \mathtt{dz}^2}} \quad . \tag{30}$$

2. Prepare the Greens function in k-space by 3D fast Fourier transformation (FFT) on the double grid $\mathcal{G}_2$. $\tilde{G}(\mathbf{k}_{\boldsymbol{\mu}}) = \mathtt{FFT}\{G(\mathbf{r}_{\boldsymbol{\nu}})\}$. The array $\tilde{G}(\mathbf{k}_{\boldsymbol{\mu}})$ is stored for further continued use.

Once properly prepared, the practical steps for computing the Coulomb field $U_{\text{Coul}}(\mathbf{r}_{\boldsymbol{\nu}})$ for a density $\rho(\mathbf{r}_{\boldsymbol{\nu}})$ given on $\mathcal{G}_1$ are

1. Extend $\rho_{\boldsymbol{\nu}}$ from $\mathcal{G}_1$ to $\mathcal{G}_2$ by zero padding:

$$\rho_2(\mathbf{r}_{\boldsymbol{\nu}}) = \begin{cases} \rho(\mathbf{r}_{\boldsymbol{\nu}}) & \text{if} \quad 1 \le \nu_i \le \mathtt{n}i \text{ for } i \in \{x, y, z\} \\ 0 & \text{else} \end{cases}$$

2. Fourier transform the density in $\mathcal{G}_2$:

$$\rho_2(\mathbf{r}_{\boldsymbol{\nu}}) \quad \longrightarrow \quad \tilde{\rho}_2(\mathbf{k}_{\boldsymbol{\mu}}) = \mathtt{FFT}\{\rho_2(\mathbf{r}_{\boldsymbol{\nu}})\}$$

3. Compute solution in k-space by multiplication with the formerly prepared Greens function

$$\tilde{U}_2(\mathbf{k}_{\boldsymbol{\mu}}) = \tilde{G}_2(\mathbf{k}_{\boldsymbol{\mu}})\tilde{\rho}_2(\mathbf{k}_{\boldsymbol{\mu}})$$

4. Compute solution in r-space by Fourier back transformation in $\mathcal{G}_2$:

$$U_2(\mathbf{r}_{\boldsymbol{\nu}}) = \mathtt{FFT}^{-1}\{\tilde{U}_2(\mathbf{k}_{\boldsymbol{\mu}})\}$$

5. Map to standard grid $\mathcal{G}_1$:

$$U_{\text{Coul}}(\mathbf{r}_{\boldsymbol{\nu}}) = U_2(\mathbf{r}_{\boldsymbol{\nu}}) \quad \text{for} \quad \boldsymbol{\nu} \in \mathcal{G}_1$$

5.2.3. Module variables

`nx2, ny2, nz2`: (INTEGER,PRIVATE) dimensions of the grid on which the Fourier transform is calculated. For the periodic case they are identical to the regular dimensions, for the isolated case they are doubled.

`coulplan1, coulplan2`: (INTEGER,PRIVATE) plans for `FFTW` complex forward and reverse transforms with array dimensions depending on the boundary condition.

`wcoul`: the Coulomb potential as a three-dimensional array. Units: MeV.

`q`: (COMPLEX,PRIVATE) array for the complex Green's function (isolated) or array of $1/r$ values. Its dimension also depends on the boundary condition.

5.2.4. Subroutine `poisson`

This subroutine solves the Poisson equation by the Fourier method. For the periodic case, this means to just Fourier transform, multiply by $1/k^2$, and the transform back. Note that the coefficient for momentum zero must be zero also: this means that the total charge in the box vanishes, corresponding to the jellium approximation.

In the non-periodic case we use use the trick of doubling the box size in every direction and then folding with $1/r$ my multiplying the Fourier transforms. The result in the physical box is then the correct solution for the boundary condition of zero potential at infinity. The key to success is not to use simply $1/k^2$ for the Fourier transform of $1/r$, but to compute it on the actual grid the same way as the densities and potentials are transformed [76].

5.2.5. Subroutine `coulinit`

This subroutine does the necessary initialization. It calculates the dimension for the Fourier transform depending on the boundary condition, allocates the necessary arrays and sets up the `FFTW` plans.

Then it composes the array `q`. For the periodic case this contains the values of the Green's function $1/k^2$, with zero at the origin, while for the isolated case it first calculates the inverse shortest distance $1/r$ from the origin with indices (1,1,1), replaces the value at the origin according to Eq. (30) and then Fourier-transforms this to use it for folding the density with the coordinate-space Green's function.

5.2.6. Subroutine `initiq`

This subroutine calculates the contributions of each Cartesian index to r^2 and k^{-2}. This depends on the boundary condition. For the periodic case, the values of the momenta are given by

$$k_i = \frac{2\pi}{n\Delta x}\big(0,\ 1,\ 2,\ \ldots \frac{n}{2}-1,\ -\frac{n}{2},\ -\frac{n}{2}+1,\ \ldots -1\big).$$

For an isolated distribution the shortest distances to the point with index 1 are calculated (periodicity used):

$$d_i = d\big(0,\ 1,\ 2,\ \ldots \frac{n}{2}-1,\ -\frac{n}{2}, -\frac{n}{2}+1,\ \ \ldots 1\big).$$

The input is the dimension along the coordinate direction given, the output is the one-dimensional array `iq` containing these values **squared**.

5.3. Module `Densities`

This module has two purposes: it defines and allocates the densities and currents making up the mean field according to Sect. 2.2.2, and also contains the subroutine `add_density` which accumulates the basic densities over the single-particle wave functions. Subroutine `skyrme` in module `Meanfield` then uses these densities to build up the components of the single-particle Hamiltonian.

5.3.1. Module variables

- **Scalar densities:** These are dimensioned `(nx,ny,nz,2)`, where the last index is 1 for neutrons and 2 for protons.

  `rho`: the density, separately for each isospin (in fm^{-3}). The definition is:

  $$\rho_q(\vec{r}) = \sum_{k\in q} w_k^2 \sum_s |\phi_k(\vec{r},s)|^2, \qquad q = n,p$$

  `tau`: the kinetic energy density, also separately for each isospin. It is defined as the sum of the spin contributions and all particles of the given isospin

  $$\tau_q(\vec{r}) = \sum_{k\in q} w_k^2 \sum_s |\nabla\phi_k(\vec{r},s)|^2, \qquad q = n,p$$

  Note that it does not include the factor $\hbar^2/2m$. Units: fm^{-5}.

- **Vector densities** These are dimensioned `(nx,ny,nz,3,2)`, where the last index is 1 for neutrons and 2 for protons, and the next-to-last stands for the Cartesian direction.

  `sdens`: the spin density. It is defined as

  $$\vec{\sigma}_q(\vec{r}) = \sum_{\alpha\in q} w_\alpha^2 \sum_{ss'} \psi_\alpha^*(\vec{r},s)\,\sigma_{ss'}\,\psi_\alpha(\vec{r},s').$$

  Note that it does not include the factor $\hbar/2$. Units: fm^{-3}.

  `current`: this is the total probability current density, defined in the familiar way as

  $$\vec{j}_q(\vec{r}) = \frac{1}{2\mathrm{i}} \sum_{\alpha\in q} w_\alpha^2 \sum_s \left(\psi_\alpha^*(\vec{r},s)\nabla\psi_\alpha(\vec{r},s) - \psi_\alpha(\vec{r},s)\nabla\psi_\alpha^*(\vec{r},s)\right).$$

  Note that the factor $\frac{\hbar}{m}$ is not included. Its units are therefore fm^{-4}.

  `sodens`: the spin-orbit density, defined as

  $$\vec{J}_q(\vec{r}) = \frac{1}{\mathrm{i}} \sum_{\alpha\in q} w_\alpha^2 \sum_{ss'} \left(\psi_\alpha^*(\vec{r},s)\nabla\times\sigma_{ss'}\psi_\alpha(\vec{r},s')\right).$$

  Its units are also fm^{-4}.

5.3.2. Subroutine `alloc_densities`

This is simply a short routine to allocate all the arrays defined in this module.

5.3.3. Subroutine `add_density`

This subroutine is given a single-particle wave function `psin` with its isospin index `iq` and occupation w_α^2 =`weight` and adds its contribution to the density arrays.

The reason for not including the loop over states in the subroutine is that in the dynamic code, the contribution of a new single-particle wave function (calculated by `tstep`) to the densities is added without saving that wave function, eliminating the requirement for a second huge wave-function array.

It may seem strange that `add_density` has the densities themselves as parameters, which are readily available in the module. The reason for this is `OPENMP` parallelization. The loop over wave functions is done in parallel under `OPENMP`. Since any of the parallel tasks must add up the contributions of its assigned wave functions, each task must have a copy of the densities to work on; otherwise they would try to update the same density at the same time. The separate copies are then combined using the `OPENMP REDUCE(+)` directive.

The local copies of the densities passed as arrays are denoted with the prefixed letter "l" for *local*; they are `lrho`, `ltau`, `lcurrent`, `lsdens`, and `lsodens`.

If the weight is zero, there is nothing to do and the subroutine returns immediately. Otherwise, the contributions not involving derivatives are first computed and added to the affected densities, i. e., number and spin density.

After this the derivative terms are evaluated by computing each Cartesian direction separately. In all three cases the derivative is evaluated first and put into `ps1`, after which the contributions are added straightforwardly. They involve the wave function itself, the derivative, and for the spin-orbit density also a Pauli matrix, so that different spin projections have to be combined properly.

The complex products always in the end evaluate to something real and the expressions are simplified to take this into account. For example, the following transformation is done:

$$\begin{aligned}\frac{1}{2\mathrm{i}}(\psi^*\nabla\psi - \psi\nabla\psi^*) &= \frac{1}{2\mathrm{i}}\left(\psi^*\nabla\psi - (\psi^*\nabla\psi)^*\right)\\ &= \frac{1}{2\mathrm{i}}\left(2\mathrm{i}\Im(\psi^*\nabla\psi)\right)\\ &\rightarrow \mathtt{AIMAG(CONJG(psin)*psi1)}\end{aligned}$$

and similarly for the other expressions.

The efficiency of this relies on the FORTRAN compiler recognizing that only the imaginary part of the complex product is needed and not computing the real part at all. This seems to be the case with all present compilers.

5.4. Module `Dynamic`

This module contains the routines needed for time propagation of the system (see Section 2.5.2). All the logic needed for this case is concentrated here; all other modules except for `External` are equally used in the static calculation.

5.4.1. Module variables

- `nt`: the number of the final time step to be calculated. In case of a restart this is smaller than the total number of time steps.
- `dt`: the physical time increment in units of fm/c.
- `mxpact`: the number of terms to be taken in the expansion of the potential according to Eq. (18).
- `rsep`: the final separation distance. The calculation is stopped if there has been a reseparation into two fragments and their distance exceeds `rsep`.
- `texternal`: this logical variable indicates that an external field is present. See module `External`.
- `text_timedep`: this logical variable indicates that the external field is time-dependent and does not describe an instantaneous boost.
- `esf`: this is the energy shift for the call to `hpsi`. Since it is not used in the dynamics part of the code, it is here set to the constant value of zero.

5.4.2. Subroutine `getin`*_dynamic*

This is a relatively simple routine that reads the input for namelist `dynamic` and prints it on standard output. If `texternal` is true, it also calls `getin_external` to read the external field parameters.

5.4.3. Subroutine `dynamichf`

This subroutine performs the main time-integration algorithm starting at time step 0 and then iterating the desired number of steps. Its building blocks are:

- **Step 1: preparation phase**: this phase consists of several substeps.
  1. If this is not a restart, the time and iteration number are zeroed (for a restart only the physical time is taken from the `wffile`). The wave functions are saved in the `wffile`, to save setup time in case the calculation has to be restarted from this initial point.
  2. The instantaneous external boost (10) is applied using `extboost`. If this subroutine has applied a boost, it sets `text_timedep` to `.FALSE.` so no further calls to external-field routines are made (except for `print_extfield`).
  3. The protocol files `*.res` are initialized with their header lines.
  4. The densities and current are calculated in a loop over wave function by calling `add_densities`. They are first set to zero and then accumulated in a loop over the set on the local node, followed by collecting them over all nodes.

5. The mean field and (if `texternal` is true) the external field are calculated for time zero using routines `skyrme` and `extfld`.
6. Then `tinfo` is called to calculate and print the single-particle quantities and the total energies at time 0 or iteration 0.
7. Finally preparations are made for the time-stepping loop: the starting index is set to iter+1: this is either after the end of a previous job in the case of a restart, or just one for a new calculation. The physical time is either 0 or the time taken from a restart file.

- **Step 2: predictor time step**: the loop over the iteration index `iter` is started. Then the densities and mean-field components are estimated, i. e., in effect the Hamiltonian $\hat{h}(t + \frac{1}{2}\Delta t)$ required by the numerical method (see Eq. (16) and Section 2.5.2). This is done by evolving the wave functions for a full `dt` using the old Hamiltonian and averaging the densities between old and new ones to obtain the mid-time Hamiltonian for propagation. In detail the procedure is as follows:

  1. The densities are not set to zero in order to allow adding the contributions of the wave functions at the end of the time step.
  2. In the MPI version, the densities are divided by the number of nodes. In this way, adding up contributions from all nodes, the densities from the beginning of the time step will be included correctly.
  3. Subroutine `tstep` is used to propagate the wave functions to the end of the time step. Note that truncation in the exponential happens at `mxpact/2`, since the accuracy need not be as high as in the full step. For each wave function its contribution is added to the densities. The wave functions themselves do not need to be saved as they are not used for anything else.
  4. After the loop, contributions from all the nodes are added up for the MPI case using subroutine `collect_densities`.
  5. The densities are multiplied by one half to form the average of the values at t and $t + \Delta t$.
  6. These average densities are then used to calculate the mean field at half time using subroutine `skyrme`; also the external field is obtained for the half time using `extfld`.

- **Step 3: full time step**: Now that the single-particle Hamiltonian has been estimated for the middle of the time step, the propagation can be carried out to the end of the time step. This is quite analogous to the half step with only three crucial differences:

  1. The densities are reset to zero before the wave function loop, so the densities summed up are the purely the densities at the end of the time step,
  2. the series expansion in `tstep` now uses the full `mxpact` terms, and
  3. the new wave functions are copied back into `psi` to be available for the next time step.

- **Step 4: Center-of-mass correction** If a center-of-mass correction is desired by the user by setting `mrescm/=0`, subroutine `resetcm` is called every `mrescm`'th time step to reset the center-of-mass velocity to zero.

- **Step 5: generating some output At this point the time is advanced by `dt` because the physical time is now the end of the time step, and this must be printed out correctly by the following output routines.** `tinfo` is called to calculate single-particle properties, total energies, and so on.

- **Step 6: finishing up the time step**: `tinfo` is called to output the calculated data, then `skyrme` and `extfld` calculate the mean field and the external field, respectively, for the end of the time step, after which the wave functions are written onto `wffile` depending on `mrest`.

This ends the time loop and subroutine `dynamichf` itself.

5.4.4. Subroutine `tstep`

In this subroutine one wave function given as the argument `psout` is stepped forward in time by the interval `dt`. The method used is the expansion of the exponential time-development operator according to Eq. (18), cut off at the power of `mxp`, which in practice is usually around 6. Suppressing the argument of $\hat{h}$ for brevity, we can write

$$\hat{U}(t, t+\Delta t)\,\phi \approx \sum_{n=0}^{m} \phi^{(n)} \tag{31}$$

with

$$\phi^{(0)} = \phi, \qquad \phi^{(k+1)} = \frac{-\mathrm{i}\,\Delta t}{\hbar c k}\,\hat{h}\,\phi^{(k)}, \quad k = 0, \ldots, m-1. \tag{32}$$

Thus the application of the polynomial to a single-particle wave function can be evaluated simply in a loop applying $\hat{h}\phi_k$ repeatedly and accumulating the results in wave function `psout`.

The argument `iq` is only necessary because `hpsi` needs information about the isospin of the wave function.

5.4.5. Subroutine `tinfo`

This subroutine is used to output various pieces of information relevant especially to the dynamic mode of the code. It is called at the end of the full time step and consists of the following steps:

- **Step 1: initialization**: the flag `printnow` is calculated to keep track of whether this is the proper time step for a full printout (determined by `mprint`).

- **Step 2: twobody analysis** the twobody analysis is performed, but only if the calculation started as a twobody scenario.

- **Step 3: moments**: the moments of the distribution are calculated using subroutine `moments`. This includes total mass, momenta, and angular momenta. They are printed out if indicated by `printnow`, both on the large output and in the specialized files `dipolesfile`, `momentafile`, and `spinfile`. If there is an external field, the routine `print_extfield` is called to print the current expectation value of the external field.

  Note that the moments need to be calculated every time step, because some of the logic may depend on them, especially the twobody-analysis,

which needs the correct c.m., for example and is calculated at every time step and on every node.

- **Step 4: single-particle quantities**: the single-particle energies are calculated straightforwardly as expectation values of the Hamiltonian. The routine `sp_properties` is then called to obtain the other single-particle properties like angular momenta. They are communicated between the processors. The angular momenta are written to `spinfile`.

- **Step 5: total energies**: The integrated energy `ehfint` and its contributions are calculated in `integ_energy`. The subroutine `sum_energy` is called to calculate the three-body energy and the single-particle based total energy `ehf`. The collective kinetic energy `ecoll` is computed directly here, because it is needed only in the dynamic calculations and only for output. It is defined as

$$E_{\text{coll}} = \frac{\hbar^2}{2m} \int \mathrm{d}^3 r \frac{\vec{\jmath}^{\,2}}{\rho} \tag{33}$$

  The energies are protocolled in `energiesfile` and on standard output.

- **Step 6: density output**: at intervals of `mprint` or in the first time step the density printer plot is generated using `plot_densities` and the binary densities are written onto `*.tdd` files using `write_densities`.

- **Step 7: other output**: in the proper `mprint` interval the two-body analysis results, the single-particle state information, and the moments are printed on standard output, using also the routines `twobody_print` and `moment_print`.

  It is important to note that when `tinfo` is called before the time step iteration starts, the two-body analysis cannot work because the fragment centers of mass from the previous time step are either not known yet (restart) or identical to the present ones (initialization). The subroutine `twobody_case` is still called to find the fragment properties, but the derived kinetic energy etc. will be incorrect. Therefore the call to `twobody_print` is suppressed in this case. The logical variable `initialcall` is used to recognize this case.

- **Step 8: check for final separation**: for the twobody case it is checked whether the separation found between the two fragments is larger than the input quantity `rsep` with positive time derivative `rdot` of the separation distance, in which case the program is terminated with an appropriate message.

5.4.6. Subroutine `resetcm`

This subroutine resets the center-of-mass velocity to zero. The velocity, or rather the corresponding wave vector, is calculated from the current density using

$$\rho(\vec{r})\vec{k} = \frac{1}{2\mathrm{i}} \sum_{\alpha\in q} w_\alpha^2 \sum_s \left(\psi_\alpha^*(\vec{r},s)\nabla\psi_\alpha(\vec{r},s) - \psi_\alpha(\vec{r},s)\nabla\psi_\alpha^*(\vec{r},s)\right).$$

The wave functions are then multiplied by a common plane-wave phase factor $\exp(-\mathrm{i}\vec{k}\cdot\vec{r})$ to give a counter boost.

5.5. Module `Energies`

This module computes the total energy and the various contributions to it in two ways. The first method evaluates the density functional, Eq. (5a), by direct integration to compute the *integrated energy* `ehfint`. The second method uses the sum of single-particle energies plus the rearrangement energies, $E_{3,\mathrm{corr}}$ for the density dependent part and $E_{C,\mathrm{corr}}$ for Coulomb exchange as explained in Section 2.6.2.

The two ways of calculating the energy are assigned to the subroutines `integ_energy`, which also calculates the rearrangement energies, and `sum_energy`. Note that since `integ_energy` is always called briefly before `sum_energy`, the rearrangement energies are correctly available.

In addition subroutine `sum_energies` also calculates the summed spin, orbital, and total angular momenta.

5.5.1. Module variables

All the variables in this module are given in MeV unless otherwise noted.

- `ehft` denotes the total kinetic energy of Eq. (5b).
- `ehf0` corresponds to the b_0 and b'_0-dependent terms of Eq. (5c).
- `ehf1`: the contribution depending on b_1 and b'_1 of Eq. (5d).
- `ehf2` accounts for the b_2, b'_2-dependent contributions of Eq. (5e).
- `ehf3` is the b_3, b'_3-dependent part of Eq. (5f) which models density dependence.
- `ehfls` is the spin-orbit energy (5g).
- `ehfc` is the Coulomb energy of Eq. (5h).
- `ecorc` is the rearrangement correction (23c) to Coulomb exchange in the Slater approximation.
- `e3corr` is the rearrangement energy (23b) to the density-dependent part of the Skyrme energy.
- `orbital`: the three components of the total orbital angular momentum in units of $\hbar$.
- `spin`: the three components of the total spin in units of $\hbar$.
- `total_angmom`: the three components of the total angular momentum in units of $\hbar$.

5.5.2. Subroutine `integ_energy`

The purpose of this subroutine is to calculate the integrated energy of Eq. (5a). This is implemented pretty straightforwardly using the integrals defined in Section 2.2.3. The only programming technique worth noting is that intermediate variables such as `rhot` for the total density are used to avoid repeating the lengthy index lists. Compilers will eliminate these by optimization.

In principle the integration loops in the subroutine could be combined, but some space is saved by using the array `worka` for different purposes in different loops.

The calculation proceeds in the following steps:

- **Step 1**: the Laplacian of the densities is calculated in `worka`, then the integrals of Eqs. (5c,5e, and 5f) are performed. After the loop the result for `ehf3` is also used to calculate `e3corr` of Eq. (23b).
- **Step 2**: the integral of Eq. (5d) is evaluated using `worka` for the $\vec{\jmath}_q^{\,2}$ term.
- **Step 3**: the spin-orbit contribution of Eq. (5g) is calculated using `worka` as storage for $\nabla \cdot \vec{J}_q$.
- **Step 4**: the Coulomb energy is evaluated from Eq. (5h) with the Slater correction taken into account if the force's `ex` is nonzero. At the same time the Coulomb correction for the summed energy is calculated according to Eq. (23c) and stored in `ecorc`. It will be used in the subroutine `sum_energy`.
- **Step 5**: the kinetic energy is integrated for Eq. (5b). Note that only at this point the correct prefactor $\hbar^2/2m$ is added; the use of `tau` in other expressions assumes its absence.
- **Step 6**: Finally all terms are added to produce the total energy, Eq. (5a), from which the pairing energies are subtracted.

5.5.3. Subroutine `sum_energy`

This subroutine mainly computes the Koopman sum of Eq. (22), but also sums up a number of other single-particle properties. For systematics, the latter should be done in a different place, but at present is left here.

The summation of the total energy uses Eq. (13c) to compute

$$\sum_k (\epsilon_k - \tfrac{1}{2} v_k) = \tfrac{1}{2} \sum_k (2t_k + v_k) = \tfrac{1}{2} \sum_k (t_k + \epsilon_k).$$

The last sum is calculated, the rearrangement corrections are added and the pairing energies subtracted.

The subroutine then sums up the single-particle energy fluctuation `sp_efluct1` and `sp_efluct2`, dividing them by the nucleon number. Finally the orbital and spin angular momentum components are summed to form the total ones.

5.6. Module `External`

This module allows the coupling of the nucleonic wave functions to an external field. As described in Sect. 2.3, this can be done either by adding a time-dependent external (i. e., not self-consistent) potential to the single-particle Hamiltonian, or by giving an initial "boost" to each wave function.. Since this is very easy to modify and will probably have to be adjusted for most applications, the present version just contains a sample for a quadrupole coupling. The logic for different time-dependence assumptions and the isospin are however, fully functional and should be useful in many cases.

5.6.1. Module variables

Most of these just describe the behavior of the external field. They are all declared as private, so that the user modifying this code can be sure that the internals are not used anywhere else. The variables are:

- `amplq0`: a strength parameter for the perturbation. As explained in Sect- 2.3, its numerical magnitude is usually not important by itself, but varying it allows studying the effects of different strengths of the excitation.
- `textfield_periodic`: if this is set to `true`, the external field is made periodic according to Eq. (9c), otherwise a damping factor is used as defined in Eq. (9b).
- `radext`,`widext`: parameters r_0 and Δr for the cutoff of the field according to Eq. (9b). Dimension: fm.
- `isoext`: isospin behavior of the external field: `isoext=0` denotes the same action on protons and neutrons, `isoext=1` that with opposing signs.
- `ipulse`: type of pulse. `ipulse=0` denotes the initial boost configuration, `ipulse=1` a Gaussian pulse according to Eq. (9d), and `ipulse=2` a cosine squared behavior as defined in Eq. (9e).
- `omega`, `tau0`, and `taut`: these correspond to the parameters ω, τ_0, and $\Delta\tau$ in Eqs. (9d) and (9e).
- `extfield`: this is the time-independent field generated according to the parameters `amplq0`, `textfield_periodic`, and depending on the latter, possibly `radext` and `widext`. It is used either to calculate the initial boost or is added to the mean field multiplied with the time-dependent factor.

5.6.2. Subroutine `getin_external`

This is quite straightforward. It reads in all the parameters of the external field and immediately does some consistency checks. The relative strength for the neutron and proton fields is set equal for `isoext=0` but reduced by the corresponding number of particles in the `isoext=1`. This avoids a shift of the center of mass in this case.

Then the array `extfield` is allocated and the time-independent spatial potential calculated for both isospin cases and in either the periodic or damped versions, depending on the value of `textfield_periodic`.This is just an illustrative sample field of type Q_{zz}; in a real calculation there is no need to provide both versions.

5.6.3. Subroutine `extfld`

This is again a very straightforward routine. It calculates the time-dependent prefactor `time_factor` depending on the parameters, and adds the time-independent field `extfield` multiplied by this factor to `upot`. The physical time is here given as an argument, because it needs to be evaluated at both full and half time steps.

5.6.4. Subroutine `extboost`

This performs the initial boost according to Eq. (10) on all single-particle wave functions. Note that it is always called from `dynamichf` and checks itself whether `ipulse` is zero. This enables `ipulse` to also be made a private variable. Its argument is used to communicate whether it has actually done anything: if it has applied a boost, it sets its argument to `.FALSE.` so that in `dynamichf` the variable `text_timedep` makes it possible to distinguish this case..

5.6.5. Subroutine `print_extfield`

This subroutine calculates the expectation of the coupling energy to the external field,

$$\sum_q \int \mathrm{d}^3r\, \rho_q(\vec{r}) F_q(\vec{r})$$

and prints one line containing the present time and this value onto the file `extfieldfile`.

5.7. Module `Forces`

Module `Forces` describes the interactions used in the code. The idea is to produce a library of Skyrme forces that can be called up simply by name, but for exploratory purposes a force can also be input using individual parameter values. A force is usually fitted together with a prescription for the pairing and center-of-mass correction, so that these properties are here defined as part of the force.

5.7.1. Module variables and types

To deal with forces and pairing as a unit, derived types are used:

- `Pairing`: This contains the following parameters for pairing:

  `v0prot` : the strength of pairing for protons in MeV.

  `v0neut` : the strength of pairing for neutrons in MeV.

  `rho0pr` : the density parameter for the density-dependent delta pairing (see description for module `Pairs`).

- `Force`: the description of a Skyrme force. It contains the following values:

  name : the name of the force used to identify it.

  `ex` : some forces are fitted excluding the Coulomb exchange term. For `ex=1` it is included (this is the normal case), for `ex=0` not.

  `zpe` : index for the treatment of the center-of-mass correction (see end of Section 2.2.3).

  `h2m` value of $\frac{\hbar^2}{2m}$, separately for neutrons and protons.

  `t0, t1, t2, t3, t4` : Skyrme parameters t_0, t_1, t_2, t_3, and t_4. Defined in the usual way.

  `x0, x1, x2, x3` Skyrme exchange parameters x_0, x_1, x_2, and x_3.

  `b4p` spin-orbit modification parameter b_4' introduced in the "SkI" series of forces (see Sect. xx).

  `power` : exponent in the nonlinear (originally three-body) term.

  `vdi` : parameter set for the volume-delta pairing case.

  `dddi` : parameter set for the density-dependent delta pairing case.

The variables defined in the module are

- `ipair`: selects one of several pairing modes. For historical reasons the values are 0: no pairing, 5: VDI pairing, and 6: DDDI pairing. In the input the symbolic names are used so these numerical values are hidden to the user. For details see the input description and module `Pairs`.

- `f`: this contains parameters for the Skyrme force actually used in the present calculation, packed into the derived-type `Force`.

- `p` : the pairing parameters used in the present calculation. This is separate from the force itself: the force definition usually contains suggestions for the associated pairing, but this often overridden, e.g, by turning off pairing.

- `h2ma`: the average of the two `h2m` values for protons and neutrons.
- `nucleon_mass`: the mass of the nucleon (average of neutron and Proton) in MeV calculated from `h2ma` and `hbc`.
- The `b` coefficients: these are the coefficients actually used for the mean-field and single-particle Hamiltonian calculations in `skyrme` and `integ_energy`. Note that only `b4p` is also included in the Skyrme-force definition; the others are derived from the `t` coefficients.

The predefined Skyrme forces are contained in `forces.data`, which contains an array `pforce` of `TYPE(Force)` data. This is a bit unreliable since the Fortran standard restricts the length of statements; it should be replaced by reading from a data file in case this limit causes problems. The present version is, however, still preferred as a data file would have to be replicated in every application directory (or an absolute path would have to be defined in the `OPEN` statement).

5.7.2. Subroutine `read_force`

The purpose of the subroutine is to read the force and pairing definitions. The `NAMELIST` `force` contains all the defining values for a Skyrme force (but now as individual variables, not in a derived type) plus a selection of pairing type and strengths (see input description). In addition there is a logical variable `turnoff_zpe` which allows turning off the center-of-mass correction.

Some quantities are first set negative to see whether required input is missing. Then a predefined Skyrme force with the given name is sought; if it is found, it is simply copied into `f`. If no force of this name is found, a new one is composed from the numbers given in the input.

If the input varable `turnoff_zpe` is true, the indicator `f%zpe` is set to 1, which implies not doing anything about the center-of-mass correction. Actually, in the current version this affects only one statement in the static module.

Now the "b" coefficients as given in Eq. (7) are calculated straightforwardly. There is one additional coefficient `slate` used for the Slater approximation to the Coulomb exchange term. This is not a free parameter, but precomputed for convenience in `forces.f90`. The b and b' coefficients are used in subroutine `skyrme` (module `Meanfield`) and `energy` (module `Energies`).

The variable `pairing` from the namelist then determines the pairing. If it is set to `'NONE'`, no pairing is included. Otherwise the strength parameters are taken from the input or from the predefined force. If this process does not find a reasonable pairing combination, stop with an error message.

Finally the routine calculates the values of `nucleon_mass` and `h2ma`. It then prints out a description of the force and pairing parameters.

5.8. Module `Fourier`

This module initializes the `FFTW3` package for doing Fourier transforms [75]. It needs the file `fftw3.f` from the `FFTW3` package to define the symbolic parameters for the initialization calls.

Note that `FFTW` is used only for the complex transforms of the wave functions and the Coulomb solver; the Fourier derivatives of the real fields are handled by explicit matrix multiplication (see module `Trivial`, routines `rmulx`, `rmuly`, and `rmulz`). The definition of the plans for the Coulomb solver is contained in subroutine `coulinit` in module `Coulomb`.

5.8.1. Module variables

The module variables describe the various `FFTW` plans used in the code.

`pforward`, `pbackward`: plans for full three-dimensional forward and backward transforms for both spin components.

`xforward`, `xbackward`: one-dimensional forward and backward transform in the x-direction, but for all values of y, z, and spin s.

`yforward`, `ybackward`: one-dimensional forward and backward transform in the y-direction, but for all values of x, z, and spin s.

`zforward`,`zbackward`: one-dimensional forward and backward transform in the z-direction, but for all values of x, y, and spin s.

5.8.2. Subroutine `initfft`

In this subroutine the `FFTW` system is initialized and a 2-wave function array `p` is allocated temporarily for performing the tests to make the plans efficient (we do not use dynamic allocation, since having this array on the stack may produce different results than on the heap — a point which has not been tested, though).

This is followed by the calls to set up the plans. These have different variations, since the way in which the index or indices over which the transform is done are intertwined with the unaffected indices is quite different for the different directions. Understanding this Section requires a thorough familiarity with the FFTW documentation and Fortran indexing.

An important point to consider when modifying the code: the setting-up of a plan must agree with its later use with respect to whether the input and output arrays are the same (in-place transform) or different. This is why `p` sometimes has a last index of 2 in these calls: in the code all transforms are in-place except for `xforward`, `yforward`, and `zforward`. It was found that very strange things happen if this rule is not obeyed.

5.9. Module `Fragments`

This module is concerned with setting up the initial condition from data files containing fragment wave functions, usually obtained in a previous static calculation. A typical application is the initialization of a heavy-ion reaction, see Section 2.5.4. Beyond that, any number of fragments (limited by the parameter variable `mnof`) can be put into the grid at prescribed positions with given initial velocities. A condition is, however, that the grid they are defined in is smaller than the new grid. If they are put close to the boundary, they may have density appearing on the other side because of periodicity; this may be acceptable if the boundary condition is periodic. A calculation may be restarted on a larger grid, but this may be accompanied by some loss of accuracy, since the wave function outside the original regions is set to a constant small value.

For the case of parallel `MPI` calculations, there is logic to read wave functions distributed over a series of files as described in connection with subroutine `write_wavefunctions`. Since only the dynamic case can run under `MPI` at present, the only application of this is to restart a dynamic calculation. The number of processors may be different in the restart.

5.9.1. Module variables

`fix_boost` **(input)** : if this is set to true, the boost (initial kinetic energy) values are used unchanged from the input; otherwise they are calculated from the initial kinetic energy of relative motion `ecm` and the impact parameter `b`. This implies a two-body initial configuration. The initial motion is assumed to be in the (x, z)-plane in this case. Note that for two initial fragments fix_boost can also be set to true for special initial conditions.

`filename` **(input)** : for each fragment this indicates the name of the file with the associated wave functions. One file can be used several times in the case of identical fragments, abut the code does not treat that as a special case.

`fcent` **(input)** for fragment no. `i` the initial three-dimensional position is determined by `fcent(:,i)`.

`fboost` **(input)** gives the initial motion `fboost(:,i)` of fragment `i` in 3 dimensions. This is not a velocity, but the total kinetic energy of the fragment *in that Cartesian direction* in MeV. The sign indicates the direction, positive or negative along the corresponding axis. The sum of absolute values thus is the total kinetic energy of the fragment. These values are used only if `fix_boost` is true.

The following variables are used only if fix_boost is false, i. e., for two-body initialization from relative motion.

`ecm, b` **(input)** The kinetic energy of relative motion in MeV and the impact parameter in fm.

`vx, vz` The components of the relative velocity in the (x, z)-plane

`xli` The angular momentum of relative motion in units of $\hbar$.

The following quantities have names of other variables prefixed by an "f" and *are indexed by fragment number*; for the most part they duplicate variables describing the whole system on a fragment-by-fragment basis.

`fcmtot`, `fmass`, `fcharge` Centers of mass, masses, and charges of the fragments.

`fnneut`, `fnprot`, `fnstmax`, `fnpmin`, `fnpsi` These have the same meaning as the variable without "f", but apply to the set of wave functions for one specific fragment.

`fnumber` gives the number of wave functions in the fragment for each isospin. For initialization of a dynamic calculation only wave functions with a nonzero occupation are taken into account.

`fnewnpmin`, `fnewnpsi` are the starting and ending indices for each fragment's wave functions in the total set of wave functions combining all the fragments.

`fnx`, `fny`, `fnz` indicate the grid size on which the fragment wave functions are defined.

`fnode` and `flocalindex` are used for MPI. When the wave functions were written by the parallel code, each node produced a file named "*nnn*.`wffile`", which contains the wave functions present on that node. For the wave function with index `nst`, `fnode(nst)` indicates the number of the file and `flocalindex(nst)` the position in the file. These are analogous to the variables `node` and `localindex` for the complete calculation and are explained more thoroughly in the description of module `Parallel`, see section 5.16.

5.9.2. Subroutine `getin_fragments`

This subroutine reads the input variables, makes some consistency checks, and then prepares for the initialization.

A restart is set up by placing one fragment taken from `wffile` at the origin with zero velocity (see Sect. 2.9).

The main loop is over fragments. The fragment files are opened a first time to obtain information about the fragments, which is stored in fragment-specific arrays. The checks include agreement of the forces and grid spacings as well as that the fragment grid is not larger than the new grid.

The code at this point makes a distinction between static and dynamic modes: for a static calculation it is assumed that the data may be needed for a restart. In this case all wave functions are read in even if not occupied even fractionally. For the dynamic case only those with non-zero occupation are input as determined from `fwocc`. This yields a reduced value for `fnumber`.

Note that at present it is assumed that the static wave functions are ordered in ascending energy, so that the occupied ones will start at index one and all empty states will be at the uppermost index positions.

Following this, the index positions in the new wave function array are calculated in `fnewnpmin` and `fnewnpsi` by adding the numbers of wave functions of each fragment-specific successively. Note that at this stage the proton indices are still counted from one.

After this input loop, the indices `npmin` and `npsi` as well as the total number `nstmax` for the combined system are calculated and finally the proton indices where the fragment wave functions should be inserted are shifted to behind the neutrons, i. e., starting at `npmin(2)`.

At the end the two-body initialization `twobody_init` is called for the dynamic case with two fragments and `fix_boost=.FALSE.`. This calculates the `fboost` values from `ecm` and `b`.

5.9.3. Subroutine `read_fragments`

This subroutine prints summary information about which fragment wave functions occupy which range of indices. Then it does a loop over fragments to read in their wave functions using `read_one_fragment`, followed by applying the boost to them using `boost_fragment`.

5.9.4. Subroutine `read_one_fragment`

This subroutine reopens a fragment file for the fragment indexed by `iff` and reads the wave functions, inserting them at the correct index into the new wave function array while also moving them to the desired center-of-mass position.

The principal loop is over isospin. The index limits in the fragment file for that isospin are put into `il` and `iu`, and the new indices into `newil` and `newiu`. In the next step the grid coordinates and the single-particle quantities are read. The latter are copied into the new arrays and the isospin is also recorded. Then this information is printed.

Once this loop is concluded, the spatial shift is prepared. The shift is calculated from the difference between the desired position `fcent` with respect to the origin of the new coordinates given by `x` etc., and the fragment center-of-mass `fcmtot` with respect to the origin in fragment coordinates `fx` etc. Subroutine `phases` is used to calculate essentially the shift phase factor $\exp(-\mathrm{i}\vec{k}\cdot\Delta\vec{r})$, which is a product of phases in each coordinate direction `akx`, `aky`, and `akz`. These have an index corresponding to $\vec{k}$ in the Fourier transform.

Now the index arrays `fnode` and `flocalindex` are read, which indicate where the wave functions are to be found in case of an MPI job. The logical variable `multifile` records whether this is the case by testing whether any of the wave functions was stored on other than node 0.

The following loop runs over the new index positions. If the wave function is not on the current node, nothing is done except for ignoring the input record.

For the case of multiple files for one fragment (which is recognized by not all the indices `fnode` being zero, a short subroutine `locate` is used to position input at the correct location.

Otherwise the wave function is read from the fragment file using the fragment grid dimensions into a variable `ps1` defined with the full new dimension, then it is Fourier transformed, multiplied by the phase factor, and transformed back. Finally it is inserted into the total wave function array `psi`, where any zeroes are replaced by a small number, presently set to 10^{-20}.

5.9.5. Subroutine `locate`

This has the task to position the file for reading wave functions at the correct place. It has as arguments the number of the file (corresponding to the node number in the previous calculation) and the index of the wave function in this file. The variable `presentfile` keeps track of which file is currently open.

If we are starting to read a new file (which is always true initially, as no wave functions are written into the header file `wffile` itself), it closes the present file, composes the file name for the new one and opens it. Then the proper number of records are ignored to position at the correct one. If the file has not changed, it simply returns and reading continues sequentially. The logic of course assumes that files are stored sequentially in each partial file, but the division into partial files need not be the same as in the previous run.

5.9.6. Subroutine `phases`

This subroutine calculates the phase factors for a one-dimensional translation Δx as

$$\exp\left(-\frac{2\pi \mathrm{i} k_j \Delta x}{L}\right).$$

The shift was calculated in `read_one_fragment`, the argument `c` includes a denominator $L = \mathtt{nx} * \mathtt{dx}$, the total length of the grid. The momentum k_j is determined in the usual way for the finite Fourier transform.

5.9.7. Subroutine `twobody_init`

The purpose of this subroutine is to calculate the boost values from `ecm` and `b` in the two-body case. The calculation can be followed with an elementary understanding of the two-body system kinematics, so we just give a brief overview.

The reduced mass `xmu`, the relative velocity `vrel` and the angular momentum `xli` are calculated first. Then the components of the vector linking the two centers of mass `dix` and `diz` are divided by the length of this vector `roft` to get the direction cosines.

The Coulomb energy is calculated assuming two point charges at distance `roft` and is subtracted from `ecm` to yield the kinetic energy remaining at this distance, from which the relative velocity `vrel_d` is calculated. Since the total center of mass is assumed to be at rest, the velocities of the fragments `v1` and `v2` can then be simply obtained. The instantaneous impact parameter `b_d` results from the angular momentum and the relative velocity, and the angle by which the two fragments need to miss each other in order to realize this impact parameter is computed in `sint` and `cost`.

This now makes it possible to calculate the boost velocity components, which are converted into kinetic energies to conform to the definition of `fboost` as given in the input. Note that these energies are signed to indicate the direction of motion.

Both velocities and energies are printed.

5.9.8. Subroutine `boost_fragment`

This subroutine multiplies the configuration-space wave functions of fragment no. `iff` by plane-wave factors to give them translational motion. This is done by calculating the wave number vector `akf` from the kinetic energy. There is one subtle point: since the boost is applied to *single-particle* wave functions, the momentum k should be the correct one for *one nucleon.* Thus the total kinetic energy of the fragment is

$$T = A\frac{\hbar^2}{2m}k^2,$$

where A is the mass number of the fragment and m the nucleon mass. Solving for k and taking the sign into account yields the expression in the code.

The rest is then straightforward.

5.10. Module `Grids`

This module deals with the definition of the spatial grid and associated operations.

5.10.1. Module variables

`nx`, `ny`, `nz`: Number of points in each Cartesian direction. All must be even numbers to preserve reflection symmetry.

`dx`, `dy`, `dz`: The spacing between grid points (in fm) in the three Cartesian directions. Usually these should be identical.

`periodic`: logical variable indicating whether the situation is triply periodic in three-dimensional space.

`wxyz`: the volume element `wxyz=dx*dy*dz`.

`x`, `y`, `z`: arrays containing the actual coordinate values in fm. They are dimensioned as `x(nx)`, `y(ny)`, `z(nz)` and are allocated dynamically, thus allowing dynamic dimensioning through input values of `nx`, `ny`, and `nz`.

`der1x`, `der1y`, `der1z`: matrices describing the first spatial derivatives in the x-, y-, and z-direction, respectively. They are dynamically allocated with dimensions `(nx,nx)` etc. and are calculated in subroutine `sder`.

`der2x`, `der2y`, `der2z`: matrices describing the second spatial derivatives in the x-, y-, and z-direction, respectively. They are dynamically allocated with dimensions `(nx,nx)` etc. and are calculated in subroutine `sder2`.

`cdmpx`, `cdmpy`, `cdmpz`: matrices describing the damping operation in the x-, y-, and z-direction, respectively. They are dynamically allocated with dimensions `(nx,nx)` etc. and are calculated using subroutine `setup_damping`, which in turn calls `setdmc`.

5.10.2. Subroutine `init_grid`

This subroutine is called during the initialization for both static and dynamic calculations. It reads the grid dimension and spacing information using namelist `Grid`, allocates the necessary arrays, and calculates the coordinate values and the derivative and damping matrices.

This is done by calling the subroutine `init_coord` once for each direction. It does everything needed except the calculation of the volume element.

5.10.3. Subroutine `init_coord`

In this subroutine the defining information for a grid direction (generically called `v`, which can be replaced by `x`, `y`, or `z`) in the form of the number of points `nv` and the spacing `dv` is used to generate the associated data. The arrays of coordinate values, derivative and damping matrices are allocated. Since all the quantities that are later used in the code are passed as arguments, this subroutine can handle all three directions in a unified way. To print the information intelligibly, it is also passed the name of the coordinate as `name`.

It is assumed that the coordinate zero is in the center of the grid, i. e., since the dimension is even the number of points to each side of zero is equal andthe origin is in the center of a cell. The special position of the origin is used in static calculations, e. g., for the parity determination. In other situations, the position of the center of mass is more important, this is defined in module `Moment`.

If a difference location of the origin in the grid is desired, it can be done by changing the statement generating the values of `v`.

Finally the derivative matrices and damping matrix are computed using `sder`, `sder2`, and `setdmc`.

5.10.4. Subroutine `sder`

This subroutine calculates the matrix for the first derivative using equation 27a. It receives the dimension `nmax` and the grid spacing `d` as arguments and returns the matrix in `der`.

5.10.5. Subroutine `sder2`

This subroutine calculates the matrix for the second derivative using equation 27b. It receives the dimension `nmax` and the grid spacing `d` as arguments and returns the matrix in `der`.

5.10.6. Subroutine `setup_damping`

This sets up the damping matrices by calls to `setdmc` for each coordinate direction. The reason for not including this in `init_grid` is that it used only in the static calculation and requires the damping parameter `e0dmp`, which is in the static module. It has to be passed as a parameter because circular dependence of modules would result otherwise.

5.10.7. Subroutine `setdmc`

This subroutine calculates the matrices corresponding to the one-dimensional operators

$$\frac{1}{1+\hat{t}/E_0} \text{ with } \hat{t} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}.$$

Here E_0 is the damping parameter called `e0dmp` in the code. Since the kinetic energy operator $\hat{t}$ contains the parameter $\hbar^2/2m$, which is force-dependent, this subroutine depends on module `Forces`.

The calculation proceeds simply by constructing the unit matrix, adding the operator to it to form the denominator, and then calculating the inverse matrix using subroutine `gauss`.

5.10.8. Subroutine `gauss`

This is a Fortran 95 implementation of the standard Gauss algorithm with pivoting. It is simplified for the special case of computing $B = B^{-1}A$ with both matrices dimensioned `(n,n)`.

5.11. Module `Inout`

This module contains the procedures for binary input and output of the larger fields. There are two variants, both of which are written at regular intervals: the wave function file `wffile` and the files containing the densities and currents. Since the former is extremely space-consuming, each output normally overwrites the previous one. These files are intended to be used for a restart, as initialization input (static solution for one fragment) for another run, or for a final analysis of the wave functions.

The densities, on the other hand, are written on a series of file `nnnnnn.tdd`, where `nnnnnn` is the number of the time step or iteration. This is useful for later graphical or other types of analysis.

Note: where the variable name `wffile` is used inside file names in the following, it should not be taken literally but is replaced by the character string it contains.

In addition the routine for printer plots, `plot_density`, is included in this module, as well as the subroutines `sp_properties` and `start_protocol`, which do not completely match the purpose of this module but are placed here for convenience.

5.11.1. Subroutine `write_wavefunctions`

This subroutine has two modes of operation depending on whether the code runs on distributed-memory systems in `MPI` mode or on a shared-memory or single-processor machine. In both cases it first determines the number of filled single-particle states `number(iq)`, which need not be the same as either the number of particles or the number of states, since pairing may lead to partial occupation and in addition there can be empty states.

Sequential operation: this case is recognized recognized by `mpi_nprocs==1`. Open `wffile`, then write four records containing general information.

Record 1: `iter`, `nstmax`, `nneut`, `nprot`, `number`, `npsi`, `charge_number`, `mass_number`, `cm`.

Record 2: `nx`, `ny`, `nz`, `dx`, `dy`, `dz`, `wxyz`.

Record 3: `x`, `y`, `z`.

Record 4: `wocc`, `sp_energy`, `sp_parity`, `sp_norm`, `sp_kinetic`, `sp_efluct1`.

These are followed by one record containing information for the `MPI` case, which is included here only for compatibility: `node`, `localindex`. This is then followed by a series of `nstloc` records (in the sequential case, `nstloc` equals `nstmax`), containing the array of `nx*ny*nz*2` wave function values for each single-particle state (including spin).

MPI operation: in this case processor #0 writes the same general data as in the sequential case onto file `wffile`, which is then closed. The purpose of record 5 in this case is to record for each wave function (in global index space) which node it is in and what the index on that node is. Since each node produces a separate output file with only its wave functions, this allows reading any wave function correctly from the set of files.

Each processor thus only writes the wave function data for its locally stored set of `nstloc` wave functions onto files with the names composed (in variable `rsfp`) of the number of the processor and `wffile` in the form `nnn.wffile`. For example, if `wffile` has the value 'Ca40', these files will be `000.Ca40`, `001.Ca40`, `002.Ca40`, etc. up to the number of processors.

5.11.2. Subroutine `write_densities`

This subroutine produces a file `iter.tdd` with density and current data for the present time step or iteration with number `iter`. In the file name `iter` is given in 6 decimal digits. The record structure is as follows:

- Record 1: this contains the variables `iter`, `time`, `nx`, `ny`, and `nz` to define the dimensions of the fields.
- Record 2: contains the variables `dx`, `dy`, `dz`, `wxyz`, `x`, `y`, and `z` to allow proper labelling of axes in plots, etc.
- Further records: for each field to be written, a record is produced with the following information:
  1. Name of the field with up to 10 characters
  2. Logical value `scalar` to indicate whether it is a scalar (`.FALSE.`) or a vector field (`.TRUE.`).
  3. Logical value `write_isospin` to indicate whether the field is summed over protons and neutrons ((`.FALSE.` or not (`.TRUE.`). In the latter case the field has a last index running from 1 to 2 for neutrons and protons, respectively. **This selection applies to all fields equally (except the Coulomb potential)**.

  After this identification record, the corresponding field itself is written. The dimension varies in the following way:

`scalar`	`write_isospin`	`dimension`
.FALSE.	.FALSE	(nx,ny,nz)
.FALSE.	.TRUE.	(nx,ny,nz,2)
.TRUE.	.FALSE.	(nx,ny,nz,3)
.TRUE.	.TRUE.	(nx,ny,nz,3,2)

The **selection of fields to be output** is handled through variable `writeselect` consisting of `nselect` characters. Each field is selected by a one-character code, where both lower and upper case are acceptable. At present the choices are:

- R: density `rho` (scalar). Name `Rho`.
- T: kinetic energy density `tau` (scalar). Name `Tau`
- U: local mean field `upot`. Name `Upot`.
- W: Coulomb potential `wcoul` (scalar). This has to be handled specially, since it has no isospin index. Name `Wcoul`.

- C: current density `current` (vector). Name `Current`.
- S: spin density `sdens` (vector). Name `Spindens`.
- O: spin-orbit density `sodens`. Name `s-o-Dens`.

This system is set up to be easily modified for writing additional fields.

5.11.3. Subroutine `write_one_density`

This subroutines does the actual output for `write_densities` in the case of a scalar field. Its functioning should be clear from the description above.

5.11.4. Subroutine (Private)

This also does the actual output for subroutines `write_densities` for the case of a vector field. Its functioning should be clear from the description above.

5.11.5. Subroutine `plot_density`

Produces a simple printer plot of the density distribution in the reaction plane. This is not supposed to replace better plotting codes, but simply allows a quick glance at what is happening in the code, even while it is running.

It is based on a very old routine found at ORNL and was translated into modern Fortran. It uses helper function for interpolation.

5.11.6. Subroutine `sp_properties`

In this routine the kinetic energy, orbital and spin angular momenta expectation values, `sp_kinetic`, `sp_orbital` and `sp_spin` of the single-particle states are calculated. The latter are both three-dimensional vectors.

Note that the single-particle energy `sp_energy` itself is not calculated here but in the main static and dynamic routines, since it is obtained by applying the single-particle Hamiltonian, which is done more conveniently there.

The procedure is quite simple: in a loop over wave functions the active one is copied into `pst` for convenience. Then its three directional derivatives `psx`, `psy`, and `psz` and Laplacian `psw` are calculated. In the big loop over the grid they are combined to the desired matrix elements; the only technical point to remark is that since the result must be real, efficiency can be achieved by formulating the complex products in an explicit way. Then `kin` contains the kinetic energy (without the $\hbar^2/2m$), `cc` the orbital and `ss` then spin matrix elements.

Finally only the volume element, the factor of one half for the spin ad the prefactor of the kinetic energy are added.

5.11.7. Subroutine `start_protocol`

This is given a file name and a character string for a header line to start the file contents. It is used for the `*.res` files. If the file already exists, nothing is done, since this probably a restart job and output should just be added at the end of the file.

5.12. Module `Levels`

This module is concerned with the wave function data: definition of the pertinent arrays, allocating their storage and simple operations on them.

5.12.1. Module variables

First there are some general variables describing the arrays:

`nstmax`: is the total number of wave functions present in the calculation. For the MPI version only `nstloc` wave functions are present on each node. Note that for all other wave-function related arrays, such as single-particle energies, the full set is stored on each node.

`nstloc`: the number of wave functions stored on the present node. In principle this should be defined in module `Parallel`, but this would lead to a circular module dependence.

`nneut`, `nprot`; the physical numbers of neutrons and protons. These may be smaller than the number of single-particle states.

`npmin`, `npsi`: the neutron states are numbered `npmin(1)` through `npsi(1)` and the proton states run from `npmin(2)` through `npsi(2)`. Protons follow neutrons, so `npmin(1)=1` and `npmin(2)=npsi(1)+1`. *Note that for each particle type the number of states can be larger than the particle number, as states may be fractionally occupied or even empty.* If initialization is not from fragments, `npsi(2)` *as an input value* refers to the total number of proton states, it is later updated (in `init.f90`) to its normal meaning as the final index for proton states, which coincides with the total number of states, `npsi(2)=nstmax`.

`charge_number`, `mass_number`: the physical charge and mass numbers.

We have to distinguish three different numbers: the number of physical particles `nneut` or `nprot`, the number of single particle states read as `npsi(1:2)`, and the number of states actually having nonzero occupation, which can differ from the particle number if pairing is used. Since the occupation given by `wocc` changes with iteration, this latter number may be iteration dependent. It becomes important for the output of the wave functions, as unfilled states need not be read from fragment data files in the dynamic case. Therefore the numbers of states with nonzero occupation are computed only in subroutine `write_wavefunctions`, where they are called `number(1:2)`..

Next are the arrays for the wave functions themselves and the single-particle Hamiltonian matrix. Each wave function is complex dimensioned `(nx,ny,nz,2)` with the last index denoting spin (1=up, 2=down for the z-direction).

`psi`: this is the main array for the wave functions. It has an additional last index counting the states. In the sequential case it runs from $1 \ldots$`nstmax`, in the MPI version each node has only `nstloc` wave functions.

`hmatr`: this is dimensioned `(nstmax,nstmax)` and is used for the single-particle Hamiltonian in the diagonalization step.

Finally arrays dimensioned `nstmax` to describe properties of each wave function.

`sp_energy`: single-particle energy in MeV.

`sp_efluct1`: single-particle energy fluctuation [MeV] calculated as

$$\sqrt{\langle\psi|\hat{h}^2|\psi\rangle - \langle\psi|\hat{h}|\psi\rangle^2}.$$

Used only as informational printout in the static part.

`sp_efluct2`: single-particle energy fluctuation [MeV] calculated as

$$\sqrt{\langle\hat{h}\psi|\hat{h}\psi\rangle - \langle\psi|\hat{h}|\psi\rangle^2}.$$

Used only as informational printout in the static part.

`sp_kinetic`: single-particle kinetic energy in units of MeV.

`sp_norm`: norm of single-particle wave function; should be unity normally.

`sp_parity`: single-particle parity w.r.t. three-dimensional reflection at the origin; calculated as

$$\sum_s \int \mathrm{d}^3r\, \psi^*(\vec{r},s)\psi_s(-\vec{r},s).$$

`wocc`: occupation probability of single-particle state, may be fractional because of pairing. In the equations this is usually denoted as w_k^2, the square added because of the pairing notation.

`sp_orbital`: dimensioned `(3,nstmax)`: expectation values of three components of single-particle orbital angular momentum, in units of $\hbar$.

`sp_spin`: dimensioned `(3,nstmax)`: expectation values of three components of single-particle spin, in units of $\hbar$.

`isospin`: keeps track of isospin of particle, 1=neutron, 2=proton.

5.12.2. Subroutine `alloc_levels`

This subroutine allocates all the arrays associated with single-particle wave functions. Note that while most have dimension `nstmax`, `psi` itself is dimensioned for the number `nstloc` of wave functions on one specific processor. It also records the isospin value.

5.12.3. Subroutines `cdervx`*,* `cdervy`*,* `cdervz`

These three routines calculate derivatives of wave functions using the FFT method explained in section 2.7.3, in the x-, y-, and z-direction, respectively. The first argument is the wave function ψ to be differentiated, the second returns the first derivative $\frac{\partial\psi}{\partial x}$, and the third one the second derivative $\frac{\partial^2\psi}{\partial x^2}$, where x can be replaced by y or z, of course. The last argument can be omitted and no second derivative is calculated in this case. The derivatives add the proper dimension of fm^{-1} and fm^{-2}, respectively. Note the dependence of the k-value on index as explained in Eq. 25b.

5.12.4. Subroutine `laplace`

depending on the presence of the third argument `e0inv`, it can calculate two things using FFT methods:

- if `e0inv` is not present, it calculates the Laplacian
$$\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2}.$$
- if `e0inv` is present and positive, it calculates
$$\frac{1}{E_{0\mathrm{inv}}+\hat{t}}\psi,$$
with the kinetic-energy operator
$$\hat{t}=-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right),$$
which is of course expressed through $\vec{k}$ in momentum space.

  The methods used in this routine are similar to those for the derivatives `cdervx` etc.

5.12.5. Subroutine `schmid`

This performs a Gram-Schmidt orthogonalization of the single particle levels in the straightforward way: loop over states and subtract the components of all lower-indexed states from a wave function. It can be parallelized under `OpenMP`, albeit not very efficiently, but becomes too slow under `MPI`, so that `MPI` is not enabled for the static calculations.

5.13. Module `Meanfield`

5.13.1. Purpose

This module calculates all the ingredients needed for the energy functional and for applying the single-particle Hamiltonian to a wave function.

The work is done by two subroutines: `skyrme` for the calculation of all the fields, which can be scalar or vector and isospin-dependent. `hpsi` then is the routine applying the single-particle Hamiltonian to one single-particle wave function.

Note the division of labor between `skyrme` and `add_density` of module `Densities`: everything that constructs fields — densities and current densities — from the single-particle wave functions is done in `add_density`, which is called in a loop over the states, while `skyrme` does the further manipulations to complete the fields entering the single-particle Hamiltonian by combining the densities and their derivatives. It does not need access to the wave functions.

5.13.2. Module variables

All of the variables in this module are fields, either of scalar or vector character. Their dimension is `(nx,ny,nz,2)` for the scalar and `(nx,ny,nz,3,2)` for the vector fields with the last index indicating isospin in all cases. They are derived from the densities calculated in module `Densities`.

`upot`: this is the local part of the mean field U_q as defined in Eq. (8b). It is a scalar field with isospin index.

`bmass`: this is the effective mass B_q as defined in Eq. (8c). It is a scalar, isospin-dependent field.

`aq`: This is the vector filed $\vec{A}_q$ as defined in Eq. (8e). It is a vector, isospin-dependent field.

`divaq`: this is simply the divergence of `aq`, i. e., $\nabla \cdot \vec{A}_q$. Its is a scalar, isospin-dependent field.

`spot`: the field $\vec{S}_q$ as defined in Eq. (8f). It is a vector, isospin-dependent field.

`wlspot`: the field $\vec{W}_q$ as defined in Eq. (8d). It is a vector, isospin-dependent field.

5.13.3. Subroutine `alloc_fields`

This subroutine has the simple task of allocating all the fields that are local to the module `Meanfield`.

5.13.4. Subroutine `skyrme`

In this subroutine the various fields are calculated from the densities that were previously generated in module `Densities`. The method of calculation is pretty much a direct application of the equations given in Section 2.2.5, but with a few modifications for efficiency. The expressions divide up the contributions into an isospin-summed part with b-coefficients followed by the isospin-dependent one with b'-coefficients. As it would be a waste of space to store the summed densities and currents, the expressions are divided up more conveniently. If we denote the isospin index q by `iq` and the index for the opposite isospin q' by `ic` (as `iq` can take the values 1 or 2, it can conveniently be calculated as `3-iq`), this can be written for example as

$$b_1\rho - b'_1\rho_q \longrightarrow b_1(\rho_q + \rho_{q'}) - b'_1\rho_q$$
$\longrightarrow$`(b1-b1p)*rho(:,:,:,iq)+b1*rho(:,:,:,ic)`

This decomposition is used in all applicable cases.

For intermediate results the fields `workden` (scalar) and `workvec` (vector) are used.

Now the subroutine proceeds in the following steps:

- **Step 1:** all the parts in `upot` (see Eq. (8b) involving an α-dependent power of the density are collected. Note that in order to avoid having to calculate different powers, ρ^α is factored out. The division by the total density uses the small number `epsilon` to avoid division by zero.
- **Step 2:** the divergence of $\vec{J}$ (`sodens`) is calculated for both isospins in `workden` and the contributions are added to `upot`.
- **Step 3:** the Coulomb potential is calculated using subroutine `poisson` (see module `Coulomb`). It and the Slater exchange correction (only if the `ex` parameter in the force is nonzero) are added to `upot` for protons, `iq=2`.
- **Step 4:** the Laplacian is applied to the densities and the result stored in `workden`. Then the remaining terms of Eq. (8b) are constructed. Note that the `iq`-loop is combined with the following steps.
- **Step 5:** the effective mass is calculated according to Eq. (8c).
- **Step 6:** the gradient of the density is calculated and the spin-orbit vector $\vec{W}_q$ is constructed in `wlspot` according to Eq. (8d).
- **Step 7:** the curl of the spin density vector is calculated and stored in `workvec`.
- **Step 8:** the vector $\vec{A}_q$ is calculated according to Eq. (8e) from the current density and the curl of the spin density.
- **Step 9:** the curl of the current density is calculated and stored in `spot`.
- **Step 10:** now the two isospin contributions in `spot` are combined in the proper way according to Eq. (8f). This way of handling it avoids the introduction of an additional work vector for $\nabla \times \vec{\jmath}_q$.
- **Step 11:** the divergence of $\vec{A}_q$ is calculated and stored in `divaq`.
- **Step 12:** finally, the gradient of the effective mass term B_q (Eq. (8c) is calculated and stored in the vector variable `dbmass`.

This concludes the calculation of all scalar and vector fields needed for the application of the Skyrme force.

5.13.5. *Subroutine* `hpsi`

This subroutine applies the single-particle Hamiltonian to a single-particle wave function `pinn` to produce an output wave function `pout`. The argument `iq` indicates the isospin for the wave function and `eshift` is an energy shift which is zero in the dynamic calculation but crucial to the static algorithm (see `grstep` in module `Static`).

For an understanding of this module the role of the following local variables is crucial. They are

- `is`: this is used in the loops over spin to indicate the spin component: `is=1` for spin up and `is=2` for spin down.
- `ic`: denotes the index for the opposite spin; it is calculated as `ic=3-is`. Note the similar handling of the two isospin projections using `iq` and `icomp` in subroutine `skyrme`.
- `sigis`: this variable denotes the sign of the spin projection. It is calculated as `sigis=3-2*is` and thus is `+1` for spin up (`is=1`) and `-` for spin down (`is=2`).

The general structure of the subroutine is as follows: first the part of the Hamiltonian not involving derivatives is applied, followed by the terms involving derivatives in order x, y, z. Since the structure of the Hamiltonian involves only first or second derivatives in one spatial direction in each term, the derivatives can be calculated for one direction and then the working space can be reused for the next one.

The expressions for the different spatial derivatives are quite analogous, so that only the x-direction will be discussed at length below.

When examining the expressions one by one, please refer to Eq. (8a), which for convenience is repeated here:

$$\hat{h} = U_q(\vec{r}) - \nabla \cdot [B_q(\vec{r})\nabla] + \mathrm{i}\vec{W}_q \cdot (\vec{\sigma} \times \nabla) + \vec{S}_q \cdot \vec{\sigma} - \frac{\mathrm{i}}{2}\left[(\nabla \cdot \vec{A}_q) + 2\vec{A}_q \cdot \nabla\right]. \qquad (34)$$

- **Step 1:** the non-derivative parts not involving spin. These arise from U_q and $-\frac{\mathrm{i}}{2}\nabla \cdot \vec{A}_q$, which are combined into a complex expression. The energy shift `eshift` is also included.
- **Step 2:** the spin current coupling is constructed by simply using the explicit definition of the Pauli matrices and multiplying the resulting matrix onto the spinor wave function.
- **Step 3:** the first and second derivative in the x-direction are evaluated and stored in the arrays `pswk` and `pswk2`. The last term in the Hamiltonian gives rise to the two contributions
$$-\frac{\partial B_q}{\partial x}\frac{\partial}{\partial x} - B_q\frac{\partial^2}{\partial x^2},$$
of which the second is evaluated straightforwardly, while the first one is combined with the spin-orbit contribution. The part of $\mathrm{i}\vec{W}_q \cdot (\vec{\sigma} \times \nabla)$ that contains an x-derivative is
$$(\mathrm{i}W_y\sigma_z - \mathrm{i}W_z\sigma_y)\frac{\partial}{\partial x} = \begin{pmatrix} \mathrm{i}W_y & -W_z \\ W_z & -\mathrm{i}W_y \end{pmatrix}\frac{\partial}{\partial x}$$

This is programmed employing the variable `sigis` to account for the different signs in the rows of the matrix.

- **Step 4:** for the derivatives in the y-direction the procedure is similar; the spin-orbit part is now

$$(\mathrm{i}W_z\sigma_x - \mathrm{i}W_x\sigma_z)\frac{\partial}{\partial y} = \begin{pmatrix} -\mathrm{i}W_x & \mathrm{i}W_z \\ \mathrm{i}W_z & \mathrm{i}W_x \end{pmatrix}\frac{\partial}{\partial y}$$

- **Step 5:** for the derivatives in the z-direction the procedure is again similar; the spin-orbit part is now

$$(\mathrm{i}W_x\sigma_y - \mathrm{i}W_y\sigma_x)\frac{\partial}{\partial z} = \begin{pmatrix} 0 & W_x - \mathrm{i}W_y \\ -W_x - \mathrm{i}W_y & 0 \end{pmatrix}\frac{\partial}{\partial z}$$

5.14. Module `Moment`

5.14.1. Purpose

In this module various moments of the density distribution are calculated. Most of them come in two versions: an isospin-dependent one characterized by a final isospin index of dimension 2, and a summed one distinguished by the ending "`tot`" in its name. Thus, e. g., three variants of center of mass (21a) are stored : that of the neutrons `cm(1:3,1)`, of the protons `cm(1:3,2)`, and of the total mass distribution `cmtot(1:3)`.

Since the geometrical arrangement in space can be arbitrary with respect to the Cartesian coordinate system — this is certainly true for non-central collision situations — some quantities associated with the axes become meaningless in the general situation. For this reason, the code also calculates the complete Cartesian quadrupole tensor (21b) and diagonalizes it to obtain the principal axes of the nucleus. In this frame then we compute the spherical quadrupole moments Q_{2m}, see Eq. (21c), with their dimensionless counterparts a_o, a_2 defined in Eq. (21d) and the deformation parameters β and γ from Eq. (21e).

5.14.2. Module variables

`pnr`, `pnrtot`: the numbers of neutrons `pnr(1)`$=N$, protons `pnr(2)`$=Z$, and the total particle number `pnrtot`$=A$. These are obtained by a simple integration of the densities `rho`. Dimensionless.

`cm`, `cmtot`: the center of mass vectors of the neutron, proton, and total mass distribution, $\vec{R}_n$, $\vec{R}_p$, and $\vec{R}$. Dimension: fm.

`pcm`: the integrated momentum vectors, not containing the nucleon mass. They thus correspond to an integral over the current density only and have a dimension of velocity: c.

`rms`(private), `rmstot`: these are the root mean-square radii (21f) of neutron, proton, and total mass distribution. Dimension: fm.

`x2m` (private), `x2mtot` (private): the vectors of second moments of the radii in the three coordinate directions, i. e.,

$$\langle r_i^2\rangle = \frac{1}{A}\int \mathrm{d}^3r\, r_i^2 \rho(\vec{r}).$$

They are useful to get an idea of the shape of the nucleus in static calculations. Dimension: fm^2

`q20` (private), `q20tot` (private): the $m = 0$ components of the spherical quadrupole tensor Q_{20} in the principal-axes frame.

`q22` (private), `q22tot` (private): the $m = 2$ components of the spherical quadrupole tensor Q_{22} in the principal-axes frame.

`beta20` (private), `beta20tot` (private): the quadrupole deformation parameters a_0.

`beta22` (private), `beta22tot` (private): the quadrupole deformation parameters a_2.

`beta`, `gamma`: the Bohr-Mottelson deformation parameters β and γ. These are calculated only for the total mass distribution. `beta` is dimensionless whereas `gamma` is expressed in degrees.

5.14.3. Subroutine `moments`

This is the principal subroutine for calculating the geometric quantities. It consists of two loops, both over isospin and space, and a final analysis Section. In detail:

1. The first loop calculates the particle numbers, centers of mass, and, for the dynamic case only, the momenta divided by nucleon mass.
2. The second loop is separate because the center-of mass must be known to use it as the origin for the vectors. The r.m.s. radii `rms` and the quantities `x2m` are calculated as well as the components `qmat` of the Cartesian quadrupole tensor $Q_{kl} = \int \mathrm{d}^3 r \left(3x_k x_l - r^2\delta_{kl}\right) \rho(\vec{r})$, see Eq. (21b).
3. After this, the subroutine `q2diag` is used to determine the spherical components Q_{20} and Q_{22} according to Eq. (21c) in the principal-axes frame (see Section 2.6.1), also generating some printout in the process. These are then multiplied with a scale factor to yield the dimensionless deformation parameters a_0 and a_2 according to Eq. (21d) and finally by conversion to polar coordinates the Bohr-Mottelson parameters β and γ with Eq. (21e).
4. The Cartesian and polar deformation parameters are then printed.

5.14.4. Subroutine `moment_shortprint`

This subroutine simply prints some information into the specialized output files. `monopolesfile` receives the r.m.s. radii and also the difference of neutron minus proton radius, while `quadrupolesfile` receives the spherical quadrupole components `q20` and `q20tot` as well as the moments `x2m`. The physical time starts each line to enable easy time-curve plotting.

5.14.5. Subroutine `moment_print`

This subroutine prints a somewhat more detailed information. Particle number , r.m.s. radius, Q_{20}, `x2m`, and center-of-mass are printed for the total distribution and also separately for neutrons and protons. This output goes to the regular output unit.

5.14.6. Subroutine `q2diag`

This subroutine diagonalizes the Cartesian quadrupole tensor. To this purpose it calls the `LAPACK` routine `DSYEV`.

The eigenvalues are obtained as `q_eig(i)` in ascending order of magnitude, and the corresponding eigenvectors as `q_vec(:,i)`. Both are printed. Then the corresponding spherical moments are calculated assuming the z-axis is selected as that of largest quadrupole moment:

$$Q_{20} = \sqrt{\frac{5}{16\pi}}\, Q_{zz}, \qquad Q_{22} = \sqrt{\frac{5}{96\pi}}\, \left(Q_{yy} - Q_{xx}\right).$$

They are returned in `q20x` and `q22x`.

5.15. Module `Pairs`

The principal part of this module is the subroutine `pair`, which computes the pairing solution based on the BCS model. It is the only public part of this module. The other subroutines are helper routines that directly use the single-particle properties defined in module `Levels`. The module variable `iq` controls whether the solution is sought for neutrons (`iq=1` or protons `iq=2` and accordingly the single-particle levels from `npmin(iq)` to `npsi(iq)` are affected.

The principal procedure followed is to first calculate the pairing gap for each single-particle state. This determines the occupation numbers `wocc`, which of course are used throughout the program. Then the Fermi energy for the given isospin is determined such that the correct particle number results.

Note that there is a factor of one half in many formulas compared to what is usually found in textbooks. This is because here the sum over states $\sum_k \ldots$ runs over all states, while in textbooks the sum is over pairs, giving half of that result.

5.15.1. Module variables

`eferm`(2): Fermi energy in MeV for the two isospins.

`epair`(2): Pairing energy in MeV for the two isospins. It is given by

$$E_{\text{pair}} = \frac{1}{2}\sum_k \Delta_k u_k v_k.$$

This is a public variable and the sum of the two values is subtracted from the total energies in module `Energies`.

`avdelt`(2): Average gap in MeV for the two isospins. It is given by

$$\frac{\sum_k \Delta_k u_k v_k}{\sum_k u_k v_k}$$

where the sum is over states with the given isospin.

`avg`(2): The average pairing force for each isospin, given by

$$\frac{E_{\text{pair}}}{\sum_k u_k v_k/2}.$$

`deltaf`(nstmax): single-particle gap in MeV for each single-particle state.

5.15.2. Subroutine `pair`

This is the only routine visible from outside the module. It solves the pairing problem and prints out summary information. The principal results used in the rest of the code are the BCS occupation numbers $v_k^2 \rightarrow$ `wocc` and the pairing energies `epair`.

The subroutine is structured straightforwardly: it first calculates the pairing gaps `deltaf` by calling `pairgap`. Then for the two isospin values `pairdn` is called with the correct particle number as argument. This does the real work of solving the equations. Finally summary information is printed.

5.15.3. Subroutine `pairgap`

This subroutine calculates the pairing gaps Δ_k stored in the array `deltaf` for all single-particle states.

First a simplified version is returned if `ipair=1` or for the initial `itrsin` (at present set to 10) iterations of a static calculation. All gaps are set equal to $11.2\,\mathrm{MeV}/\sqrt{A}$ in this case.

In the general case, there is a loop over the two isospin values `iq`. The pairing density is obtained by evaluation of

$$\texttt{work}(\vec{r}) = \sum_k u_k v_k \left|\phi_k(\vec{r})\right|^2 \tag{35}$$

where the simple conversion

$$u_k v_k = v_k\sqrt{1-v_k^2} = \sqrt{v_k^2(1-v_k^2)} = \sqrt{\texttt{wocc}-\texttt{wocc}^2} \tag{36}$$

is used.

The isospin-dependent pairing strength `v0act` is obtained from the force definition. The pairing field $V_P(\vec{r})$ is then given by two different expressions: for `VDI` pairing (`ipair=5`), the pairing density is simply multiplied by `v0act`, while for `DDDI` pairing (`ipair=6`) it is

$$V_P(\vec{r}) = \texttt{v0act}\cdot\texttt{work}\cdot(1-\rho(\vec{r}))/\texttt{rho0pr} \tag{37}$$

involving the *total* density ρ and the parameter `rho0pr` from the pairing force definition.

In the final step the gaps are computed as the expectation values of the pairing field,

$$\Delta_k = \int \mathrm{d}^3r V_P(\vec{r}) \left|\phi_k(\vec{r})\right|^2 . \tag{38}$$

5.15.4. Subroutine `pairdn`

The subroutine `pairdn` determines the pairing solution by using `rbrent` to find the correct Fermi energy for the given particle number. After that a few averaged or integral quantities are calculated.

As the starting value for the Fermi energy the one for gap zero is used, i. e., the average of the first unfilled and last filled single-particle energies. Then `rbrent` is called to calculate the correct solution, after which there is a loop for a straightforward evaluation of the module variables `epair`, `avdelt`, and `avg`.

5.15.5. Subroutine `rbrent`

This subroutine is an adapted version of the *Van Wijngaarden-Dekker-Brent* method for finding the root of a function (see [58]). Given the desired particle number as an argument, it searches for the value of the Fermi energy that makes this particle number agree with that returned by `bcs_occupation`. It is clear that this subroutine is in a very antiquated style of Fortran; it will be replaced at some time in the future.

5.15.6. Subroutine `bcs_occupation`

For a given Fermi energy ϵ_F passed as argument `efermi`, this subroutine evaluates the particle number that would result with such a Fermi energy and returns it as its second argument, `bcs_partnum`. The isospin is controlled by module variable `iq`. First the occupation probabilities are calculated using the standard BCS expression

$$v_k^2 = \frac{1}{2}\left(1 - \frac{\epsilon_k - \epsilon_F}{\sqrt{(\epsilon_k - \epsilon_F)^2 + \Delta_k^2}}\right). \tag{39}$$

They are stored in `wocc`. A small correction is added, so that they are not exactly identical to 1 or 0. The particle number is finally obtained as $N = \sum_k v_k^2$.

5.16. Module `Parallel`

This module organizes the execution on distributed-memory machines using `MPI`. Its interface is made such that on sequential machines `parallel.f90` can simply be replaced by a special version `sequential.f90` which contains a definition of this module generating trivial data.

5.16.1. Parallelization concept

`MPI` parallelization is based on distributing the wave functions onto the different nodes. Thus all operations acting directly on the wave functions can be done in parallel, not only the time evolution by the application of the single-particle Hamiltonian, but also the summing up of the densities and currents over those wave function stored on the node. This means that only the final summation of the densities and the calculations done with them have to be communicated across the nodes.

It is important that the single-particle properties also defined in `Levels`, e.g. `sp_energy` are not split up up for the nodes but the full set is present on each node. The values are communicated by summing from all nodes with zeroes in those index positions not present on a particular one. This method of handling them avoids having to communicate many small arrays.

Since an efficient way of dealing with Gram-Schmidt orthogonalization in this case was yet not found, at present the code can be run in `MPI` parallel mode only for the dynamic case.

5.16.2. Module variables

`tmpi`: a logical variable set to true if `MPI` parallelization is activated. It is used to turn the calling of all the `MPI` routines in the code on or off.

`node`, `localindex`: these are vectors of integers with dimension `nstmax`. For the single-particle state with index `i` the wave function is stored on computing node `node(i)` and its index on that node is `localindex(i)`.

`globalindex`: tells the index of the single-particle state in the whole array of `nstmax` states (it could be dimensioned `nstloc` but is dimensioned as `nstmax` to make its allocation simpler). So for wave function index `i` *on the local node*, `i=1..nstloc`, the single-particle energy must be obtained using `sp_energy(globalindex(i))`.

`mpi_myproc`, `mpi_nprocs`, `mpi_ierror`: these are variables associated with MPI and the MPI documentation should be consulted.

5.16.3. Subroutine (Public)

This subroutine merely allocates the internal arrays of module `Parallel`.

5.16.4. Subroutine `init_all_mpi`

This subroutine initializes `MPI` and finds out the number of processors `mpi_nprocs` as well as the index of the current one `mpi_myproc`. The flag `wflag` is set to true only for the processor numbered 0.

5.16.5. Subroutine `allocate_nodes`

The first loop in this subroutine distributes the wave functions over the nodes. This is done by looping over the wave functions and assigning one to each processor in turn. When the number of processors has been reached, it restarts from processor 0. This way of allocation is to some extent arbitrary and can be changed.

The second loop then calculates which wave functions are present on the local node and records their index `gobalindex` in the complete sequence. The third loop sets up the reverse pointers `localindex`, which has to be done in a loop over all processors to set up the information for the proper global indices.

5.16.6. Subroutine `collect_densities`

This subroutine uses the `MPI` routine `mpi_allreduce` to sum up the partial densities from the different nodes, using temporary arrays `tmp_rho` and `tmp_current` (depending on whether it is a scalar or vector field) in the process.

5.16.7. Subroutine `collect_sp_properties`

This subroutine collects the single-particle properties calculated from the wave functions and thus available only for the local wave functions on each node. It uses a simple trick: the arrays like `sp_energy` are defined for the full set of indices but set to zero before the calculation of these properties. On each node then the local values are calculated but inserted at the proper index for the full set of wave functions. In this subroutine the results from all the nodes are added up using `mpi_reduce`, so that effectively for each index one node contributes the correct value and the others zeroes. This process sounds inefficient but considering the small size of the arrays that does not matter.

5.16.8. Subroutine `finish_mpi`

This is just a wrapper for the `MPI` finalization call.

5.16.9. Using `sequential.f90`

In the sequential case module variables are added to simulate the `MPI` environment. Thus variables with names starting `mpi_` have to be supplied. The only ones whose value is important are the processor number `mpi_myproc` which is always zero in this case, and the total number of processors `mpi_nprocs`, which is 1.

All subroutines with names starting with `mpi_` stop the code, because the calls to `MPI` routines should only happen in parallel execution (they are conditioned by `tmpi`. Thus it is an error if the code runs into one of these.

The indexing arrays become trivial: `node(i)=0`, `localindex(i)=i`, and `globalindex(i)=i` for all `i`.

5.17. Module `Params`

This module contains some general parameters critical to controlling the code and some mathematical and physical constants used everywhere.

5.17.1. Module variables

5.17.2. General parameters

`db`: this is a constant determining the precision of real numbers in the code in a portable manner. In practice it will usually be `REAL(8)`.

`pi`: the constant π.

`hbc`: the constant $\hbar c$ in units of MeV*fm.

`e2`: the electron charge squared. It is calculated as $\alpha\hbar c$ and has units of MeV*fm.

5.17.3. File names and units

These variables allow the user to change the names of some of the files used by the code. Except for `wffile`, output is produced for an iteration or a time step at selected intervals.

`wffile`: file to contain the static single-particle wave functions plus some additional data. It is also used for restarting an interrupted calculation and is rewritten regularly (see variables `trestart` and `mrest`). It can be turned off completely using the name `'NONE'`.

`converfile`: contains convergence information for the static calculation. Default: `conver.res`.

`monopolesfile`: contains moment values of monopole type. Default: `monopoles.res`.

`dipolesfile`: contains moment values of dipole type. Default: `dipoles.res`.

`quadrupolesfile`: contains moment values of quadrupole type. Default: `quadrupoles.res`.

`momentafile`: contains components of the total momentum. Default: `momenta.res`.

`energiesfile`: energy data for time-dependent

`spinfile`: time-dependent total, orbital, and spin angular-momentum data as three-dimensional vectors. calculations. Default: `energies.res`.

`extfieldfile`: contains the time-dependence of the expectation value of the external field. Present only if an external field for boost or time-dependent excitation is used. Default: `extfield.res`.

`scratch`, `scratch2`: the unit numbers used for temporary storage. Default: 11 and 12.

5.17.4. Switches

These are logical variables to turn various features on or off.

`tcoul`: indicates whether the Coulomb field should be included or not.

`tstatic`, `tdynamic`: these are set `true` for a static or dynamic job, respectively. They are not input directly but from the input variable `imode`, which is 1 for the static and 2 for the dynamic case.

`trestart`: if `true`, restarts the calculation from the `wffile`.

`tfft`: if `true`, the derivatives of the wave functions, but not of the densities, are done directly through FFT. Otherwise matrix multiplication is used, but with the matrix also obtained from FFT. Default is `true`.

5.17.5. Output control

`mprint`: control for printer output. If `mprint` is greater than zero, more detailed output is produced every `mprint` iterations or time steps on standard output.

`mplot`: if `mplot` is greater than zero, a printer plot is produced and the densities are dumped onto `*.tdd` files every `mplot` time steps or iterations.

`mrest`: if greater than zero, a `wffile` is produced every `mrest` iterations or time steps.

5.17.6. Globally used variables

`iter`: number of the current time step or iteration. Used in both static and dynamic modes.

`time`: the simulation time of the current step in fm/c; only meaningful in a dynamic calculation.

`wflag`: indicates whether printing is allowed. This is necessary for the parallel job to have only one processor print and concerns both the standard output and the `*.res` files.

`printnow`: this variable is set to true if conditions for printing are met, such as a certain interval in iteration number.

5.17.7. Field output control

`nselect`: parameter limiting how many fields can be selected for binary output, it is just the length of the character string `writeselect`.

`writeselect`: it is used to determine which fields should be output under the control of `mplot`.

`write_isospin`: if this is `.FALSE.`, the proton and neutron contributions of a field are added up before output. Otherwise both are written.

5.17.8. Fragment number parameters

`nof`, `mnof`: number of fragments for the initialization and maximum allowed. These should really be determined in module `Fragments`; the reason for putting these here and reading `nof` early is that `nof` must be known and the arrays for fragments allocated before reading the other fragment input. Setting `mnof` to a large number is no problem since the arrays are quite small.

`r0`: nuclear radius parameter. The nuclear radius $R = r_0 A^{1/3}$ is used to compute the β and γ deformation parameters in subroutine `moments`. Units: fm, default value 1.2 fm.

5.18. Module `Static`

5.18.1. Module variables

`tdiag`: if `true`, there is a diagonalization of the Hamiltonian during the later (after the 20th) static iterations. The 20 is hard coded in `static.f90`. Default is `false`.

`tlarge`: if `true`, during the diagonalization the new wave functions are temporarily written on disk to avoid doubling the memory requirements. Default is `false`.

`maxiter` maximum number of iterations allowed.

`serr` convergence criterion. Iterations are stopped if the fluctuation in single-particle energies falls below this value (see near the end of subroutines `statichf`).

`e0dmp`,`x0dmp`: these correspond to the parameters E_0 and x_0 appearing in the damped gradient iteration of Eq. (12).

`radinx`, `radiny`, `radinz`: these are the radius parameters used in the harmonic-oscillator initialization (see subroutine `harmosc`).

`delesum`: used to sum up the changes in single-particle energies during one iteration; it is calculated in `statichf` but printed in `sinfo` so that it is a module variable.

5.18.2. Subroutine `getin_static`

This routine reads the input for the static calculation using namelist `static`. This includes the module variables of this module, but also the numbers of particles, which are input quantities only in the static mode if initialization is not done via fragment files. In the case of user initialization they are also needed to correctly allocate the fields; only the wave functions are then calculated in `user_init`. The values given in the input are overwritten by fragment data otherwise.

Thus only if `nof<=0` the input numbers are used. If `npsi` is not given in the input, the values of `nneut` and `nprot` are used for the number of wave functions, except for the pairing case, when they are computed from a formula.

The variables `charge_number` and `mass_number` are also set for this case.

5.18.3. Subroutine `init_static`

This subroutine essentially just prints the static input and then initializes the header files with their header lines. This is not included in `getin_static`, because the particle and state numbers may have been changed by fragment input.

In addition, the damping matrices are constructed by calling `setup_damping`. Finally for some Skyrme forces the effective mass is changed to account for the center-of-mass correction. This should only be used if necessary and not in dynamic calculations.

5.18.4. Subroutine `statichf`

This is the principal routine for the static iterations. It applies the gradient step repeatedly until convergence is achieved or the maximum number of iterations is reached. The following local variables are worth defining:

`sumflu` keeps track of the fluctuations (13) in single-particle energies summed over the states. It is used as the convergence criterion.

addnew, addco are simple factors with standard values 0.2 and 0.8 used for relaxation (see Step 8). They are defined as parameter variables so they can be changed easily if desired.

denerg is used as an argument to grstep to contain the relative change in energy of the single-particle state.

- **Step 1: Initialization**: if this is a restart, the number of the initial iteration is set to the value of iter+1 obtained from wffile. In this case the single-particle quantities do not have to be set to zero and orthogonalization is not necessary. If this is not a restart, the initialization is done as zeroth iteration and the first iteration number for the loop is set to one. some variables are initialized to zero and the matrix for the diagonalization hmatr is allocated. Then schmid is called for initial orthogonalization.
- **Step2: calculating densities and mean fields**: the densities are reset to zero and then in a loop over states the contributions of the single-particle states are added up. The subroutine skyrme is called to compute the mean-field components.
- **Step 3: initial gradient step**: in a loop over the single-particle wave functions the gradient step (see Section 2.4.2) is applied — see subroutine grstep. The sum of relative changes in single-particle energies and fluctuations are accumulated in delesum and sumflu. This loop is followed by the pairing calculation (which needs the single-particle energies calculated in the gradient step) and renewed orthogonalization. Then the detailed single-particle properties and energies are calculated using sp_properties and sinfo.
- **Step 4: start iteration**: this is the principal loop for the static calculation. The iteration number is printed.
- **Step 5: gradient step**: this is identical to the initial gradient step in "Step 3".
- **Step 6: diagonalization**: after 20 iterations and only if the switch tdiag is true, diagstep is called to diagonalize the single-particle Hamiltonian.
- **Step 7: pairing and orthogonalization**: these are called for the new wave functions.
- **Step 8: calculate densities and fields with relaxation**: the old density rho and kinetic energy density tau are saved in upot and bmass, which are here used purely as work arrays. Then the new densities are accumulated from the wave functions and for rho and tau they are mixed with the old densities in a ratio given by addnew and addco, if this is turned on by taddnew. skyrme is called to calculate the new fields. Then the detailed single-particle properties and energies are calculated and printed using sp_properties and sinfo.
- **Step 9: finalizing the loop**: convergence is checked by comparing sumflu per particle to serr, if it is smaller, the job terminates after writing the final wave functions. Otherwise, the wave functions are written if indicated by mrest and the loop continues.

5.18.5. *Subroutine* `grstep`

This subroutine applies the damped gradient step (12) to a wave function given as argument `psin`. Its index is `nst` and isospin `iq` — these data are needed for the construction of the Hamiltonian matrix `hmatr`. The argument `spe` $\rightarrow \epsilon$ represents the single-particle energy which is used a an energy shift in the calculation.

The work is done in the following steps:

- **Step 1**: apply $\hat{h} - \epsilon$ to the wave function `psin` to obtain `ps1`.
- **Step 2**: calculate the diagonal matrix element of $\hat{h}$

  $$\texttt{xnormb} = \langle\texttt{psin}|\texttt{ps1}\rangle = \langle\texttt{psin}|\hat{h}|\texttt{psin}\rangle$$

  and the squared norm of `psin` (in `xnorm`). The expression `xnormb/xnorm` thus corresponds to the expectation value $\|\hat{h}\|$ in `psin`.

  Then the matrix elements of $\hat{h}$ with all other states of the same isospin are calculated and inserted into `hmatr`; in the diagonal matrix elements the energy shift ϵ is added back.
- **Step 3**: for the calculation of the fluctuation in the single-particle energy, the quantity

  $$\texttt{exph2} = \langle\texttt{psin}|\hat{h}^2|\texttt{psin}\rangle$$

  is used to compute

  $$\texttt{sp\_efluct1} = \sqrt{\|\hat{h}^2\| - \|\hat{h}\|^2}$$

  and the squared norm

  $$\texttt{varh2} = \|\hat{h}\,\texttt{psin}\|^2$$

  similarly for the second fluctuation measure

  $$\texttt{sp\_efluct2} = \sqrt{\|\hat{h}\,\texttt{psin}\|^2/\|\texttt{psin}\|^2 - \|\hat{h}\|^2}.$$

  They are calculated only for time steps with output turned on.
- **Step 4**: now the damping is performed. We first compute

  $$|\texttt{ps1}\rangle - \texttt{xnormb}\,|\texttt{psin}\rangle = \left(\hat{h} - \langle\texttt{psin}|\hat{h}|\texttt{psin}\rangle\right)\,|\texttt{psin}\rangle$$

  replacing `ps1`, on which then the real damping operator acts. If `FFT` is being used for the derivatives, we use the routine `laplace` from module `levels` to compute

  $$\frac{x_{0\text{dmp}}}{E_{0\text{inv}} + \hat{t}}\,|\texttt{ps1}\rangle.$$

  If derivatives are to done by matrices, the damping matrices `cdmpx` etc. are used (see subroutine `setdmc` in module `Grids`. The factors `x0dmp` and `e0dmp` have to be manipulated a bit in this case. Finally we subtract this result multiplied by `x0act` from the original wave function to get the damped one.
- **Step 5**: the single-particle energy is calculated from its new expectation value with the energy shift restore, and the comparison with the initial value yields the relative change `denerg`, which is passed back to the caller.

5.18.6. Subroutine `diagstep`

This subroutine performs a diagonalization of the single-particle Hamiltonian using the LAPACK routine `ZHBEVD`. This is of course done separately for protons and neutrons indexed by `iq`. The matrix `hmatr` is produced in `grstep`.

The details here depend on the requirements of this routine, which is written in Fortran-77 and so has a complicated calling sequence with many arguments used for intermediate storage.

We therefore do not give excessive detail here but summarize the main points, which should be easy to analyze. The steps are:

- **Step 1**: the matrix is copied into lower-diagonal form into array `hmatr_lin`. The `ZHBEVD` is called, which leaves as main results the vector of eigenvalues `eigen` and the unitary matrix `unitary` describing the transformation from the original states to the diagonalized ones. The latter matrix is also stored in lower-diagonal form.
- **Step 2: transform states**: the matrix `unitary` is used to form the appropriate linear combinations of the original single-particle states. If `tlarge` is true, each state if formed independently and written onto a scratch file, otherwise an intermediate array `ps1`, dimensioned to hold either only protons or only neutrons (number of states through the argument `nlin`), is allocated to receive the new states, which are then copied back into `psi`.

5.18.7. Subroutine `sinfo`

This subroutine computes the data that are not needed for the calculation itself, but only for informative output and writes them onto the appropriate output files.

First `moments`, `integ_energy`, and `sum_energy` are called to compute the relevant physical quantities. Output lines are added to `converfile`, `dipolesfile`, `momentafile`, and `spinfile`. Information on the current energy `ehf`, the total kinetic energy `tke`, the relative change in energy over the last iteration `delesum`, the fluctuations in single-particle energy, and the energy corrections are printed on standard output.

At intervals of `mplot` iterations the density printer plot is produced and the `*.tdd` file containing the densities written.

Finally, on standard output more detailed output is given: an overview of the different contributions to the energy, a list of single-particle state properties, and a listing of various moments using `moment_print`.

5.19. Module `Trivial`

This module contains some basic calculations with wave functions and densities, which are similar enough to be grouped together.

5.19.1. Subroutines `cmulx`, `cmuly`, *and* `cmulz`

The routines do a complex matrix multiplication on a wave function along one of the Cartesian directions, which we denote as `[x,y,z]` to indicate the possible choices. The first argument `[x,y,z]mat` is the matrix dimensioned as a square matrix with the number of points in the direction affected. The second argument `pinn` is the input wave function, and the third one, `pout`, the output wave function. The result of the multiplication is added to the output wave function if the final argument, `ifadd`, is nonzero. This allows accumulating results but is not used in the present code. If `ifadd` is zero, the output wave function is cleared to zeroes and thus is simply the result of the matrix multiplication.

The operation carried out here includes a loop over the spin index.

The effective calculation is, e. g., for the x-direction

$$\texttt{pout(i,j,k,s)} = \sum_{i'=1}^{nx} \texttt{xmat(i,i}'\texttt{)pin(i}'\texttt{,j,k,s)}$$

for all values of `i,j,k,s`.

Not that for the z-direction an explicit `DO`-loop is programmed instead of the `FORALL` used in the other cases. This is because the compiler at some point did not optimize well. This should be reexamined in the future.

5.19.2. Function `rpsnorm`

This function with one argument calculates the norm of a wave function, i. e.,

$$|\psi| = \sum_{s=\pm\frac{1}{2}} \int |\psi(\vec{r},s)|^2 \,\mathrm{d}^3r \to \texttt{wxyz} \sum_{s=1}^{2} \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} |\texttt{psi(i,j,k,s)}|^2,$$

with `wxyz` the volume element.

This could be calculated with the built-in `SCALAR_PRODUCT` function, but this was found to be less efficient. Again, future compilers may change that.

5.19.3. Function `overlap`

This calculates the overlap of two wave functions psi_L and psi_R given as arguments `pl` and `pr`. This is defined as

$$\begin{aligned}\langle\psi_L|\psi_R\rangle &= \sum_{s=\pm\frac{1}{2}} \int \psi_L^*(\vec{r})\,\psi_R(\vec{r})\,\mathrm{d}^3r \\ &\to \texttt{wxyz} \sum_{s=1}^{2} \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} \texttt{CONJG(pl(i,j,k,s))pr(i,j,k,s)}.\end{aligned}$$

This also might be replaced by `SCALAR_PRODUCT` eventually.

5.19.4. Subroutines `rmulx`*,* `rmuly`*, and* `rmulz`

The subroutines work quite analogously to `cmulx` etc., with two important differences:

- The array `finn` operated on is not a complex wave function, but a real field in space as is the result `fout`. Thus the arithmetic is real and there is no summation over spin.
- The final argument `ifadd` has an additional function depending on its sign. If it is zero, `fout` is set to zero before the calculation. If it is *positive or zero*, the result is *added* to `pout`, otherwise it is *subtracted.*

This facility is used in the code to calculate the curl of vector fields, for example, by applying the derivative matrices to the Cartesian components of the field and adding and subtracting appropriately.

5.20. Module `Twobody`

This module contains the code to analyze the final (and also, though less useful) initial stage of a heavy-ion reaction. It is applicable only to the dynamic calculations and only if there are essentially two separated fragments. It calculates the fragment masses and charges, their distance, the relative motion kinetic energy, angular momentum, and the scattering angle.

The analysis assumes that the two fragments are separated by a region of noticeably lower density, tries to find the line connecting their centers of mass and then divides up space by a plane perpendicular to this line. Of necessity the results are not of high precision, but tend to be useful nevertheless.

The user should be critical and always make sure that the real situation is a two-fragment one before accepting the results of these routines. It is assumed that the reaction plane is the (x, z) plane. The code could be generalized to a three-dimensional situation, if necessary, but usually the assumption of a fixed scattering plane should not be a problem. If the initial nuclei have nonzero internal angular momentum, e. g., the reaction plane can rotate and then a more general analysis cannot be avoided.

5.20.1. Module variables

`roft`: separation distance between fragments in fm.

`rdot`: relative-motion velocity in units of c.

`xmin`, `zmin` (private): coordinates of the point in the (x, z)-plane where minimum density is found between the fragments.

`slope`, `slold` (private): Slope of the line connecting the fragment centers-of-mass. `slold` is the value from the previous time step or iteration where the 2-body analysis was last performed.

`bb` (private): intercept of the line. The line is thus given as `z=bb+slope*x`.

`centerx`, `centerz` (both private): coordinates of the two fragment centers of mass.

`istwobody`: logical, indicates whether this is a two-body case (`TRUE`) or not.

`vacuum`: parameter indicating the limiting density below which vacuum is assumed. This value is not critical, since it is only used in looking for “empty” regions between the nuclei.

The two-body properties finally calculated are the following:

`xmu` μ: the reduced mass in MeV.

`vxx, vzz` : x- and z-velocities of the fragments, calculated by differencing the center-of-mass positions.

`tke2body` : total kinetic energy in MeV.

`tketot` : relative-motion kinetic energy in MeV.

`roft` : R: distance between the fragments in fm.

`rdot` : time-derivative of the separation distance, units of c.

`teti` : present scattering angle.

`tdotc` : ω: time derivative of scattering angle. Units c/fm.

`xlf` : angular momentum of relative motion in $\hbar$. It is calculated assuming two point bodies via $\mu R^2\omega$.

`xcoul` : Coulomb energy, calculated from point charges.

`xcent` : centrifugal energy, based on angular momentum and distance.

`ecmf` : final relative motion energy after extrapolated to infinite separation.

5.20.2. Subroutine `twobody_case`

This routine tries to find a separation of the system into two fragments and determine their properties as well as those of the relative motion. It keeps track of previous analysis results, since the 2-body properties are expected to change slowly and this makes the analysis easier. Also, the motion of the fragments is determined by time-differencing.

The parameter `xdt` is the current time step. It cannot be imported directly from module `Dynamic` since that would lead to a circular module dependence.

It uses the following steps:

- **Step 1**: the results of the last analysis are partially saved. This includes the positions of the fragments, their distance, the slope of the connecting line, and its angle.

- **Step 2:** the fragment division is sought in an iterative process with (at present) a fixed iteration limit of 10. The reason for the iterations is that the connecting line, which determines the dividing plane, which is orthogonal to it at the dividing point, should link the centers-of-mass, but these can be calculated only assuming a dividing plane. There is thus a self-consistency problem which as usual is solved iteratively. During the iterations the new value of the center of mass is kept in `centerx` and `centerz`, while the previous one is in `centx` and `centz`. Each iteration has several steps:

  1. The connecting line (`slope` and intercept `bb`) is determined in one of two ways: if this is the first iteration, subroutine getslope is called to calculate the slope from the quadrupole tensor; the intercept is then calculated assuming that the line passes through the center-of-mass (which it should do in principle, but it might be off in practice). For the later iterations the line is calculated directly from the two centers of mass.
  2. Next function `divpoint` is called to find out whether this is indeed a two-body situation. It also returns the midpoint between the fragments in `xmin` and `zmin`. This result is not used immediately, since it might be that shifting the connecting line could change this situation.
  3. The slope of the line is now rotated by 90° to get the line defining the dividing plane in the (x, z)-plane. This has `slopev` and `bb` as slope and intercept.

4. Now in a simple loop integrals are done separately over the two regions, summing up `charge`, `mass`, and `center` of mass for the two fragments. The assignment to the fragments is recognized by checking whether the point is above or below the dividing line (variable `diff`). Note that the y-component of the centers of mass is not calculated.
5. Finally the iteration process is stopped if no two-body situation is found or the center-of-mass vectors have converged.

- **Step 3:** The two-body analysis is now assumed to be complete and the physical quantities are evaluated. These are defined above in the list of module variables. The following calculation concerns the final scattering angle `tets` extrapolated to infinity. It can be understood using the Rutherford trajectories.

5.20.3. Subroutine `getslope`

This subroutine calculates the slope of the line determined by the eigenvector of largest quadrupole moment in the (x, z)-plane. The first loop sums up the quadrupole tensor, which is dimensioned `q2(3,3)` but of which the index 2 is not actually necessary, since the y-direction is not involved. The definition is kept three-dimensional to reduce confusion and make later generalization easier.

The two-dimensional eigenvalue problem for `q2` has the secular equation (remember $q_{31} = q_{13}$:

$$(q_{11} - \lambda)(q_{33} - \lambda) - q_{13}^2 = 0,$$

with λ the eigenvalue. For the larger eigenvalue we get

$$\lambda = \frac{1}{2}\left(q_{11} + q_{33} + \sqrt{(q_{11} - q_{33})^2 - 4q_{13}^2}\right),$$

and solving the equation $q_{13}x + (q_{33} - \lambda)z = 0$ for z yields

$$\begin{aligned} z &= \frac{q_{13}}{\lambda - q_{33}}\,x \\ &= \frac{q_{13}}{\frac{1}{2}\left(q_{11} - q_{33} + \sqrt{(q_{11} - q_{33})^2 - 4q_{13}^2}\right)}. \end{aligned}$$

The code calls the denominator `denom` and makes sure no division by zero happens (this could happen for a spherical distribution). The resulting slope is returned in the module variable `slope`.

5.20.4. Function `divpoint`

The function divpoint is a helper routine. It examines the line determined by `slope` and intercept `bb` and finds the point `(xmin,zmin)` which is in the center of the void between the two fragments, returning .true. if this is possible.

To this end it looks at the behavior of the densities along this line. Since the line has no relation to the numerical grid, this is not trivial.

- First loop: Essentially it looks through the (x, z)-plane to find points closer than half a grid spacing to the desired line with equation `z=slope*x+bb` (logical variable `online`). If the slope is larger than one, i. e., if the nuclei are separating predominantly in the x-direction, we need to take the equation `x=(z-bb)/slope` instead to get better resolution. To make the result monotonic along the line, the do loop in z runs backward for negative slopes. The points found are collected in index vectors `ixl`, `izl` with accompanying densities `rhol` stored in arrays of length `il`.

- Second loop: now the number of fragments `nf` is counted by examining this one-dimensional density curve, looking for disconnected density humps above `vacuum` density. It is also recorded where the "vacuum" region starts and ends in variables `n1` and `n2`. The logic is as follows:

  1. The logical variables `in_vacuum` keeps track of whether the search is in a vacuum region at the moment or not. It starts as `TRUE`.
  2. Go to the next point. If its density is above `vacuum`, and we are in the vacuum, a new fragment is starting and we increase the number of fragments `nf` by 1. If it becomes bigger than 2, exit, because there are three or more fragments. If it is now 2, record the starting index for the second fragment in `n2`.
  3. If the density is below `vacuum` and we are not in vacuum, a fragment is being ended. If it is the first fragment, we record this index in `n1`.

- Final processing At this point we expect a two-fragment situation if `nf=2` and in this case the void region between the fragments extends from `n1` to `n2`, which are indices into arrays `ix1` and `iz1` giving the position in the (x, z)-plane. The code calculates the midpoint between the two positions and returns `TRUE` in this case.

5.21. Module `User`

This is included as a place to insert arbitrary user initialization of the wave functions. It really does the same job as subroutine `harmosc`, which is a relatively complicated example. For this reason a sample user initialization is also provided in the file `user_sample.f90`. It produces Gaussian wave functions for three alpha-particle-like nuclei separated by a distance `d` and with radii `r`.

The only routine that has to be defined is `user_init` , but for more complicated initializations there can be any number of additional procedures accompanying it in `user.f90`.

The setup assumes that the relevant particle numbers `nneut` and `nprot` and numbers of states `npmin`, `npsi`, and `nstmax` are set correctly using the static input. In this case we assume `nprot=6`, `nneut=6`, `npmin=1,7`, `npsi=6,12`, and `nstmax=12`. Note that the occupation numbers `wocc` still have to be set explicitly, in this case they are all unity.

The routine then reads the parameters for the setup from namelist `user`. This namelist can be used to read anything desired. If there is no user initialization, it is simply omitted from the input file.

Now there is a loop over center positions with index `ic`. For each of them, the appropriate Gaussian is calculated and put into wave function #`ic` in the spin-up component; the spin-down component is set to zero. Then the Gaussian is copied into index position `ic+3`, spin-down component, and finally the complete wave functions are copied to the proton indices by adding 6.

There is no need to orthonormalize the wave functions, since `schmid` is called before the static iterations are started. User initialization for the dynamic case does not appear useful; it could be easily done without modifying the code by running one static iteration and using the wave-function file generated at the beginning to initialize the dynamic calculation.

6. Input description

All the input is through `NAMELIST` and many variables have default values. **The `NAMELIST`s should be in this file in the order in which they are described here, any `NAMELIST` not used for a particular job may be omitted or left in the input file, in which case it is ignored**. The input is from a file called `for005`, so the input data have to be produced with an editor. If the large output listing is to go into `output`, the code should be run using, e. g.,

```
./sky3d.seq␣>␣output
```

The reason for not using redirected input is that in most MPI implementations for an input through "`< for005`" is passed only to node 0, while all nodes can read the same file in parallel using an explicit `OPEN` statement.

6.1. Namelist `files`

This `NAMELIST` contains names for the files used in the code. They are defined in module `Params` and are:

`wffile`: file to contain the static single-particle wave functions plus some additional data. This can be used for fragment initialization or for restarting a job. Default: `'none'`, i. e., nothing is written.

`converfile`: contains convergence information for the static calculation. Default: `conver.res`.

`monopolesfile`: contains moment values of monopole type. Default: `monopoles.res`.

`dipolesfile`: contains moment values of dipole type. Default: `dipoles.res`.

`quadrupolesfile`: contains moment values of quadrupole type. Default: `quadrupoles.res`.

`momentafile`: contains components of the total momentum. Default: `momenta.res`.

`energiesfile`: energy data for time-dependent calculations. Default: `energies.res`.

`spinfile`: time-dependent total, orbital, and spin angular-momentum data as three-dimensional vectors.

`extfieldfile`: time dependence of expectation value of the external field.

6.2. Namelist `force`

This defines the Skyrme force to be used. In most cases it should just uses the input values:

`name` : the name of the force, referring to the predefined forces in `forces.data`.

`pairing` : the type of pairing, at present either `NONE` for no pairing, `VDI` for the volume-delta pairing, or `DDDI` for density-dependent delta pairing. The pairing parameters are included in the force definition. **Note that the pairing type must be written in upper case.**

`turnoff_zpe` : if this is input as `.TRUE.`, the zero-point energy correction is turned off independent of the setting of `zpe` in the force definition. Its default value is `.FALSE.`.

There is also the possibility for inputting a user-defined force; this is described in detail with module `Forces`, see Section 5.7.

6.3. Namelist `main`

This contains general variables applicable to both static and dynamic mode. They are mostly defined in module `Params`.

`tcoul:` determines whether the Coulomb field should be included. Default is `true`.

`trestart:` if `true`, restarts the calculation from `wffile`. Default is `false`.

`tfft:` if `true`, the derivatives of the wave functions, but not of the densities, are done directly through FFT. Otherwise matrix multiplication is used, but with the matrix also obtained from FFT. Default is `true`.

`mprint:` control for printer output. If `mprint` is greater than zero, more detailed output is produced every `mprint` iterations or time steps on standard output.

`mplot:` if `mplot` is greater than zero, a printer plot is produced and the densities are dumped every `mplot` time steps or iterations. Default is 0.

`mrest`: if greater than zero, a `wffile` is produced every `mrest` iteration or time step. Default is 0.

`writeselect` : selects the output of densities by giving a string of characters choosing them (see subroutine `write_densities` for details. Default is `'r'`, i. e., only the density is written.

`write_isospin` : determines whether the densities should be output isospin-summed (`false`) or separately for neutrons and protons (`true`). Default is `false`.

`imode` : selects a static `imode=1` or dynamic `imode=2` calculation.

`nof` : (number of fragments) selects the initialization. `nof=0`: initialization from harmonic oscillator, only for the static case; `nof<0`: user-defined initialization by subroutine `init_user` in module `User`; `nof>0`: initialization from fragment data as determined in `NAMELIST fragments`.

`r0`: nuclear radius parameter. The nuclear radius $R = r_0 A^{1/3}$ is used to compute the β and γ deformation parameters in subroutine `moments`. Units: fm, default value 1.2 fm.

6.4. Namelist `grid`

This defines the properties of the numerical grid.

`nx, ny, nz` : number of grid points in the three Cartesian directions. They must be even numbers.

`dx, dy, dz` : spacing between grid points in fm. If only `dx` is given in the input, all three grid spacings become equal. The grid positions are then set up to be symmetric with the coordinate zero centrally between point number `nx/2` and `nx/2+1`.

`periodic` : chooses a periodic (`true`) or isolated (`false`) system.

6.5. Namelist `static`

These input variables control the static calculations.

`tdiag:` if `true`, there is a diagonalization of the Hamiltonian during the later (after the 20th) static iterations. This 20 is hard coded in `static.f90`. Default is `false`.

`tlarge:` if `true`, during the diagonalization the new wave functions are temporarily written on disk to avoid doubling the memory requirements. Default is `false`.

`nneut, nprot:` The numbers of neutrons and protons in the nucleus. These are used for the harmonic-oscillator and user initialization.

`npsi:` the numbers of neutron (`npsi(1)`) and proton (`npsi(2)`) wave functions actually used including unfilled orbitals. Again, useful only for harmonic-oscillator or user initialization.

`radinx, radiny, radinz:` the radius parameters of the harmonic oscillator in the three Cartesian directions, in fm.

`e0dmp:` the damping parameter. For its use see subroutine `setdmc`. The default value is 100 MeV.

`x0dmp:` parameters controlling the relaxation. The default value is 0.2. In special cases it may be desirable to change this to accelerate convergence.

`serr:` this parameter is used for a convergence check. If the sum of fluctuations in the single-particle energies, `sumflu` goes below this value, the calculation stops. A typical value is `1.E-5`, but for heavier systems and with pairing this may be too demanding.

6.6. Namelist `dynamic`

These are variables controlling the dynamic (TDHF) calculation.

`nt` : number of time steps to be run.

`dt` : the time step in fm/c. A standard value is of the order of 0.2 to 0.3 fm/c, it depends somewhat on the value of `mxpact`. If the combination of these two is not good enough, the calculation becomes unstable after some time, in the sense that the norm of the wave functions and the energy drift off and can diverge (see Sect. 2.10).

`mxpact` : the order of expansion for the exponential time-development operator. The predictor (trial) step calculation uses `mxpact/2` as the order. For more information see Sect. 2.10.

`rsep` : termination condition. If the final state in a two-body reaction is also of two-body character, the calculation is terminated as soon as the separation distance exceeds `rsep`. Units: fm. No default. The purpose of this variable is to prevent the calulation of continuing into meaningless configurations, like crossing of the boundary.

`texternal` : indicates that an external perturbing field is used. In this case the namelist `extern` must be present. Default: `false`.

6.7. Namelist `extern`

The variables read here describe the external field that is applied to get the nucleus into a collective vibration. Details can be found in the description of module `External`. It is read only if the parameter `texternal` read in namelist `dynamic` is true.

`ipulse` : the type of pulse applied. For `ipulse=0` the wave function is multiplied with a phase factor that produces an initial excitation. For `ipulse=1` a Gaussian time dependence is used, for `ipulse=2` a $\cos^2$ one. Default: 0. Details are given in Eqs. (9d) and (9e).

`isoext` : isospin character of the excitation. If this is zero, protons and neutrons are exited in the same way. For a value of 1, they behave oppositely but with a coupling that leaves the center-of-mass invariant. Default: 0.

`tau0`, `taut` : time at which the excitation field reaches its maximum, and width of the pulse. No defaults.

`omega` : if this is nonzero, the time-dependence of the external field gets an additional cosine factor with frequency `omega`.

`radext`, `widext` : radius and width of a Woods-Saxon-type cutoff factor in radius for the external field. Defaults: 100 fm and 1 fm, which practically implies no damping. Definition in Eq. (9b).

`amplq0` : amplitude for quadrupole excitation of the Q_{20} type. Defined as usual with respect to the z-axis.

6.8. Namelist `fragments`

The variables in this namelist control fragment initialization for the case of `nof>0`. Most quantities are dimensioned for the fragments and we indicate this by index "`i`" in the following.

`filename(i)` : the name of the file containing the wave functions of fragment `i`.

`fcent(1:3,i)` : initial position of fragment `i` given as three Cartesian coordinate values in fm. The position must be such that the complete fragment grid fits inside the new computational grid.

`fix_boost` : used only for the two-fragment case. if this logical variable is `TRUE`, the initial velocities are calculated from the `fboost` values; otherwise from the relative motion quantities `ecm` and `b`.

`fboost(1:3,i)` : the initial boost of the fragment in the three Cartesian directions. It is given as the total kinetic energy in each direction in MeV, with the sign indicating positive or negative direction. Thus `SUM(ABS(fboost(:,i)))` is the total kinetic energy of fragment `i`.

`ecm`, `b` : center-of-mass kinetic energy in MeV and impact parameter in fm. Used only if `fix_boost` is `FALSE`. These are the values at infinite distance and are corrected using Rutherford trajectories (assuming spherical nuclei) for initialization at the finite distance given by the `fcent` coordinates.

6.9. Namelist `user`

This namelist is read only if needed for user initialization (see module `User`). Its contents depend on the specific user initialization and the only thing to be said here is that it should appear last in the input file. Since the namelist is defined and used only in module `User`, its name can also be changed arbitrarily, of course.

7. Output description

7.1. Output and analysis

The code produces a number of output files containing various pieces of information. The bulky observables, such as densities or currents, are selectively output at certain time steps into special binary output files *nnnnnn*`.tdd`, where *nnnnnn* indicates the iteration or time step number. These files can then be used for further analysis or converted to be used as input in visualization codes. Examples of this are found among the utility codes provided.

The complete set of wave functions is saved at regular intervals of `mrest` iterations or time steps. Because this leads to large storage requirements, only the last such file in a run is kept. It can be used for restarting the calculation or for inputting fragment wave functions for initializing another calculation.

In `MPI` mode the wave functions are distributed over several files, each containing only those present on a specific processor. An additional header file contains the remaining information and can be used to read the wave functions even on a different processor configuration.

Aside from these binary files there are a number of text files. The `*.res` files contain one line for each time step or iteration where output is triggered according to the value of `mprint`. There is an explanatory header line in these that has a leading '#', so that is treated as a comment by `gnuplot` — thus `gnuplot` can be used immediately to plot the behavior of any one column of numbers, i. e., the dependence of a physical quantity on iteration or time. In addition more complicated output is printed on standard output, which can be redirected into a file using shell redirection.

The names of the output files can be adjusted using input variables as listed in Section 5.17.3, so that the files are here denoted by the default names given there. The `*.res` files are relatively small, so that no mechanism was implemented to suppress them.

7.2. File `conver.res`

This is produced in `sinfo` only in the static calculation and its purpose is to give a quick impression of the convergence behavior. The numbers given in each line are the iteration count, the total energy in MeV, the relative change in energy from one iteration to the next, the average uncertainties in the single-particle energies `efluct1` and `efluct2`, the root-mean-square radius in fm and finally the deformation parameters β and γ (see Section 2.6.1). The latter give an impression as to where the nuclear shape is ending up.

For the judging of convergence, the `efluct` values are more important than the change in total energy, since the energy can remain constant while the wave functions still change considerably.

7.3. File `monopoles.res`

At present this file is generated in subroutine `moment_shortprint`, but only in the dynamic mode. It contains the time, the neutron, proton, and total root-mean-square radii, and the difference of neutron minus proton root-mean-square radii.

7.4. File `dipoles.res`

This file is produced both in the static and dynamic calculations in subroutines `sinfo` and `tinfo`. It contains the iteration or time step number followed by the three components of the center of mass vector $\vec{R}$ and those of the difference of proton minus neutron center-of-mass vectors $\vec{R}^{(T=1)}$, both in fm, for the definition see Eq. (21a). The first of these is useful as a check to see whether the center of mass drifts off during the calculation, while the second vector may be useful to look at proton vs. neutron vibrations.

7.5. File `quadrupoles.res`

This is also generated in `moment_shortprint` and thus only in dynamic mode. It contains the Cartesian quadrupole moments (21b) for neutrons, protons, and the full mass distribution followed by the expectation values of x^2, y^2, and z^2 for neutrons and protons, all in fm^2.

7.6. File `energies.res`

This is the important monitoring file for the dynamic calculation. It is written in subroutine `tinfo` and each line contains the simulation time, the number of neutrons and protons in the system (these should be constant, so this is a stability check), the total energy (again, this should be conserved), the total kinetic energy, and finally the collective energy `ecoll` separately for neutrons and protons — see Eq. (33. Units for the energies are all MeV.

7.7. Standard output

This contains all the additional information that in most cases is not needed directly for further processing in, e. g., graphics programs. If it should be found necessary to utilize some data from this file, it is in most cases easy to use `grep` or a scripting language like `Perl` or `Python` to extract the necessary data. Of course the code can also be modified to produce additional output files.

The initial part of the output essentially echoes all the data from the NAMELISTs in tabular form, to enable checking the correctness of input data. In the case of fragment initialization this is more involved and is discussed below for the dynamic case, since it is not so common for static calculations.

In general the layout of the information is compact with sequences of "*" characters to provide separation between input groups as some guidance for the eye.

7.7.1. Static calculation

The code first prints the current iteration number. Iteration "0" refers to the state before iterations are started, for the later iteration numbers, the information refers to the end of the iteration.

The overview of the various energy contributions is printed: the first part is similar to what is in file `conver.res`, while a second list shows the energies calculated from the density functional and split up for the various contributions.

Next there is a simple printer plot of the density distribution in the $(x - z)$-plane. This is often quite helpful, since it shows what is going on in the calculation without the need to start a graphics program, which requires converting the data first.

Next there is a listing of single-particle states. For each state this shows its parity, occupation probability `wocc` (which is called v^2, as it is interesting mostly in the pairing case), the energy fluctuations `sp_efluct1` and `sp_efluct2`, the norm, the kinetic and total energies of the state, and finally the expectation values of the three components of the orbital and spin angular momentum, respectively.

Finally a summary of some integrated quantities is given, separately for neutrons, protons, and all nucleons: the particle number, the root-mean-square radius, quadrupole moment, and the average of the coordinates squared, followed by the center-of-mass components.

Then iterations continue and only one line is printed for each, as this may be quite slow and it is important to be able to check progress while the code is running. After `mprint` iterations the detailed information is repeated.

7.7.2. Dynamic calculation

After echoing the parameters for the dynamic calculation, the fragment definitions are given and all the resulting information is printed: the computed boost values in case of twobody initialization, the properties of the single-particle states read in, including which index in the fragment file is transferred to which index in the total set of wave functions.

In case of an external field, the input data is also echoed in a detailed form.

The time stepping starts and detailed output is produced every `mprint` steps at the end of the time step. Much of it is similar to the static case, so only the differences are pointed out.

Because in the dynamic case the situation can have a general three-dimensional character, the full information on the quadrupole tensor (21b) is printed, separately for the neutron, proton, and total mass distributions. The three eigenvalues and associated normalized eigenvectors are given, followed by Cartesian and polar deformation parameters a_0, a_2, β, and γ, as defined in Section 2.6.1.

The separation of the two fragments and its time derivative is printed next **and repeated every time step**, as examining these quantities is meaningful only with more frequent sampling.

The energy information is shortened by omitting the quantities not of interest in the dynamic case. After the printer plot the results of the two-body analysis are given (for the meaning of the various quantities see Section 5.20). The Section on "collision kinematics" shows the mass, charge, position, and kinetic energy of the two fragments. **It should be kept in mind that the two-body analysis is only valid if the reaction plane is the $(x-z)$-plane and the results printed may not be useful if the physical situation is not of two-body nature but the code does not recognize that.**

The single-particle property list and the integrated quantities are as in the static case, but the energy fluctuations are omitted.

The next time step is then indicated and the fragment separation data are printed for every time step until after `mprint` steps the full output recurs.

8. Utilities

A set of short programs is designed to help with further processing of output from the code. The currently available set is described here.

Most of these routines contain a loop to input a file name from the terminal. If it is desired to do this in a loop over a set of files, a simple trick can be used: generate a list of file names using, e. g.,

```
ls -1 *.tdd > list
```

and then execute the program with "list" as input, e. g.,

```
./fileinfo < list
```

For `Tdhf2Silo` a script `convert` is provided that handles this (see below). It can easily be adapted to the other utilities and must be stored in the same directory as the executable utility program itself in order for the `dirname` command to work properly.

8.1. Fileinfo

This is a short program to print information about binary files generated by Sky3D. It takes the name of either a `*.tdd` or a wave function file as input and prints out essentially all the information contained in the header. It can be compiled simply by executing

```
gfortran -o fileinfo fileinfo.f90
```

8.2. Inertia

This program calculates the tensor of inertia relative to the center of mass from the density distribution. It is intended as an example of an analysis code reading `*.tdd` files and doing some computation, which can be used as a model for doing similar things. It illustrates looking for the desired field in the file and taking into account whether it is stored as a total density or isospin-separated. Being given a filename as input, it reads the density and calculates the inertia tensor to print all its 9 components. Compile it using

```
gfortran -o Inertia Inertia.f90
```

8.3. Cuts

This utility reads the density from a file *nnnnnn*`.tdd` file and produces output files named *nnnnnn*`rxy.tdd`, *nnnnnn*`rxz.tdd`, and *nnnnnn*`ryz.tdd`, which contains two-dimensional cuts through the system in the (x, y), (x, z), and (y, z) plane, respectively. The cuts are evaluated at the origin for the third coordinate by averaging the two neighboring planes.

These data files are written in such a format that they can be read by `gnuplot` for use in its commands for 2-dimensional plotting.

This program is intended again as a template that can be modified for other applications.

8.4. Overlap

This is a code to calculate the overlap of two Slater determinants. Given the names of two wave function files (which must contain compatible data: dimensions, force, etc.) it reads the wave functions, generates the matrices of overlaps between one set and the other separately for neutrons and protons, and then calculates the determinant of each, which is the overlap between the two Slater determinants. It prints some summary information: distance between the centers of mass of each set, minimum and maximum diagonal elements, maximum absolute value of off-diagonal elements, and finally the overlaps for protons and neutrons as well as their product.

This code uses subroutines from `LINPACK` (stored at `NETLIB.ORG`), which are included in the file `det.f` with appropriate copyright. It can be compiled using

```
gfortran -o overlap overlap.f90 det.f.
```

8.5. Tdhf2Silo

This program is quite complicated. It reads a set of `*.tdd` files and converts them into `Silo` files. `Silo` is a library for handling scientific datasets developed at Lawrence Livermore National Laboratory (`https://wci.llnl.gov/codes/silo/index.html`). This is the most appropriate library to use in conjunction with the LLNL graphics visualization tool `VisIt` (`https://wci.llnl.gov/codes/visit/home.html`), which was found to be highly suitable for plotting `Sky3D` results and producing movies. It should be noted that a copy of the included file `silo.inc` is provided, which defines symbolic names for the various parameters used in the library calls. This file is from Silo version 4.9. **Older versions of this file may cause problems as they use fixed-format Fortran style; the present version of `silo.inc` should, however, also work with older versions of the library.**

The conversion code is quite flexible in that it decides what to produce for the different field types: isospin-summed or not, vector or scalar. They are given appropriate names for `Silo` with suffixes p and n for protons and neutrons, and x, y, z for the vector components. In the case of vector fields a variable containing the vector definition is also written so that the field can be plotted immediately as a vector field in `VisIt`.

If the user wants another dataset handling method, the code should be readily adaptable to other libraries. `VisIt` itself has many ways of importing data, but of course there are also alternative 3D visualization systems.

9. Running the code

9.1. Compilation and linking

To produce executable files the code comes with several `Makefile`s. The standard `Makefile` produces a sequential code `sky3d.seq`, the file `Makefile.openmp` a parallel code using `OpenMP`, while `Makefile.mpi` should produce an `MPI` distributed system code.

The `Makefile`s are written for the `gfortran` compiler and the commands and options must be adapted if other compilers are used. **The user may also have to modify the library names and execution of the code under MPI will require consulting the local documentation or system administrator.**

No attempt was made to select the compiler and linker options optimally for speed, since experience has shown that optimization at the cutting edge is highly time-dependent. Thus users should do some speed tests before embarking on major calculations.

9.2. External libraries needed

The `LAPACK` library is used in the code to supply the routines `ZHBEVD` and `DSYEV`. `LAPACK` should be installed in most scientific computing centers; if not, the files can be obtained from `www.netlib.org` and just be added as additional source files to the code. Note that a complete set of routines called by these two subroutines must be downloaded.

The other external routine library that is used is `FFTW3`. Again, it will be preinstalled in most systems. If not, there are two possibilities:

1. Download the source code from `www.fftw.org` and compile the library yourself. In our experience this worked smoothly. The generated library can be installed in a system library directory or kept in a user account. In the latter case the use of `-lfftw3` in the makefiles does not work anymore and the full path name of the library file must be given.
2. Replace it by another FFT routine. This requires quite a bit of work: `FFTW` organizes its calls around "plans", which describe a set of operations to be done on the three-dimensional arrays. In `init_fft` quite sophisticated plans are set up to, for example, transform in the y-direction for all x- and z-values. This means that all calls to subroutines beginning with `dfftw` have to be examined and possibly replaced by loops over one-dimensional FFT transforms. This should be relatively straightforward, but there are two more important points to consider: 1) normalization differs between FFT codes. For `FFTW` transformation followed by inverse transformation multiplies the original data by `nx*ny*nz` and this factor is taken into account in several places. 2) For the non-periodic case the Fourier transform in the Coulomb solver uses doubled dimensions in all three directions. Some FFT codes have an initialization that sets up the transformation factors depending on the dimension; in such cases the initialization may have to be repeated.

9.3. Running with `OPENMP`

The `OPENMP` version can be compiled using the file `Makefile.openmp`, which produces an executable `sky3d.omp`. The main difference to the sequential makefile is the addition of an openmp compiler option. Since this depends on the compiler used, it may have to be modified. For `gfortran` the option is `-fopenmp`, while for Intel Fortran it is simply `-openmp`.

For controlling the running of the code the user should set the environment variable `OMP_NUM_THREADS` to the number of parallel threads to be used (usually the number of processors). In addition it may be necessary to set `OMP_STACKSIZE`. The two parallel loops in `dynamichf` need to store all the density fields in parallel, and the second loop adds `ps4` to that. Taking into account vector fields and isospin, a total of 24 three-dimensional `COMPLEX(8)` fields need to be stored, amounting to `nx*ny*nz*24*16` bytes, which is the stack size needed.

9.4. Running under `MPI`

The situation for `MPI` is a bit more complex than for `OPENMP`, so that the file `Makefile.mpi` will almost certainly have to be modified. One crucial difference to the other makefiles is that the module `Parallel` is now generated from the source file `parallel.f90`. In addition the compilation commands have to be adapted; something like `mpif90` will be needed but is installation dependent. In addition a command like `mpirun` will be needed for execution; the user is advised to consult local documentation.

9.5. Required input

Here it is just summarized what input is needed for a static or dynamic calculation. A full description can be found with the documentation for the `NAMELIST`s.

9.5.1. Static calculation

The `NAMELIST`s needed are, in that order:

> `files`, `force`, `main`, `grid`, `static`. In addition, if initialization is from fragments, `fragments`, and for user initialization possibly `user` (only if the user initialization requires input).

9.5.2. Dynamic calculation

The `NAMELIST`s needed are, in that order:

> `files`, `force`, `main`, `grid`, `dynamic`, For external field excitation `extern`, and in all cases `fragments`.

9.6. Test cases

To allow checking the proper behavior of the code, we provide three test cases exercising different functions: a static calculation for the ground state of ^{16}O, a dynamic calculation using an external excitation to stimulate a giant resonance in ^{16}O, and finally a sample deep-inelastic collision of two ^{16}O nuclei. The test cases directory contains a more detailed description of these cases and what to look for principally in the results. Note that since the calculations are quite large-scale, differences in roundoff errors may lead to the output not being quite identical to the samples provided.

10. Caveats concerning the code

The user should be aware of the limitations of the code in various respects but also note some less straightforward procedures for improving accuracy in certain cases.

10.1. Static calculations

Since the main use of the code is expected to be in time-dependent calculations, the static part is less highly developed. The omission of all symmetry restrictions, while very useful for innovative applications with time-dependence, can cause some problems in the static case.

1. The spin is not aligned along a fixed direction. Since for even-even nuclei Kramers degeneracy operates, two degenerate levels will mix in an uncontrolled way to produce an arbitrary spin alignment. This can be remedied by diagonalizing the spin operators in such a two-level subspace.
2. The center of mass may move away from the origin during the iterations. This is typically a very small effect and will be corrected when the wave functions are placed at a given position in the dynamic initialization. The calculation of observables, however, should always use coordinates relative to the real center of mass.
3. In very heavy nuclei sometimes even a rotation was seen, as the reoccupation of high-lying levels can change the geometric orientation. If this is a serious problem, constraints should be introduced or another code used for the static calculation.
4. The harmonic oscillator initialization can be quite deficient for heavier nuclei. It is planned to develop a Nilsson-model alternative; meanwhile in case of problems the use of wave functions from axial or symmetry-restricting codes could be implemented by generating a wave function file from such results. Since this will involve interpolation, a number of static iterations should then be performed using the present code to improve stability.

10.2. Dynamic calculations

Here it is important what the required accuracy will be. Most exploratory calculations will not pose high accuracy demands. There are several possible ways in which accuracy can be improved if needed:

1. The initial configuration may be improved. If the fragment nuclei are not situated at grid points, one should run a number of static iterations with each fragment situated alone in the new grid.
2. If the fragments are deformed, the energy estimate from the Rutherford trajectory will not be reliable; in this case it is recommended to run a number of dynamic iterations, observe the change in relative distance, and then correct the boost energies to match the correct velocity of relative motion.

11. Modifying the code

Since the code was developed with a view for easy modification, in this Section we give some advice on how to add new things to it and how to run simulations.

11.1. Modifying the Skyrme force

The code comes with a quite large database of Skyrme force parametrizations together with appropriate pairing parameters built in. This is certainly useful to avoid mistakes in the input by having to indicate only the name of the force. Still there will be a need to add new forces and even forces with a different density functional to it, for which we suggest three different approaches.

11.1.1. Direct parameter input

If a specific parametrization is not expected to be a permanent addition to the code, for example if one or several parameters are varied to study the sensitivity of the results to specific parts of the density functional, the best way is to use the facility for giving the parameters directly in the input. This is triggered by using some force name that is not in the database, in which case the routine expects all parameters to be given in `NAMELIST forces`.

11.1.2. Expanding the database

At present the database of forces is contained in the file `forces.data` as a long initialization statement. This has the advantage of readability and avoids having to make a database file accessible in every directory used for code applications.

So a new force which will be used more permanently can be added simply by adding the appropriate lines in `forces.data` and increasing the number assigned to `nforce` accordingly. There is a slight danger that the total length of the list will exceed the 255 lines allowed by the Fortran standard; in this case either remove some outdated forces, remove the separation lines of all stars, or if there is still a problem, convert the initialization to `DATA` statements initializing a smaller number of forces in each case.

11.1.3. Adding new physics to the density functional

This can of course require a lot more modifications to the code. Generally speaking, such a new term will appear not only in the density functional, but also leads to new contributions in the single-particle Hamiltonian and may require new types of densities and currents. In addition, it will involve new force parameters. So in general the following steps will be needed:

1. **Parameter definition:** The new parameter can be added to the general `Force` type definition in `forces.f90`. This is the most logical and systematic way, but we recommend it only if the new physics is to be there permanently, since in this case the whole database has to be updated to give a default value — probably zero — to the new parameter for each existing force. The alternative is to leave this parameter as a separate entity, which can still be a module variable in `Forces` and be input using the same namelist. Derived parameters should also be calculated here.
2. **New densities and currents:** everything that can be defined directly as a sum over the occupied single-particle wave functions should be defined and calculated in module `Densities`. It should be a module variable and allocated during initialization similar to the densities already defined. Then the contributions of the different derivatives of the wave function can be accumulated separately as is done for the existing contributions.

3. **Calculation of mean-field components:** subroutine `skyrme` in module `Meanfield` is the place where the fields appearing in the single-particle Hamiltonian are calculated. The difference to module `Densities` is that wave functions are not involved directly, but only combinations and derivatives of the densities and currents need to be evaluated. The new fields should be defined and allocated and then calculated in subroutine `skyrme`. The handling of derivatives again can be imitated based on the existing terms.
4. **Single-particle Hamiltonian:** The additional contribution to the single-particle Hamiltonian must be calculated in subroutine `hpsi` of module `Meanfield`. Code has to be added to calculate how the additional terms act on the input wave function `pinn` and the result has to be added to the output wave function `pout`. Again, for efficiency the spatial derivatives can be handled in separate loops.
5. **Contribution to the energy:** subroutine `integ_energy` in module `Energies` must include an additional contribution of the new term, which in this case means simply computing the expression for the energy functional. Probably it will be useful to add this up in some new variable (defined as a module variable), so that the contribution of the new term can be printed out together with the other contributions in subroutines `sinfo` of module `Static` or subroutine `tinfo` of module `Dynamic`, respectively.
6. **Output of densities:** it may be desirable to write out the new densities, currents, or mean-field contributions into the `*.tdd` files. To do that, subroutine `write_densities` in module `Inout` must be modified. A new letter to use for `writeselect` must be defined and selected in the `SELECT CASE` statement, and then depending on whether it is a scalar or a vector field, `write_one_density` or `write_vec_density` is called with a descriptive name given to the field.
   The complication that occurs here is that these writing routines assume isospin-dependent fields and output either isospin-separate or isospin-summed fields depending on the value of `write_isospin`. If the field has no isospin dependence, the statements used to output `wcoul` should be imitated.

11.2. Using constraints in the static calculation

There are many situations in which it is useful to solve the static Hartree-Fock equations with added constraints. The most well-known application is a quadrupole constraint, in which the expectation value $\langle\Psi|\hat{H}-\lambda\hat{Q}_{20}|\Psi\rangle$ is minimized to obtain deformed states of the nucleus. In the simplest case the Lagrange multiplier λ may be kept fixed, in which case one has to accept the resulting quadrupole moment, or, alternatively, some iterative change in λ is applied to make the solution converge to a desired value for $\langle\Psi|\hat{Q}_{20}|\Psi\rangle$.

The various ways of introducing a constraint are discussed extensively in chapter 7.6 of [52], so that here we only briefly discuss practical considerations for adding a constraint to Sky3D.

Adding a constraint corresponds to including another potential term in the single-particle Hamiltonian, e. g., for the quadrupole case

$$\hat{h} \longrightarrow \hat{h}-\lambda(2z^2-x^2-y^2) \tag{40}$$

This could be implemented essentially analogously to the use of an external exciting field in the time-dependent case. Construct a subroutine to compute this external potential and add it to the mean-field potential upot. This should be applied every time the single-particle Hamiltonian is applied to a wave function in module Static, i. e., after each call to subroutine skyrme. Then an additional subroutine should be written which is called at the end of each iteration to compute the expectation value of the constraining operator (if not already done elsewhere in the code) and adjust the value of λ if necessary. The method for adjusting λ will depend on the specific constraint used [? ?].

If the constraint is generated by some thing more complicated than a scalar potential, like the orbital and spin operators in the case of cranking, it will be more convenient to implement the constraint inside the subroutine hpsi at those places where the appropriate derivatives are available.

Since the numerical method used does not restrict deformation, care must be taken for unstable constraints like the quadrupole, which can go to large values at the edges of the computational box and may pull the wave functions there. If that becomes a problem, the operator should be damped suitably [50].

11.3. Analyzing the results in new ways

The code provides quite a large number of physical observables in its output files. For new applications it may be necessary to look at additional ones. There are essentially two ways to implement this:

1. If the new observable depends only on density and mean-field components, the easiest way is to use the *.tdd files, where if necessary more fields can be output by modifying the subroutine write_densities. As an example for reading the *.tdd files, we provide a code calculating the tensor of inertia among the utilities.
2. It becomes more complicated if the wave functions have to be used. Here one possibility is to add a call to a user-written routine at the beginning of the static or dynamic loops, which then can use the array psi in any way desired. This may not be a good option if the calculations are lengthy and the analysis routine may have to be modified several times. In this case it is better to generate a new file name for the wffile each time write_wavefunctions is called, similar to the way it is done in write_densities, and to do the analysis by reading the wave function files.

Acknowledgments

This work was supported by the Bundesministerium für Bildung und Forschung under contract No. 05P12RFFTG, by the UK STFC under grant number ST/J000051/1, and by the U.S. Department of Energy under grant No. DE-FG02-96ER40975 with Vanderbilt University.

References

[1] A. L. Fetter, J. D. Walecke, Quantum Theory of Many-Particle Systems, McGraw-Hill, New York, 1971.

[2] J. Maruhn, P.-G. Reinhard, E. Suraud, Simple models of many-fermions systems, Springer, Berlin, 2010.

[3] R. M. Dreizler, E. K. U. Gross, Density Functional Theory: An Approach to the Quantum Many-Body Problem, Springer-Verlag, Berlin, 1990.

[4] M. Bender, P.-H. Heenen, P.-G. Reinhard, Self-consistent mean-field models for nuclear structure, Rev. Mod. Phys. 75 (2003) 121.
URL `http://dx.doi.org/10.1103/RevModPhys.75.121`

[5] P. Dirac, Exchange phenomena in the Thomas-Fermi-atom , Proc. Cambridge Philos. Soc. 26 (1930) 376.

[6] G. E. Brown, Unified Theory of Nuclear Models and Forces, 3rd Edition, North-Holland, Amsterdam, London, 1971.

[7] P.-G. Reinhard, E. Suraud, Introduction to Cluster Dynamics, Wiley, New York, 2004.

[8] P. Bonche, S. E. Koonin, J. W. Negele, One–dimensional nuclear dynamics with time–dependent hartree–fock approximation, Phys. Rev. C 13 (1976) 1226–1258.

[9] J. W. Negele, The mean–field theory of nuclear structure and dynamics, Rev. Mod. Phys. 54 (1982) 913–1015.

[10] K. T. R. Davies, K. R. S. Devi, S. E. Koonin, M. R. Strayer, TDHF calculations of heavy–ion collisions, in: D. A. Bromley (Ed.), Treatise on Heavy–Ion Physics, Vol. 3 Compound System Phenomena, Plenum Press, New York, 1985, p. 3.

[11] J. J. Bai, R. Y. Cusson, J. Wu, P.-G. Reinhard, H. Stöcker, W. Greiner, M. R. Strayer, Mean field model for relativistic heavy ion collisions, Z. Phys. A 326 (1987) 269–277.

[12] H. Berghammer, D. Vretenar, P. Ring, Computer program for the time-evolution of a nuclear system in relativistic mean-field theory, Comp. Phys. Comm. 88 (1995) 293.

[13] D. Vretenar, A. Afanasjev, G. Lalazissis, P. Ring, Relativistic hartree-bogoliubov theory: Static and dynamic aspects of exotic nuclear structure, Phys. Rep. 409 (2005) 101.

[14] Y. Hashimoto, K. Nodeki, A numerical method of solving time-dependent hartree-fock-bogoliubov equation with gogny interaction, arXiv:0707.3083.

[15] A. Umar, V. Oberacker, Density-constrained time-dependent hartree-fock calculation of 16o + 208pb fusion cross-sections, Eur. Phys. J. A 39 (2009) 243. doi:10.1140/epja/i2008-10712-5.

[16] N. Loebl, A. S. Umar, J. A. Maruhn, P.-G. Reinhard, P. D. Stevenson, V. E. Oberacker, Single-particle dissipation in a time-dependent hartree-fock approach studied from a phase-space perspective, Phys. Rev. C 86 (2012) 024608. doi:10.1103/PhysRevC.86.024608.
URL `http://link.aps.org/doi/10.1103/PhysRevC.86.024608`

[17] C. Simenel, P. Chomaz, Phys. Rev. C 68 (2003) 024302.

[18] J. Maruhn, P.-G. Reinhard, P. Stevenson, I. Stone, M. Strayer, Phys. Rev. C 71 (2005) 064328.

[19] A. S. Umar, V. E. Oberacker, Phys. Rev. C 71 (2005) 034314.

[20] T. Nakatsukasa, K. Yabana, Phys. Rev. C 71 (2005) 024301.

[21] C. Simenel, Nuclear quantum many-body dynamics, The European Physical Journal A 48 (11) (2012) 1–49. doi:10.1140/epja/i2012-12152-0.
URL `http://dx.doi.org/10.1140/epja/i2012-12152-0`

[22] A. S. Umar, V. E. Oberacker, Time dependent hartree-fock fusion calculations for spherical, deformed systems, Phys. Rev. C 74 (2) (2006) 024606. doi:10.1103/PhysRevC.74.024606.

[23] K. Washiyama, D. Lacroix, Energy dependence of the nucleus-nucleus potential close to the coulomb barrier, Phys. Rev. C 78 (2) (2008) 024610. doi:10.1103/PhysRevC.78.024610.

[24] C. Simenel, M. Dasgupta, D. Hinde, E. Williams, Microscopic approach to coupled-channels effects on fusion, Phys. Rev. C 88 (6) (2013) 064604. doi:10.1103/PhysRevC.88.064604.
URL `http://link.aps.org/doi/10.1103/PhysRevC.88.064604`

[25] C. Simenel, Particle transfer reactions with the time-dependent hartree-fock theory using a particle number projection technique, Phys. Rev. Lett. 105 (19) (2010) 192701. doi:10.1103/PhysRevLett.105.192701.

[26] P. Goddard, N. Cooper, V. Werner, G. Rusev, P. Stevenson, a. Rios, C. Bernards, a. Chakraborty, B. Crider, J. Glorius, R. Ilieva, J. Kelley, E. Kwan, E. Peters, N. Pietralla, R. Raut, C. Romig, D. Savran, L. Schnorrenberger, M. Smith, K. Sonnabend, a. Tonchev, W. Tornow, S. Yates, Dipole response of ¡span class="aps-inline-formula"¿¡math¿¡msup¿¡mrow¿¡/mrow¿¡mn¿76¡/mn¿¡/msup¿¡/math¿¡/span¿Se above 4 MeV, Physical Review C 88 (6) (2013) 064308. doi:10.1103/PhysRevC.88.064308.
URL `http://link.aps.org/doi/10.1103/PhysRevC.88.064308`

[27] R. Balian, M. Vénéroni, Fluctuations in a time-dependent mean-field approach, Phys. Lett. B 136 (5–6) (1984) 301–306. doi:10.1016/0370-2693(84)92008-2.
URL `http://www.sciencedirect.com/science/article/pii/0370269384920082`

[28] J. M. A. Broomfield, P. D. Stevenson, Mass dispersions from giant dipole resonances using the BalianVénéroni variational approach, Journal of Physics G: Nuclear and Particle Physics 35 (9) (2008) 095102. doi:10.1088/0954-3899/35/9/095102.

URL http://stacks.iop.org/0954-3899/35/i=9/a=095102?key=crossref.1508d88d4e2901b4fce484de470d4ed8

[29] C. Simenel, Particle-number fluctuations and correlations in transfer reactions obtained using the balian-veneroni variational principle, Phys. Rev. Lett. 106 (11) (2011) 112502. doi:10.1103/PhysRevLett.106.112502.

[30] B. Avez, C. Simenel, P. Chomaz, Pairing vibrations study from time-dependent hartree-fock-bogoliubov formalism, Intl. J. Mod. Phys. E 18 (10) (2009) 2103–2107. doi:10.1142/S0218301309014378.

[31] B. Avez, P. Chomaz, T. Duguet, C. Simenel, Pairing vibrations study using a time-dependent energy-density-functional approach, Mod. Phys. Lett. A 25 (21-23) (2010) 1997–1998. doi:10.1142/S0217732310000836.

[32] S. Ebata, T. Nakatsukasa, T. Inakura, K. Yoshida, Y. Hashimoto, K. Yabana, hys. Rev. C 82 (2010) 034306.

[33] D. Lacroix, P. Chomaz, S. Ayik, On the simulation of extended {TDHF} theory, Nucl. Phys. A 651 (4) (1999) 369 – 378. doi:http://dx.doi.org/10.1016/S0375-9474(99)00136-0.
URL http://www.sciencedirect.com/science/article/pii/S0375947499001360

[34] M. Tohyama, A. S. Umar, Quadrupole resonances in unstable oxygen isotopes in time-dependent density-matrix formalism, Phys. Lett. B 549 (1-2) (2002) 72–78. doi:10.1016/S0370-2693(02)02885-X.

[35] Y. M. Engel, D. M. Brink, K. Goeke, S. J. Krieger, D. Vautherin, Time-dependent hartree–fock theory with skyrme's interaction, Nucl. Phys. A 249 (1975) 215–238.

[36] J. Dobaczewski, J. Dudek, Time-odd components in the mean field of rotating superdeformed nuclei, Phys. Rev. C 52 (1995) 1827–1839. doi:10.1103/PhysRevC.52.1827.
URL http://link.aps.org/doi/10.1103/PhysRevC.52.1827

[37] A. S. Umar, V. E. Oberacker, Tdhf studies with modern skyrme forces, Eur. Phys. J. A 25 (2005) 553–554. doi:10.1140/epjad/i2005-06-087-y.

[38] J. Erler, P. Klüpfel, P.-G. Reinhard, Self-consistent nuclear mean-field models: example skyrme-hartree-fock, J. Phys. G 38 (2011) 033101. doi:10.1088/0954-3899/38/3/033101.

[39] J. C. Slater, Phys. Rev. 81 (1951) 385.

[40] P. Klüpfel, P.-G. Reinhard, T. J. Bürvenich, J. A. Maruhn, Variations on a theme by Skyrme, Phys.Rev. C 79 (2009) 034310, http://www.arxiv.org/abs/0804.3385.
URL http://link.aps.org/doi/10.1103/PhysRevC.79.034310

[41] P. Klüpfel, J. Erler, P.-G. Reinhard, J. A. Maruhn, Systematics of collective correlation energies from self-consistent mean-field calculations, Eur. Phys. J A 37 (2008) 343, http://www.arxiv.org/abs/0804.340.
URL http://dx.doi.org/10.1140/epja/i2008-10633-3

[42] J. Bartel, P. Quentin, M. Brack, C. Guet, H.-B. Håkansson, Towards a better parametrisation of skyrme forces: A critical study of the SkM force, Nucl. Phys. A 386 (1982) 79–100.

[43] E. Chabanat, P. Bonche, P. Haensel, J. Meyer, R. Schaeffer, New skyrme effective forces for supernovae and neutron rich nuclei, Physica Scripta 1995 (T56) (1995) 231.
URL http://stacks.iop.org/1402-4896/1995/i=T56/a=034

[44] K.-H. Kim, T. Otsuka, P. Bonche, Three-dimensional tdhf calculations for reactions of unstable nuclei, Journal of Physics G: Nuclear and Particle Physics 23 (10) (1997) 1267.
URL http://stacks.iop.org/0954-3899/23/i=10/a=014

[45] M. Kortelainen, J. McDonnell, W. Nazarewicz, P.-G. Reinhard, J. Sarich, N. Schunck, M. V. Stoitsov, S. M. Wild, Nuclear energy density optimization: Large deformations, Phys. Rev. C 85 (2012) 024304. doi:10.1103/PhysRevC.85.024304.
URL http://link.aps.org/doi/10.1103/PhysRevC.85.024304

[46] E. Perlińska, S. G. Rohoziński, J. Dobaczewski, W. Nazarewicz, Local density approximation for proton-neutron pairing correlations: For malism, Phys. Rev. C 69 (1) (2004) 014316.

[47] K. J. Pototzky, J. Erler, P.-G. Reinhard, V. O. Nesterenko, Properties of odd nuclei and the impact of time-odd mean fields: A systematic skyrme-hartree-fock analysis, Eur. Phys. J. A 46 (2010) 299. doi:10.1140/epja/i2010-11045-6.

[48] D. Vautherin, D. M. Brink, Hartree–fock calculations with skyrme's interaction I. spherical nuclei, Phys. Rev. C 5 (1972) 626.

[49] P.-G. Reinhard, H. Flocard, Nuclear effective forces and isotope shifts, Nucl. Phys. A 584 (1995) 467–488.

[50] K. Rutz, J. Maruhn, P.-G. Reinhard, W. Greiner, Fission barriers and asymmetric ground states in the relativistic mean field theory, Nucl. Phys. A 590 (1995) 680, http://www.arxiv.org/abs/nucl-th/9610037.

[51] W. Greiner, J. A. Maruhn, Nuclear Models, Springer Verlag, New York, 1996.

[52] P. Ring, P. Schuck, The Nuclear Many-Body Problem, Springer–Verl., New York, Heidelberg, Berlin, 1980.

[53] P.-G. Reinhard, M. Bender, K. Rutz, J. Maruhn, An HFB scheme in natural orbitals, Z. Phys. A 358 (1997) 277, http://www.arxiv.org/abs/nucl-th/9705054.

[54] M. Bender, K. Rutz, P.-G. Reinhard, J. Maruhn, Pairing gaps from nuclear mean–field models, Eur. Phys. J. A 8 (2000) 59, http://www.arxiv.org/abs/nucl-th/0005028.

[55] P.-G. Reinhard, R. Cusson, A comparative study of Hartree-Fock iteration techniques, Nucl. Phys. A 378 (1982) 418.

[56] V. Blum, G. Lauritsch, J. Maruhn, P.-G. Reinhard, Comparison of coordinate-space techniques in nuclear mean-field calculations, J. Comp. Phys. 100 (1992) 364.

[57] G. Bertsch, private communication (2012).

[58] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, 2nd Edition, Cambridge University Press, New York, 1992.

[59] F. Calvayrac, E. Suraud, P.-G. Reinhard, Spectral signals from electronic dynamics in sodium clusters, Ann. Phys. (N.Y.) 255 (1997) 125.
URL `http://dx.doi.org/10.1006/aphy.1996.5654`

[60] P.-G. Reinhard, L. Guo, J. A. Maruhn, Nuclear giant resonances and linear response, Eur. Phys. J. A 32 (2007) 19, http://www.arxiv.org/abs/nucl-th/0703044.
URL `http://dx.doi.org/10.1140/epja/i2007-10366-9`

[61] P.-G. Reinhard, From sum rules to RPA: 1. nuclei, Ann. Phys. (Leipzig) 504 (1992) 632.

[62] N. Loebl, J. Maruhn, P.-G. Reinhard, Equilibration in the time-dependent hartree-fock approach probed with the wigner distribution function, Phys. Rev. C 84 (2011) 034608. doi:10.1103/PhysRevC.84.034608.
URL `http://link.aps.org/doi/10.1103/PhysRevC.84.034608`

[63] P.-G. Reinhard, P. D. Stevenson, D. Almehed, J. A. Maruhn, M. R. Strayer, Role of boundary conditions in dynamic studies of nuclear giant resonances, Phys. Rev. E 73 (2006) 036709.
URL `http://link.aps.org/doi/10.1103/PhysRevE.73.036709`

[64] P. Bonche, B. Grammaticos, S. E. Koonin, Three-dimensional time-dependent hartree-fock calculations of ^{16}O $+^{16}$ O and ^{40}Ca $+^{40}$ Ca fusion cross sections, Phys. Rev. C 17 (1978) 1700.

[65] L. Guo, P.-G. Reinhard, J. A. Maruhn, Conservation properties in the time-dependent Hartree Fock theory, Phys. Rev. C 77 (2008) 041301, http://www.arxiv.org/abs/0804.2127.
URL `http://link.aps.org/doi/10.1103/PhysRevC.77.041301`

[66] U. De Giovannini, D. Varsano, M. A. L. Marques, H. Appel, E. K. U. Gross, A. Rubio, *Ab initio* angle- and energy-resolved photoelectron spectros copy with time-dependent density-functional theory, Phys. Rev. A 85 (2012) 062515. doi:10.1103/PhysRevA.85.062515.

[67] K. Boucke, H. Schmitz, H.-J. Kull, Radiation conditions for the time-dependent schroedinger equation: Applications to strong field photoionization, Phys. Rev. A 56 (1997) 763.

[68] M. Mangin-Brinet, J. Carbonell, C. Gignoux, Exact boundary conditions at finite distance for the time-dependent schrodinger equation, Phys. Rev. A 57 (1998) 3245.

[69] C. I. Pardi, P. D. Stevenson, Continuum time-dependent Hartree-Fock method for giant resonances in spherical nuclei, Physical Review C 87 (1) (2013) 014330. doi:10.1103/PhysRevC.87.014330.
URL `http://link.aps.org/doi/10.1103/PhysRevC.87.014330`

[70] C. I. Pardi, P. D. Stevenson, K. Xu, arXiv:1306.4500.

[71] J. L. Krause, K. J. Schafer, K. C. Kulander, Phys. Rev. A 45 (1992) 4998.

[72] P.-G. Reinhard, E. Suraud, Cluster dynamics in strong laser fields, in: M. A. L. Marques, C. A. Ullrich, F. Nogueira (Eds.), Time-dependent density functional theory, Vol. 706 of Lecture Notes in Physics, Springer, Berlin, 2006, p. 391.
URL `http://dx.doi.org/10.1007/3-540-35426-3_26`

[73] B. Chapman, G. Jost, R. van der Pas, Using OpenMP, MIT Press, Cambridge, 2008.

[74] MPI: A Message-Passing Interface Standard, Version 3.0, High Performance Computing Center, Stuttgart, 2012.

[75] M. Frigo, S. G. Johnson, The design and implementation of fftw3, Proc. IEEE 93 (2005) 216. doi:doi:10.1109/JPROC.2004.840301.

[76] J. W. Eastwood, D. R. K. Brownrigg, J. Comp. Phys. 32 (1979) 24.

Index of modules

Index of procedures

Index of variables

Index of namelists