Nuclear density functional theory calculations for the r-process nucleosynthesis

Daniel Karlsson
carlzone@kth.se

SH204X Degree Project in Physics, Second Cycle
Master’s programme in Engineering Physics
Subatomic and Astrophysics track

Supervisor: Chong Qi
Department of Physics
School of Engineering Sciences
The Royal Institute of Technology (KTH)

Stockholm, Sweden
May 5, 2019
Abstract

Reliable nuclear mass calculations are pivotal not only for understanding the nuclear structure but also for performing accurate large scale network calculations such as the astrophysical rapid neutron capture process (r-process). In this work the nuclear binding energies and the odd and even neutron drip lines were calculated with the code HFB+THO using the blocking scheme for even-odd nuclei through a python code as a framework to find and implement the blocking for a large range of nuclei. It was found that while it is very costly in computational time to perform the calculations for the blocking, over 48% of odd neutron nuclei had a lower ground energy state than which could be found without using this method. We also studied the binding energies by using the liquid drop model (LDM), the DZ10, DZ19, and DZ33 shell model formulae. A massive random sampling scheme was implemented to analyze the uncertainties of these models, their parameters, and their convergence with limited sets of fitting data. It was found that to construct a minimal model that predicts the known binding energies one only need a small portion of the available experimental data, less than 1% of the data for the LDM and DZ10 models, but slightly more for the DZ19, and DZ33 models.

The neutron drip lines found for the HFB+THO calculations were then used to investigate the variations between the DZ models for very neutron rich Sn and Ca isotopes. It is found that the DZ33 model exhibits greater uncertainties in its predictions than the reduced DZ10 and DZ19 models with fewer parameters.
Sammanfattning


Neutrondropplinjerna som hittades genom HFB+THO beräkningarna användes sedan för att undersöka variationerna mellan DZ modellerna för mycket neutronrika Sn och Ca isotoper. Det konstaterades att DZ33 modellen uppvisar större osäkerheter i dess förutsägelser än de reducerade modellerna DZ10 och DZ19 som har färre parametrar.
Contents

1 Introduction 2
  1.1 Background ......................................................... 2
  1.2 Mass models ......................................................... 3
    1.2.1 The liquid drop model ........................................ 3
    1.2.2 The Duflo-Zuker mass formulae ................................ 4
    1.2.3 Hartree theory ................................................... 7
    1.2.4 Hartree Fock theory ............................................. 7
    1.2.5 Hartree Fock Bogolyubov ....................................... 8
    1.2.6 Skyrme interactions ............................................ 10

2 HFB calculations with the HFBTHO code 12

3 Mass formulae calculations with DZ 10,19,33 and the LDM. 23

4 Summary and Discussions 45

5 Appendix 47

References 56
Chapter 1

Introduction

1.1 Background

The r-process is believed to describe how about 50% of the heavy elements in the universe were created. It is a great challenge to simulate this process. It is likely that the r-process mainly occurs in neutron star mergers and that they happen often enough to explain the abundances in the universe. Recent observations of the GW170817 neutron star merger [1][2][3] largely agree with simulations but the uncertainties are large both in the astrophysical initial conditions and in the nuclear properties of the heavier and more unstable elements. In astrophysics, the same tables of masses are often used and little time is spent on improving them. Furthermore an interpolation method is often used instead of calculating the odd neutron energies directly which should introduce large uncertainties. Recently it has been shown that the propagation of nuclear uncertainties to the r-process calculations has a large effect on the final results [4] and other research has been made to measure the impact of each nuclei on the relevant r-process rates [5][6]. This should call for an effort to improve these mass tables to reduce the uncertainty in the final simulations.

The best fit of nuclear binding energies to experimental data is currently from models such as the Duflo-Zuker mass formulae. They are however not truly microscopic models and may become unreliable when approaching more unstable nuclei [7][8] where pairing and continuum effects become increasingly important. The predictions of different models can differ greatly [9], sometimes by as much as several MeV for very neutron rich nuclei. Therefore the r-process simulations rely upon mass tables generated by a mixture of empirical results from experiments, and Hartree-Fock-Bogolyubov calculations. The Hartree-Fock-Bogolyubov on the other hand, the one microscopic model that is possible to use to calculate the whole nuclear chart within a reasonable time frame, does not produce as accurate results compared to known experimental data and the calculations are still computationally heavy. Since these calculations take a long time to run (CPU time on the order of 10 years for the full nuclear chart of even-even and even-odd nuclei), they are mainly performed on super computers that are not accessible for small projects or students.

This work has two main parts, the first is the writing of a code and development of a method to permit more flexibility to the use of the HFB+THO code, as well as gather and analyze the results of those calculations on a normal laptop. The second is to analyze the uncertainties and extrapolation power of the liquid drop model and three different
Duflo-Zuker mass formulae by using a fractional sampling method. During this thesis work, we have also done extensive r-process simulations with the codes r-Java and nucl-net. Those will be reported in another publication.

1.2 Mass models

Although the different mass models analyzed in this work have relatively unknown areas of uncertainties. The simplest model, the six term liquid drop model (LDM), is a good starting point due to its accuracy, its well defined physics, and well understood deficiencies with regard to pairing and shell effects. The calculations will also be compared with three versions of the more accurate Duflo-Zuker model (DZ), and with Hartree-Fock-Bogolyubov (HFB) calculations.

1.2.1 The liquid drop model

The basic assumption of the LDM is that a nucleus can be described up to a high level of accuracy by treating it as a high density, incompressible fluid. The model does a good job of describing the total nuclear binding energy of a nucleus but does not account for shell effects or deformation corrections without further additions. The analogy with a liquid drop motivates five right hand side terms to describe the total binding energy, or mass, of a system of nucleons.

The first term is the so called volume energy term and represents the strong nuclear force which is proportional to the number of nucleons in the system, which in turn in proportional to the volume of the nucleus.

$$V = \frac{4\pi R^3}{3} = 4\pi (r_0^3 A^{\frac{4}{3}})^3 = 4\pi r_0 A$$

$$E_v = C_1 A$$ (1.1)

The second term is the surface energy term, or surface tension. This term is motivated by the fact that nucleons on the surface are not surrounded by nucleons like the bulk nucleons, and thus they are more loosely bound. The loss of binding energy at the surface can be described by the surface tension $\sigma_s$ and the area of the surface as:

$$E_s = -\sigma_s \cdot Area = -\sigma 4\pi R^2 = -\sigma 4\pi r_0^2 A^{\frac{4}{3}} = -C_2 A^{\frac{4}{3}}$$

$$E_s = -\sigma_0 \cdot Area = -\sigma_0 \cdot 4\pi r_0^2 A^{\frac{4}{3}} = -C_2 A^{\frac{4}{3}}$$ (1.2)

The third term comes from the repulsive force of the charged nucleons, the protons, and is aptly named the coulomb energy term $E_c$. It is negative as its effect is lowering the total binding energy. By Coulomb’s law for a charged sphere the electrostatic energy is therefore:

$$E_c = -\frac{3}{5} Z^2 e^2 R = -C_3 \frac{Z^2}{A^{\frac{4}{3}}}$$

$$E_c = -\frac{3}{5} Z^2 e^2 R = -C_3 \frac{Z^2}{A^{\frac{4}{3}}}$$ (1.3)

The fourth term is inspired by Raoult’s law in two component liquids and states that the minimum energy of the system between two non polar attractive components occurs
when there is an equal concentration of the two. In a nucleus this equates to a negative
symmetry term whenever there is an asymmetry in the number of neutrons and protons.

\[ E_{\text{sym}} = -C_4 \frac{(N - Z)^2}{A^2} \]  

(1.4)

The fifth and last term making up the binding energy in this model is the pairing
energy \( E_p \) which accounts for the pairing of nuclei. This will either increase or decrease
the binding energy for even-even and odd-odd nuclei respectively. For odd-even or even-
odd nuclei the effect is zero.

\[ E_p = C_5 \frac{\delta}{A^{1.2}} \]  

(1.5)

Where \( \delta = +1, 0, -1 \) for even-even, odd A, and odd-odd, respectively.

Together these terms form the equation for the Liquid Drop Model:

\[ BE_{\text{LDM}} = C_1 A - C_2 A^{2/3} - C_3 \frac{Z^2}{A^{2/3}} - C_4 \frac{(N - Z)^2}{A^{2/3}} + C_5 \frac{\delta}{A^{1.2}} \]  

(1.6)

In this work a modified version of the LDM is used [10]. In this modified version a
sixth term representing a correction to the Coulomb energy and the binding energy is
added. One gets:

\[ BE_{\text{LDM}} = a_v \left[ 1 + \frac{4k_v}{A^2} T_z(T_z + 1) \right] A + a_s \left[ 1 + \frac{4k_s}{A^2} T_z(T_z + 1) \right] A^{2/3} + 3Z^2 e^2 \frac{5r_0}{A} + \frac{C_4 Z^2}{A} \]  

(1.7)

The terms here represent the volume energy, the surface energy, the Coulomb energy,
and the correction to the Coulomb energy where \( T_z \) is the third component of isospin, 
\( e \) is the electronic charge, and \( a_v, k_v, a_s, k_s, r_0, \) and \( C_4 \) are free parameters that will be
fitted to experimental data.

1.2.2 The Duflo-Zuker mass formulae

The Duflo-Zuker mass formulae are constructed from the assumption that a smooth
enough effective pseudo potential exists, so that it’s possible to perform Hartree-Fock
calculations where the corresponding Hamiltonian can be broken up into a monopole
and a multipole part.

\[ H = H_m + H_M \]

The monopole Hamiltonian is responsible for saturation and single particle properties
and functions as a platform for Hartree-Fock calculations that is extracted from the interacting shell model as a mean field. All residual interactions such as pairing interactions,
quadruple, and Wigner correlations are a part of the multipole Hamiltonian. The binding
energy of the system can thus be written as:

\[ BE = \langle H \rangle - E_C - E_{\text{sym}} - E_P \]  

(1.8)

Where \( BE \) is the binding energy and P, C, and sym, stands for for the pairing, Coulomb, and symmetry energies.
The monopole Hamiltonian is defined as a sum of its microscopic and macroscopic parts.

\[ H_m = H_M + H_s + H_d \] (1.9)

Where M is the macroscopic term that considers all nucleons, s stands for micro
scopical spherical effects, and d stands for microscopic deformed effects. Here three DZ models will be used [11] where the DZ-19 [12] model is a simplified version of the DZ-33 model [13], meaning certain terms have been neglected and those have been kept who are deemed to represent clear physics. The full DZ model has 33 terms and contains 28 monopole terms, the Coulomb energy, symmetry energy, surface symmetry energy, and two pairing terms. While the simplest DZ model has only 10 terms, of which six are monopole terms, and the other four are Coulomb energy, symmetry and surface symmetry energies, and the pairing energy. The DZ-10 model is, in contrast to DZ-19, not a simplified DZ-33 model, as the monopole terms in the DZ-10 model are defined differently than the corresponding terms in the full model, while the shell structure is assumed to be the same in all DZ models. The models are constructed by a filling of the shells in sequence, and deformation states are defined as the promotion of four neutrons and four protons to the next major shell.

The symmetry energy terms are defined differently in the DZ models than in the LDM and takes the following forms for the symmetry energy and the surface symmetry energy respectively.

\[ E_{sym} = a_{sym} \frac{T(T + 1)}{A \rho} \]

\[ E_{ssym} = -a_{ssym} \frac{T(T + 1)}{A^{\frac{7}{3}} \rho^2} \] (1.10)

In addition there are also contributions to the symmetry energy from the other monopole terms which can be seen in the dependence on the neutron and proton numbers squared in the two master terms.

\[ FM^+ = \frac{1}{2} \left[ \left( \sum_{p=0} m_p \sqrt{D_p} \right)^2 + \left( \sum_{p=0} t_p \sqrt{D_p} \right)^2 \right] \]

\[ FM^- = \frac{1}{2} \left[ \left( \sum_{p=0} n_p \sqrt{D_p} \right)^2 + \left( \sum_{p=0} z_p \sqrt{D_p} \right)^2 \right] \] (1.11)

Again without the scaling factor.

The Coulomb energy in the DZ model is given by the expression:

\[ E_C = a_c \frac{-Z(Z - 1) + 0.76[Z(Z - 1)]^{\frac{3}{2}}}{A^{\frac{7}{4}}[1 - (\frac{T}{A})^2]} \] (1.12)

Which is also distinctly different than the corresponding LDM Coulomb energy.

The terms in the full model be summarized in short here. The 28 monopole terms are a mix of volume terms, surface terms, the Coulomb energy, symmetry energy, surface symmetry energy, and two paring terms. The volume terms are named FM+, FS+, FS-,
FC+, PM+, PS+, PS-, S3, SQ-, D3, QQ+, D0, QQ-, and SS. The surface terms are named the same except with small letters and defined as:

\[ fs+ = \frac{FS+}{\rho} \]  

(1.13)

The volume terms are given (without scaling factors) as:

\[ FM+ = \frac{1}{2} \left[ \left( \sum_{p=0}^{\infty} \frac{m_p}{D_p} \right)^2 + \left( \sum_{p=0}^{\infty} \frac{t_p}{D_p} \right)^2 \right] \]

\[ FS+ = \left( \sum_{p} \frac{s_\pi + s_\nu}{p + 1} \right)^2 + \left( \sum_{p} \frac{s_\pi - s_\nu}{p + 1} \right)^2 = 2 \left( \left( \sum_{p} \frac{s_\pi}{p + 1} \right)^2 + \left( \sum_{p} \frac{s_\nu}{p + 1} \right)^2 \right) \]

\[ FS- = \left( \sum_{p} \frac{s_\pi + s_\nu}{p + 1} \right)^2 - \left( \sum_{p} \frac{s_\pi - s_\nu}{p + 1} \right)^2 = 4 \left( \sum_{p} \frac{s_\pi}{p + 1} \right)^2 \left( \sum_{p} \frac{s_\nu}{p + 1} \right)^2 \]

\[ FC+ = \left( \sum_{p} \frac{m_p}{D_p^3} \right) \left( \sum_{p} \frac{s_\pi + s_\nu}{p + 1}D_p^2 \right)^2 + \left( \sum_{p} \frac{t_p}{D_p^3} \right) \left( \sum_{p} \frac{s_\pi - s_\nu}{p + 1}D_p^2 \right)^2 \]

\[ PM+ = \left( \sum_{p} \frac{n_\pi p}{D_p^3} \right)^2 + \left( \sum_{p} \frac{n_\nu p}{D_p^3} \right)^2 \]

\[ PS+ = 2 \left( \left( \sum_{p} \frac{s_\pi D_p^3}{p + 1} \right)^2 + \left( \sum_{p} \frac{s_\nu D_p^3}{p + 1} \right)^2 \right) \]

\[ PS- = 4 \left( \sum_{p} \frac{s_\pi D_p^3}{p + 1} \right)^2 \left( \sum_{p} \frac{s_\nu D_p^3}{p + 1} \right)^2 \]

where: \( s = \frac{n_\nu p - n_\nu p}{2} \)

\[ S3 = \frac{n_\nu \tilde{n}_\nu (n_\nu - \tilde{n}_\nu)}{N_\nu} + \frac{n_\nu \tilde{n}_\pi (n_\nu - \tilde{n}_\pi)}{N_\pi} \] (The same as in DZ-10)

\[ SQ- = \frac{2n_\nu \tilde{n}_\nu + 2n_\nu \tilde{n}_\pi}{N_\nu} \]

\[ D3 = \frac{n_\nu' \tilde{n}_\nu' (n_\nu' - \tilde{n}_\nu')} {N_\nu^2} + \frac{n_\pi' \tilde{n}_\pi' (n_\pi' - \tilde{n}_\pi')} {N_\pi^2} \]

\[ QQ+ = \frac{2(n_\nu' \tilde{n}_\nu')^2}{N_\nu^3} + \frac{2(n_\nu' \tilde{n}_\pi')^2}{N_\pi^3} \]

\[ D0 = 16 - \frac{2n_\nu \tilde{n}_\nu + 2n_\pi}{N_\nu^2} \frac{n_\nu' \tilde{n}_\nu'}{N_\pi} = 16 - QQ- \]

\[ QQ- = \frac{2n_\nu' \tilde{n}_\nu' + 2n_\pi \tilde{n}_\pi'}{N_\nu^2} \frac{n_\nu' \tilde{n}_\pi'}{N_\pi^2} \]

\[ SS = \sum_{p>2} ss_{\pi,p} \left( \frac{n_\pi p}{D_p^2} + \frac{s_\pi}{p + 1} \right) + \sum_{p>2} ss_{\nu,p} \left( \frac{n_\nu p}{D_p^2} + \frac{s_\nu}{p + 1} \right) \]

where:

\[ ss_p = \begin{cases} 
\frac{n_\nu(p-1)}{2p} & \text{if } n_\nu \leq p(p-1) \\
\frac{(n_\nu-p(p-1))}{p} & \text{if } n_\nu > p(p-1) 
\end{cases} \]
Where the subscript \( p \) denotes the principal quantum number of the proton or neutron Harmonic Oscillator major shell, and \( n_p \) denotes the number of neutrons in that shell. The total number of nucleons is given as \( m_p = n_p + z_p \), and \( t_p = |n_p - z_p| \). The degeneracy of the \( p \) shell is \( D_p = (p + 1)(p + 2) \).

### 1.2.3 Hartree theory

Hartree theory treats particles which interact with a central mean field and with each other but neglects antisymmetric particle exchange and pairing correlations. It can be described by the Hamiltonian

\[
H = \sum_i A \{T_i + u_0(r_i)\} + \frac{1}{2} \sum_{ij} Z v(r_i, r_j) \quad (1.15)
\]

where \( T_i \) is the kinetic energy of the particle, \( u_0 \) is the central field potential and \( v(r_i, r_j) \) is the interaction potential. The average electrostatic interaction felt by one charged particle from all the other charged particles is

\[
u_1(r_i) = \sum_{j \neq i} Z \int |\psi_j(r_j)|^2 v(r_i, r_j) dr_j \quad (1.16)
\]

This leads to the wave equation

\[
\left\{ -\frac{\hbar^2}{2M} \nabla^2 + u_0(r) + u_1(r) \right\} \psi_i(r) = \epsilon_i \psi_i(r) \quad (1.17)
\]

### 1.2.4 Hartree Fock theory

In an extension to this, the Hartree-Fock theory it is assumed that the wave function of the nucleus is the antisymmetric product of independent particle wave functions which can be represented by the slater determinant.

\[
\phi(1, 2...N) = \hat{A} \psi_1(1)\psi_2(2)...\psi_N(N) = \frac{1}{A!^{1/2}} \begin{bmatrix} \psi_1(1) & \psi_2(1) & \ldots & \psi_N(1) \\ \psi_1(2) & \psi_2(2) & \ldots & \psi_N(2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(N) & \psi_2(N) & \ldots & \psi_N(N) \end{bmatrix}
\]

Where \( \hat{A} \) is an antisymmetric operator. The variational principle states that for small perturbations that preserve the normalization of the single particle wave functions of an eigenstate the energy expectation value is stationary and thus

\[
\delta \langle \phi | H | \phi \rangle = \langle \delta \phi | H | \phi \rangle = 0 \quad (1.18)
\]

For the Hamiltonian

\[
H = \sum_i A T_i + \frac{1}{2} \sum_{ij} A v(r_i, r_j) \quad (1.19)
\]
with the energy expectation value

$$\langle \phi | H | \phi \rangle = -\frac{\hbar^2}{2M} \sum_i A_i \int \psi_i^\ast(r) \nabla^2 \psi_i(r) dr + \frac{1}{2} \sum_{ij} \int \int \psi_i^\ast(r) \psi_j^\ast(r') v(r,r') \psi_i(r) \psi_j(r')$$

$$- \frac{1}{2} \sum_{ij} \int \int \psi_i^\ast(r) \psi_j^\ast(r') v(r,r') \psi_i(r) \psi_j(r')$$

(1.20)

and when we apply the variation $\langle \delta \phi | H | \phi \rangle = 0$ we get the single-particle wave equation

$$-\frac{\hbar^2}{2M} \nabla^2 \psi_i(r) + \sum_j \int d r' \psi_j^\ast(r') v(r,r') \psi_j(r) \psi_i(r)$$

$$- \sum_j \int d r' \psi_j^\ast(r') v(r,r') \psi_j(r) \psi_i(r') = \epsilon_i \psi_i(r)$$

(1.21)

where $\epsilon_i$ is the single particle energy.

Equation (1.21) can be written in the form

$$-\frac{\hbar^2}{2M} \nabla^2 \psi_i(r) + \int d r u(r,r') \psi_i(r') = \epsilon_i \psi_i(r)$$

(1.22)

where

$$u(r,r') = \delta(r - r') \sum_j \int d r'' v(r,r'') \psi_j(r'') \psi_j^\ast(r'') - \sum_j \int v(r,r') \psi_j(r) \psi_j^\ast(r')$$

(1.23)

where the first term can be identified as the Hartree field from equation (1.25) and the second term is the non-local exchange contribution.

### 1.2.5 Hartree Fock Bogolyubov

To include pairing correlations HFB theory look for wave functions of independently moving quasi particles that are determined by a variational principle. The Hamiltonian then reduces to

$$H = H_{HF} + H_\Delta$$

(1.24)

where $\Delta$ is the pairing field. The quasi particle operators are linked to the single particle operators by the Bogolyubov transformation

$$\beta_n^+ = \sum_i (U_{in} a_i^+ + V_{in} a_i)$$

(1.25)

which can be written in a short hand notation as

$$\begin{pmatrix} \beta \\ \beta^+ \end{pmatrix} = \begin{pmatrix} U & V^* \\ V & U^* \end{pmatrix} \begin{pmatrix} a \\ a^+ \end{pmatrix}$$

(1.26)

Where $a$ and $a^+$ are bare particle operators that creates or destroys a particle state, for example a plane wave or a harmonic oscillator state. The new quasi particle operators
β and β⁺ must obey the same fermionic commutation relations as the particle state operators so that the matrix

\[ W = \begin{pmatrix} U & V^* \\ V & U^* \end{pmatrix} \]  

(1.27)

must be unitary. From this follows the relations of the coefficients of the transformation matrix

\[ U^+U + V^+V = 1 \quad UU^+ + V^*V^T = 1 \]
\[ U^TV + V^TU = 0 \quad UV^+ + V^*U^T = 0 \]  

(1.28)

Using these conditions, (1.25) can be inverted to give

\[ a_n^+ = \sum_k U^*_{nk}\beta_k^+ + V_{nk}\beta_k \]  

(1.29)

The new quasi particle vacuum can then be constructed as

\[ |\Phi\rangle = \prod_n \beta_n |\pm\rangle \]  

(1.30)

Where \(|\pm\rangle\) is the bare particle vacuum. The HFB wave functions are then the wave functions which obey these conditions for the set of quasi particle operators given in (1.25).

The coefficients \(U_{nk}\) and \(V_{nk}\) are not uniquely defined by the HFB wave function, to accomplish this two new quantities, the density matrix and the pairing tensor are defined as

\[
\rho_{nn'} = \langle \Phi | a_{n'}^+ a_n | \Phi \rangle \\
\kappa_{nn'} = \langle \Phi | a_{n'}^+ a_n | \Phi \rangle
\]

(1.31)

which in matrix notation can be written as

\[
\rho = V^*V^T, \quad \kappa = V^*U^T = -UV^+
\]

(1.32)

where \(\rho\) is hermitian and \(\kappa\) is skew symmetric, that is

\[
\rho = \rho^+, \quad \kappa^T = -\kappa
\]

(1.33)

These quantities may be written as part of a of a general quasi-particle density

\[
\mathcal{R} = \begin{pmatrix} \rho & \kappa \\ -\kappa^* & 1 - \rho^* \end{pmatrix}
\]

(1.34)
It is important to note that the HFB model, like the BCS model where the handling
of the quasi particles was developed, does not conserve the number of particles. Instead
the conserved quantity is the expectation value

$$\langle \Psi | \hat{N} | \Psi \rangle = N \quad (1.35)$$

In the HFBTHO code the particle number error can be corrected for by following
the Lipkin - Nogami prescription [14] for an approximate particle number projection,
followed by an exact particle number projection.

The ground state is obtained by minimizing the total energy

$$E = \langle \Psi | H | \Psi \rangle = E[\rho, \kappa, \kappa^*] \quad (1.36)$$

where $$E[\rho, \kappa, \kappa^*]$$ is the energy functional.

The Hamiltonian is assumed to be

$$H = \sum_{\nu \nu'} T_{\nu \nu'} a^\dagger_{\nu} a_{\nu'} + \frac{1}{4} \sum_{\mu \nu \mu' \nu'} \Delta_{\mu \nu} a^\dagger_{\mu} a^\dagger_{\nu'} a_{\mu'} a_{\nu} \quad (1.37)$$

The HFB wave equations can be derived by the variational principle or by the use of
the wick’s theorem method and give the result

$$\begin{pmatrix} h & \Delta \\ -\Delta^* & -h^* \end{pmatrix} \begin{pmatrix} U_k \\ V_k \end{pmatrix} = E_k \begin{pmatrix} U_k \\ V_k \end{pmatrix} \quad (1.38)$$

which is a diagonalization problem where

$$h_{ij} = \frac{\delta E}{\delta \rho_j}, \quad \Delta_{ij} = -\frac{\delta E}{\delta \kappa_j^*} = -\Delta_{ji} \quad (1.39)$$

### 1.2.6 Skyrme interactions

The energy density functional used in the HFBTHO code for the Skyrme force [15] has
the form

$$E[\rho, \hat{\rho}] = \int d^3r H(r) = \int d^3r H(r) + H'(r) \quad (1.40)$$

Where $$H(r)$$ is the mean-field energy density and $$H'(r)$$ is the pairing energy density. Specifically the mean-field has the form

$$H(r) = \frac{\hbar^2}{2m} \tau + \frac{1}{2} t_0 \left[ (1 + \frac{1}{2} x_0) \rho - (\frac{1}{2} + x_0) \sum_q \rho_q^2 \right]$$

$$+ \frac{1}{2} t_1 \left[ (1 + \frac{1}{2} x_1) \rho (\tau - \frac{3}{4} \Delta \rho) - (\frac{1}{2} + x_1) \sum_q \rho_q (\tau_q - \frac{3}{4} \Delta \rho_q) \right]$$

$$+ \frac{1}{2} t_2 \left[ (1 + \frac{1}{2} x_2) \rho (\tau - \frac{1}{4} \Delta \rho) - (\frac{1}{2} + x_2) \sum_q \rho_q (\tau_q + \frac{1}{4} \Delta \rho_q) \right]$$

$$+ \frac{1}{12} t_3 \rho^6 \left[ (1 + \frac{1}{2} x_3) \rho^2 - (x_3 + \frac{1}{2}) \sum_q \rho_q^2 \right]$$

$$- \frac{1}{8} (t_1 x_1 + t_2 x_2) \sum_{ij} J_{ij}^2 + \frac{1}{8} (t_1 - t_2) \sum_{q,ij} J_{q,ij}^2 - \frac{1}{2} W_0 \sum_{ijk} \epsilon_{ijk} (\rho \nabla_k J_{ij} + \sum_q \rho_q \nabla_k J_{q,ij}) \quad (1.41)$$
and the pairing energy is

\[
H'(\mathbf{r}) = \frac{1}{2} V_0 \left[ 1 - V_1 \left( \frac{\rho}{\rho_0} \right)^\gamma \right] \sum_q \tilde{\rho}_q^2
\]  

(1.42)

Where the sums goes over \( q = n \), and \( q = p \) for neutron and proton densities. All densities without indices are the sums of proton and neutron contributions (the isoscalar density). The kinetic energy density is given by \( \tau \), and \( \tilde{\rho} \) is the local pairing density. For the complete picture one also has to add the Coulomb interaction for the proton states which depends on the charge density \( \rho_{ch} \), in this method that density is approximated by a point proton density. The Coulomb interaction has two parts, a direct interaction term, and a non-local exchange term which is treated with the Slater approximation.

\[
H'_{\text{dir}} = \frac{e^2}{2} \int d^3r d^3r' \frac{\rho_{ch}(\mathbf{r}) \rho_{ch}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \approx e^2 \int d^3r' \frac{\rho_p(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \]  

(1.43)

\[
H'_{\text{ex}} = -\frac{3}{4} e^2 \left( \frac{3}{\pi} \right)^{\frac{3}{2}} \int d^3r \rho_p^{\frac{3}{2}}(\mathbf{r}) \approx -e^2 \left( \frac{3}{\pi} \right)^{\frac{3}{2}} \rho_p^{\frac{3}{2}}(\mathbf{r}) \]  

(1.44)

The free parameters of the model, \( t_0, t_1, t_2, t_3, x_0, x_1, x_2, x_3, W_0 \), and \( \alpha \) are fitted to experimental data, as well as the pairing strength \( V_0 \). In this work the pairing strength found in [16][17][18] is used.
Chapter 2

HFB calculations with the HFBTHO code

The calculations were made with the code HFBTHOv2.00 [19] and were run repeatedly with a python script to alter the input and to record and analyze the output data.

The HFB code has very little flexibility, which may be less important when using a super computer and all calculations are done in bulk, but when you need to be very selective about which calculations to perform it's important to be able to do as few as possible and still get an accurate result. In this work python code has been written which allows one to specify a range of nuclei to be calculated, to specify a range of initial deformations, and to specify how many shells above the fermi level that should be calculated.

The first thing to note is the fact that you cannot predict what the final deformation will be, and the calculations can get stuck in a local energy minimum as shown in 2.1. To find the correct deformation of the nuclei and thereby the correct ground state energy, several initial deformations ranging from -0.3 to 0.3 have been used. The sign indicates the shape of the nucleus, $\beta > 0$ for a prolate and $\beta < 0$ for an oblate deformation.
To find the minimum ground state energy for the odd neutron nuclei (and odd proton nuclei which are not considered in this work), the code uses time reversal symmetry and the equal filling approximation (EFA) one has to account for the unpaired particles as well as the quasi particles. This means in practice that the unpaired particle is assumed to occupy a single state (also its time-reversed partner) which is specified by it’s Nilsson numbers and thus that level can not be occupied by a quasi particle or participate in pairing effects. This is called the blocking scheme. The blocking induces a splitting of any j shell by a breaking of the spherical symmetry. The level that should be blocked is the level that gives the lowest energy state, and while that is the most often the state with the maximum angular momentum projection \( \Omega = +j \), that is not always the case and cannot be known beforehand. This means that for every initial deformation state, one also has to try blocking as many states with different values of \(|\Omega|\) as possible to be sure to get the lowest ground state. There is a possible error introduced with this in that a configuration with the angular momentum projection \(|\Omega| < j\) may not belong to the j shell being calculated [20]. In this work this problem is ignored and as many configurations as was feasible was calculated and the blocked state configurations’ Nilsson numbers were saved.

One can assume it becomes increasingly unlikely that the lowest state will be found the further out you go from the fermi level but it is unknown exactly where it is reasonable to stop. The HFB+THO code permits the option to calculate a two MeV range of shells above the fermi level, typically 20 shells, or to perform one calculation at a time and specify the Nilsson configuration of the specific shell to be blocked. Since the calculations are very time consuming one wants to avoid unnecessary calculations without losing accuracy. It was found here that over 48% of the odd neutron ground states for nuclei up until the odd neutron drip line was not the first blocking candidate (the one with the highest \(|\Omega|\) projection). In fact for some nuclei the lowest energy was found blocking as high as the 10th candidate and the energy difference between the blocked state and the first candidate can be up to several MeV. Table 2.1 lists the number of times the
different blocking candidates for odd nuclei were used to find the lowest ground state and the mean energy difference between that level and the energy of state calculated using the first blocking candidate.

Table 2.1: Blocked levels

<table>
<thead>
<tr>
<th>Block Candidate</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts</td>
<td>439</td>
<td>281</td>
<td>103</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>E diff [MeV]</td>
<td>0</td>
<td>-0.432</td>
<td>-1.036</td>
<td>-0.57</td>
<td>-1.45</td>
<td>-4.1</td>
<td>-0.038</td>
<td>-2.309</td>
<td>-0.689</td>
<td></td>
</tr>
</tbody>
</table>

No lowest states were found by blocking higher states than the 10th candidate. However more levels than that were only blocked for a few nuclei so those extra calculations could very well be necessary.
A crude analysis was made to see if it’s possible to save time by aborting calculations at an early iteration. If this procedure is invalid then the energy obtained will be larger than the ones already found for the same nuclei. This is shown in figure 2.2. A good way to implement this would be to run all calculations in parallel and compare the energy at each iteration.

![Figure 2.2: Error % of calculations when interrupted at early iterations when the energy difference from the already found minimum is greater than the set limit.](image)

Figure 2.2: Error % of calculations when interrupted at early iterations when the energy difference from the already found minimum is greater than the set limit.

From this one can conclude that to have an acceptable level of error one should wait until the 15th iteration to interrupt the calculations and set an appropriate cutoff limit. A typical calculation first performs around 40 iterations for a preliminary HFB calculation with a woods saxon shape potential, and then another 40 iterations for the HFB+THO. Of course we will not have the lowest state to compare with so it is advisable to start with the deformations that are most likely to yield the lowest state first. The number of initial deformations of a certain type and magnitude that was used to find the lowest state are shown in table 2.2.

It appears it would be a good strategy to start with the deformation 0.2 and then work your way down the list. Considering that most the configurations that need to be blocked are among the first candidates, and that most deformations will be found using the first few deformations, a conservative estimate would be that time saved could be more than 50%. The results of the HFB+THO calculations are summarized in the following list for all even-even and even-odd nuclei between $Z = 6, N = 8$ up to $Z = 84, N = 189$. 

15
Table 2.2: Initial deformations

<table>
<thead>
<tr>
<th>Initial deformation</th>
<th>-0.3</th>
<th>-0.2</th>
<th>-0.1</th>
<th>0.0</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts</td>
<td>0</td>
<td>69</td>
<td>314</td>
<td>263</td>
<td>372</td>
<td>523</td>
<td>155</td>
</tr>
</tbody>
</table>

particularly the most calculations have been performed on neutron rich nuclei to get the most accurate result possible for the neutron drip lines

the results include:

- Binding Energies
- Deformations
- Single particle energies
- Pairing energies

From these data, the following properties have been calculated or extracted.

- The neutron- and two neutron separation energies.
- The neutron drip lines
- The binding energy difference between the found lowest ground state and the first blocking candidate.

The results are comparable to earlier calculations made in [21] but as can be seen in figure 2.3 the results are slightly different. This is partly because of the different pairing forces used, but also due to that the 2003 calculations used the Lipkin-Nogami prescription in the seniority pairing approximation. It is also possible that the HFB+THO code has undergone changes since then.
Figure 2.3: Binding energies of this work (blue) and the data of M. V. Stoitsov, J. Dobaczewski, W. Nazarewicz, S. Pittel, and D. J. Dean (red) compared to the AME2016 nuclear binding energy data.
The deformations and pairing energies found are shown in the following figures:

Figure 2.4: Calculated quadrupole deformations with HFB+THO.
The odd and even neutron drip lines are defined as the place where the neutron separation energies become negative for single neutrons and a pair of neutrons. The separations energies are defined as:

\[
S_n(Z, N) = BE(Z, N - 1) - BE(Z, N)
\]
\[
S_{2n}(Z, N) = BE(Z, N - 2) - BE(Z, N)
\]

This means that it is no longer energetically favorable to absorb more neutrons and
thus more neutron rich isotopes will not exist. In figure 2.6 the neutron separation energies for tin isotopes is shown. The drip line is highlighted at the point where the neutron separation energy becomes negative which is at \( N = 105 \) for odd neutrons and \( N = 126 \) for even neutrons.

Figure 2.6: One and two neutron separation energies for tin calculated with HFB+THO.
The difference between the odd and even neutron drip line is unusually large for tin as is illustrated in figure 2.7 where the difference between the two neutron separation energies and the single neutron separation energies has been plotted for HFB+THO with all three DZ models, Z = 50 for tin isotopes has been marked with a dashed line.

![Graphs showing differences between neutron separation energies and single neutron separation energies for HFB+THO and three DZ models.](image)

(a) HFB+THO  (b) DZ10  
(c) DZ19  (d) DZ33

Figure 2.7: The minimal and maximal values of the RMS deviation compared to the training sample (blue), and to all data (red) as a function of the sample size (fraction).

In figure 2.8 is shown the drip lines found for each model.
Figure 2.8: One and two neutron drip lines for three different DZ models and HFB+THO.

Since the r-process path operates at or near the neutron drip lines, the small differences between the models can be of great importance. It appears that the drip lines found for HFB are generally further out and more neutron heavy than for the DZ models, which means that there are more nuclei that can participate and could potentially play an important role.
Chapter 3

Mass formulae calculations with DZ 10,19,33 and the LDM.

For these models, the approach as been to try to find the relationship between the amount of fitting data, and the type of fitting data, to the accuracy and uncertainty of the results. It is important that the terms used in a model to predict unknown mass regions are well anchored in physics. Adding more terms with no clear physical meaning may increase the accuracy to predict already known data, but it can also introduce added uncertainty when extrapolating to unknown regions. To examine this, a massive random sampling approach has been used, where each model is fitted to a different sample size with random nuclei from the total pool of $n_{\text{tot}} = 2305$ good experimental binding energies in the AME2016 database. This section has been submitted for review to the journal Physical Review C.

The conditions for good nuclear data are defined here as:

\[
\begin{align*}
Z & \geq 6 \\
A & \geq 12 \\
\text{uncertainty} \cdot A & < 100keV/A
\end{align*}
\]  

Experimental Data

The sample size is defined in this work as a fraction of the total number of good experimental binding energies.

\[
f = \frac{n}{n_{\text{tot}}}
\]

Where the fractions considered are $f = 0.005$ to 0.6 for LDM and dz10, $f = 0.015$ to 0.6 for dz19, and dz33, and $f = 1$ for all models. The parameters of the model are fitted using the random sample, and the RMS deviations of the binding energies from the training sample ($\sigma_s$) and from all data ($\sigma_a$), along with binding energies and parameters values are collected for each calculation. For each fraction over 10,000 calculations were performed with random training data.

The binding energies of each DZ model with $f = 1$, compared to HFB+THO, are displayed in figure 3.1 as a function of the neutron number $N$. 

23
It can be seen clearly that DZ33 predicts the experimental binding energies better than the other models. The question is how well it does when extrapolating to unknown regions. This is explored by using smaller fractions of fitting data, and then comparing the results to all known experimental data.
Figure 3.2 illustrates how the RMS deviation of the fit to the sample and to all data changes with fraction size.

It is interesting to note how this figure shows that it’s possible to construct a minimal model with good accuracy comparable to that of the fully fitted model with a fraction as small as $f = 0.1$ of the known data for the models with 6, 10, and 19 parameters. However for the DZ33 model a fraction of around 0.4 is necessary to reach the same level of convergence. Though to construct this minimal model in practice, one would need to know ahead of time which sample will yield the best fit, and that composition of nuclei is unknown before the massive sampling is completed. In this case around 10000 calculations were performed for each fraction.

As a test of our method figure 3.3 shows the convergence of the results when fewer calculations are used for the DZ-10 model.
As can be seen the minimal RMS error converges with larger fraction sizes and with a higher number of calculations performed, as expected. This is a sign that the massive random sampling approach is valid, it also shows that more calculations than 10,000 would be preferable if smaller fractions are used. The uncertainty of the models are further illustrated in figure 3.4.
Figure 3.4: The distributions of RMS deviations from the sample (blue) and all data (yellow) for a small fraction comparable to roughly three times the number of parameters in the model.

It is noticeable that the DZ33 model has a significantly wider distribution of deviations than the other DZ models. This can be assumed to be because the model has a greater number of parameters, some of which are not clearly based on physics. On the other hand, the reduced DZ19 model shows a narrow distribution on par with or even better than the DZ10 model. This suggests that while the model with more parameters perform the best to predict the binding energies of the known experimental data used in the fit, we should use a more minimized model for extrapolating to unknown regions. This is also increasingly important as we get closer to the neutron drip lines where pairing forces become increasingly important and the r-process is active.
To see how the uncertainties shape the results of the calculations, a comparison of binding energy predictions for different fractions are shown in figure 3.5 for the recently measured $^{55-57}\text{Ca}$ isotopes [22], and for the experimentally unreachable $^{154-156}\text{Sn}$ and $^{175-177}\text{Sn}$ isotopes at and around the one and two neutron drip lines found by the HFB+THO calculations in this work. In the DZ models the results shown are the results given by the best fit for the given fraction, that is the fit with the minimal RMS deviation $\sigma_\alpha$.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{ca_binding_energy_residual.png}
\caption{\textit{55–57 Ca} Binding Energy Residual}
\end{figure}
Figure 3.5: a) The binding energy residuals (a) and binding energy predictions (b, c) for four fits with different fractions for selected Ca and Sn isotopes.
Figure 3.5 shows that DZ33 indeed has a greater uncertainty in the results than the other models, and especially striking is the large differences between the binding energies of the different fractions as we venture further out from stability to the even neutron drip line. It is also noteworthy that the spread of the results from DZ10 and DZ19 stay almost constant.

One can wonder what the optimal composition of the sample is to capture the essential physics and to get the best and (or worst) fits out of each model. Presumably a mix of light and heavy nuclei is needed from all over the nuclear chart. But perhaps there could be patterns of stable or unstable nuclei that could influence the fit more than others. Here two approaches to this has been taken, the first is to track all the nuclei used in the random sampling and identify the 100 samples which gives the best and worst result when predicting all data. This is illustrated in figure 3.6 to figure 3.9.
Figure 3.6: The nuclei used for the 100 best and worst fits for the LDM model and fraction $f = 0.85$, $f = 0.15$, and $f = 0.39$. For the Zoomed figures, the counts in one interesting area is given.
For every massive sampling with different fractions, the same pattern appears. It seems that the accuracy of the LDM model is dependent on fitting, or in this case not over fitting to certain portions of the nuclear chart. In particular the models seems to perform the worst when over fitting to areas around the doubly magic numbers Z = 50, N = 82, and Z = 82, 126. One wonders then if the DZ models have similar correlations between accuracy and certain regions.

For DZ10 and DZ19 a similar pattern appears, but for different regions than the LDM. One of these regions is emphasized in figure 3.7 and is located around the magic numbers Z = 28, N = 50. It is interesting to note that the DZ10, and DZ19, seen in figure 3.8 have the same correlations for several of the areas of interest, which is natural since the models are not that different, but for the DZ33 model seen in figure 3.9 there are two striking differences.

The first is that the model is no longer dependent on that area around N = 30, and N = 50, and the second is that another other interesting area shown in DZ10, and DZ19 around Z = 20, N = 30, where that area if under fitted leads to maximal errors, now has the opposite correlation in DZ33. Where now an over fit of that area leads to maximal errors, and an under fit leads to a better result.
Figure 3.7: The nuclei used for the 100 best and worst fits for the DZ10 model and fraction $f = 0.085$, $f = 0.15$, and $f = 0.23$. For the Zoomed figures, the counts in one interesting area is given.
Figure 3.8: The nuclei used for the 100 best and worst fits for the DZ19 model and fraction $f = 0.085$, $f = 0.15$, and $f = 0.23$. For the Zoomed figures, the counts in one interesting area is given.
Figure 3.9: The nuclei used for the 100 best and worst 100 fits for the DZ33 model and fraction \( f = 0.085 \), \( f = 0.15 \), and \( f = 0.23 \). For the Zoomed figures, the counts in one interesting area is given.
These results may be explained by the added terms in the model, it seems that the new terms are addressing some problems with the reduced models, and thereby decreasing the dependency of those models to certain parts of the nuclear chart. On the other hand they may be overcompensating in another area which instead of fixing the problem, reverses it. This method may be possible to use to investigate the impact of the parameters on the nuclear chart and used as a part of improving the models for the future, it is also possible that these relations are artifacts of the method used. In the future it is planned to investigate this further by using a more sophisticated machine learning algorithm, but it is outside the scope of this work.

A second approach to figuring out the correlations between the accuracy of the model and to the nuclei used for the fit was to randomize the samples based on their instability (measured by their half-life), and to track the RMS deviation as a function of the fraction and half-life. Figure 3.10 shows as expected that a mix of short lived and long lived nuclei are needed.
In the figure different weights have been put on nuclei with shorter and longer half lives, where 0% means that 0 nuclei used in the fit have a half life longer than the specified limit on the x axis. The last point on the x axis plot is means and means that a random sample from all nuclei has been used for that fit. It can be deduced from figure 3.10 that for deviations from the training sample the best fits are found when only long lived nuclei are used and that the fits get progressively worse as more unstable isotopes are introduces into the mix. This is expected since it is easier to nail down the physics of only stable and less deformed elements. Sub figures c) and d) show the minimal and maximal RMS deviations of the DZ10 model when fitted to all data, in effect measuring the extrapolation power of the model. As can be seen the best performance is given when there is a mix of stable and unstable elements, and if too much weight is put on either end of the spectrum, the predictive capability quickly deteriorates. The same relationship is found for smaller fractions.
It is also interesting to see whether there is a correlation between the power of a model to accurately predict the training sample binding energies and to accurately predict all data. The correlations between the RMS deviations of the sample and all data are given in figure 3.11.

![Scatter plots of σ_a and σ_s for various fractions for all four models.](image)

Figure 3.11: Correlations of $\sigma_a$ and $\sigma_s$ for a few fractions for all four models.

In this figure, it is shown to be no correlation at all. However when viewed in the light of figure 3.10 one may expect a negative correlation, since according to figure 3.10b the model that is best at predicting it’s own training data is also one with mostly stable nuclei in the sample. In contrast figure 3.10b clearly shows that a model loses its predictive power as it is fitted to only nuclei with progressively longer half lives. If a correlation exists, then the signal is much too weak to be visible here. Since a massive random sampling of all nuclei is used, that is not surprising because even the best samples will include a mix of long lived and short lived nuclei. This means that to look for this correlation a more skewed set of data is required. The result of this analysis can be seen in figure 3.12.
In this figure it is clear that when the training samples are restricted by stability of the nuclei, the expected indirect correlation emerges. In figure 3.12a it is noticeable that as the training samples change from only very short lived, to a mix of all half lives, both $\sigma_s$ and $\sigma_a$ decreases. In figure 3.12b the training samples changes from a mix of short and long half lives to strictly long half lives, which makes it easier to describe the physics of the sample but reduces the models ability to predict the whole body of data.
The massive sampling method also enables the possibility to examine the uncertainties of the parameters of each model. For this purpose every calculated value for each parameter and fraction has been recorded and the mean value and deviation derived by observing that the calculated values are normally distributed. The parameter values are plotted in figure 3.13 to 3.16 for the LDM, DZ10, DZ19, and DZ33 respectively.

Figure 3.13: The parameter value distributions of the LDM fitted to a Gaussian for each fraction (left), and the mean value and deviations ($\pm \sigma_N$, $\pm 2\sigma_N$, right)

As shown here the mean values of the parameters remain fairly constant except at very small fractions, and the parameters have a constant sign except for the surface symmetry, and surface Coulomb terms which have a small probability of a sign change. All six terms show a large and similar uncertainty at low fractions.
The first five parameters of the DZ10 model are similar to the macroscopic terms of the LDM, but the last five are added for microscopic shell correction, and the last term which is the pairing energy correction. The mean values are again fairly constant but it is noticeable that the variance of around 50% of the value of the added shell correction and pairing terms are much larger than the variance of the first five terms. For this model there is a small probability of a sign change for the $H_d$ term, but the other terms have a constant sign.
Figure 3.15: Same as figure 3.13 but for DZ19
The parameters of the DZ19 mostly show a small variance, however the $d_0$, and $D_0$ parameters are very ill defined at low fractions and show a huge variance. The signs of these two parameters as well as for $sq$—but to a lesser extent, are all not well defined.
In the DZ33 model, another 13 shell correction terms and a surface pairing term are introduced. Many of these terms show a large variance and at small fractions these parameters can hardly be pinned down at all. This shows us that although the overall bias of the model decreases and the accuracy increases as the model becomes more complex, the variance can increase substantially. The signs of the newly introduced parameters are ill defined and the large variance of up to 4 magnitudes can explain the models inability to make reliable predictions as discussed earlier.
Chapter 4

Summary and Discussions

The difficulty of making accurate predictions of nuclear properties in regions outside of experimental reach is well known. In this work tools have been developed to facilitate easier use of the HFB+THO code for calculations and data extraction. The blocking scheme has been implemented for a large range of nuclei to get accurate drip lines and values of binding energy of neutron heavy nuclei, for the purpose of future r-process network calculations. In relation to this the extrapolations of the HFB+THO model were compared to the not fully microscopic LDM, DZ-10, DZ-19, and DZ-33 models. A method was proposed to optimize these models by fitting to a large number of random samplings and evaluating the results statistically. It was found that it is possible to construct a model that describes all experimental data with a minimal amount of fitting data, the experimentally known binding energies are around 30% of the estimated amount of possible nuclei. It is shown here that all models are highly sensitive to fitting data at and around the doubly magic numbers around 20, 28, 50, 82, and 126 for $Z \neq N$. This could mean that possible magic numbers higher than 126 that are inaccessible to our measurements could be very important for pinning down the physics of the entire theoretical nuclear chart. The uncertainties of the parameters are shown to be strongly linked to the complexity of the model. A greater number of parameters offers a better flexibility to known data, but causes an increase in the variance when limited data is available for the fit. This severely limits the predictive power of the more complex DZ-33 model, instead a more minimal model should be used. The DZ-10 and DZ-19 models both show a smaller variance when extrapolating to unknown regions, as does the LDM but that model is limited in more fundamental ways. The signs of the parameters are mostly well defined, but the LDM the two surface correction terms and similar terms in the DZ-10, and DZ-19 models exhibit large uncertainties and can change sign. The mean value of all models parameters and predictions stay fairly constant even at very low fractions. This should make it possible to use this random sampling method to improve accuracy of the models. The predictions of neutron rich Ca compared to new experimental data, and Sn isotopes close to the neutron drip lines were used to investigate the variance of the model predictions.

It was a conscious choice to include the very different HFB model to the LDM, and DZ models in this work. Perhaps this random sampling method can be applied to the fitting of the Skyrme parameters, or perhaps the knowledge gained from investigating the importance of certain regions of the nuclear chart can guide the fitting even for that model. However to do this the calculations of the HFB+THO must be sped up.
significantly. Some steps were taken to investigate how much time could be saved from stopping calculations prematurely, and it seems possible to discard many calculations even after only 5 iterations. A conservative estimate is that it should be able to cut the time in half with this method. However to be able to compute the entire nuclear chart at least another factor of 100 is preferable. To accomplish this, work has begun to implement GPU calculations with CUDA for the HFB+THO code.
Included here is a minimal version of the python codes used for the HFB+THO calculations. A full code with additional functionality and debugging will be posted online at http://www.nuclear.kth.se/cqi/mm/.

To run HFB+THO:

```python
import subprocess
import sys

def get_list_from_row(text):
    while ' ' in text:
        text = text.replace(' ', ' ')
    text = text.replace('
', ' ').strip().split(' ')
    return text

def read_iso(isolist):
    with open(isolist) as file:
        nuclides = []
        for line in file:
            n = get_list_from_row(line)
            nuclides.append([n[0], n[1]])
        return nuclides

def set_params(n, d, v):
    nuclide = n
    v_pair = v

    with open('hfbtho_NAMELIST_original.dat') as file:
        textstr = '

        for line in file:
            if 'basis_deformation' in line:
                ind = line.index('basis_deformation = ')
                textstr += line[0:ind+20] + str(d) + '

            elif 'proton_number' in line:
                ind = line.index('proton_number = ')
                ind2 = line.index('neutron_number = ')
                textstr += line[0:ind+16] + str(nuclide[0]) + line[ind2-2:ind2+17] + str(nuclide[1]) + line[ind3:len(line)]

            elif 'vpair_' in line:
                ind1 = line.index('user_pairing = ')
                ind2 = line.index('vpair_p = ')
                textstr += line[0:ind1+15] + str(v_pair[2]) + line[ind1+16 : ind1+16 + len(str(v_pair[0]))] + str(v_pair[1]) + line[ind2+10+len(str(v_pair[0]))] + str(v_pair[0]) + line[ind2+10+len(str(v_pair[0]))]

            else:
                textstr = textstr + line

        text_file = open('hfbtho_NAMELIST.dat', 'w')
        text_file.write(textstr)
        text_file.close()

    def set_blocking(b):
        block_level = b

        with open('hfbtho_NAMELIST.dat') as file:
            for line in file:
                if 'neutron_blocking' in line:
                    block_level = line.split('neutron_blocking = ')[1].join(block_level) + '

                    textstr += pl[0] + 'neutron_blocking = ' + block_level
                else:
                    textstr += line
```

Chapter 5
Appendix
text_file = open('hfbtho_NAMELIST.dat', 'w')
text_file.write(text_str)
text_file.close()

def read_blockinfo(ee):
    ee_dat = ee
text = ''
with open(ee_dat) as file:
    text = file.read()
blocking = text.split('*HFB+THO>').[1].split('proton eqpmin')[0].split('Blocking candidates are:')[1].split('num=')[0].split('
')
blocking.pop(0)
blist = [
    for b in blocking: #
        bl = get_list_from_row(b) [12:]
        if '-' in bl[0]:
            w = bl[0].split('*+')[0]
pair = '*1'
        elif '*' in bl[0]:
            w = bl[0].split('-*')[0]
pair = '*-1'
        bl = ''.join(bl[1:]).replace(']*', '').split(')')
        block = [
            block.append(w)
            pair = ' ' .join(bl[1:]).replace('] ' , ' ').split(' , ')
        block.append(pair)
        for i in bl: 
            block.append(pair)
        blist.append(block)
return blist

def new_calc(f):
    filename = f
    o = ''
    print(filename)
    output = str(subprocess.check_output(['ls', '/results' , '-1'])))
    if filename[8] in output:
        return False
    else:
        return True

def run_HFBTHO(n , v , deform , block_levels):
    nuc = n
    v_pair = v
    i = deform
    blockprint = ''
    filename = ''
    set_params(nuc , i , v_pair)
    set_blocking('0 , 0 , 0 , 0 , 0 , 0')
    filename = 'results/ + str(nuc[0]) + '_ + str(nuc[1]) + '_def_' + str(i) + '.dat'
    if new_calc(filename) is True:
        subprocess.call(['cp', 'thout.dat', filename])
        block = read_blockinfo('thout.dat')
    else:
        block = read_blockinfo(filename)
        for b in range(len(block)):
            if b >= block_levels:
                break
            set_blocking(block[b])
            blockprint = ''.join(block[b])
            filename = 'results/ + str(nuc[0]) + '_ + str(int(nuc[1]) + 1) + '_def_' + str(i) + '_level_' + blockprint + '.dat'
            if new_calc(filename) is True:
                subprocess.call(['cp', 'thout.dat', filename])
    def find_nth(haystack , needle , n):
        start = haystack.find(needle)
        while start >= 0 and n > 1:
            start = haystack.find(needle , start + len(needle))
            n -= 1
        return int(start)
v_pair = ['3287.85' , '3287.85' , 'T']
deformation_states = ['0.3' , '-0.2' , '-0.1' , '0.0' , '0.1' , '0.2' , '0.3']
nuc = read_iso('iso_even.dat')
block_levels = 1
for k in nuc:
    for i in deformation_states:
        run_HFBTHO(k , v_pair , i , block_levels)

To analyze the results:

import sys
import subprocess
import random
from timeit import default_timer as timer

class nuclide:
    def __init__(self , protons , neutrons , deformation , blocking , energy ,
        filename , blocklist , convergence , ediffiter , iterationtime , totaltime):
        self.z = protons
        self.n = neutrons
self.d = deformation
self.b = blocking
self.bl = blacklist
self.bn = 0
self.e = energy
self.ediff = 0.0
self.s = 0
self.f = filename
self.c = convergence
self.et = iterationtime
self.t = totaltime
def __repr__(self):
    return str(self.z) + " " + str(self.n) + " " + str(self.d) + " " + str(self.b) + " " + str(self.bn) + " " + str(self.e) + " " + str(self.s) + " " + str(self.ediff) + " " + str(self.c) + "n"
def get_Energy(inp, rat = ' '):
    E = ' '
    text = ' '
    conv = 'yes'
    context = ' '
t = 0
tfirst = 0
for line in inp:
    if "CPU time=" in line:
        if t == 0:
            tfirst = float(get_list_from_row(line)[3])
        t += float(get_list_from_row(line)[3])
    text += line
    textfind = "FORTRAN 95 CODE"
    if find_nth(text, textfind, 2) < 0 and find_nth(text, '"Energy: ehfb', 3) < 0 and "[HFB+THO]\"" and "Energy: ehfb (qp)\"" and "Calculated but\"" in text:
        hfbtext = text.split("[HFB+THO]\""),[0]
        text = text.split("[HFB+THO]\""),[1]
    if "iterations limit interrupt\"" in text:
        print('[HFB+THO] non convergence')
        conv = 'no'
    E_it, itt = check_iteration_energy(text, rat)
    text = text.split("[Calculated but]\""),[0]
    for i in range(0, len(text)):
        if text[i] != " ":
            E += text[i]
    E = E[:(-2)]
    return E, conv, E_it, itt + tfirst*60, t*60
else:
    return 0, conv, 0, 0, 0
def analyze_corrupted(directory):
    #Opens corrupted files and splits them into correct output files if the calculations in them are complete
corruptnucs = create_nuc_list(directory)
print(corruptnucs)
for i in corruptnucs:
    text = ""
    count = 0
    with open(i.f) as file:
        for line in file:
            if "FORTRAN 95 CODE\"" in line:
                count += 1
            if count == 2:
                newfile = open(str(i.z) + " " + str(i.n - 1) + " " + str(i.d) + " " + str(i.bn) + " " + str(i.e) + " " + str(i.s) + " " + str(i.ediff) + " " + str(i.c) + "n", "w")
                newfile.write(text)
                newfile.close()
                text = ""
                newfile = open(i.f, "w")
                newfile.write(text)
                newfile.close()
subprocess.call(['mv', i.f, '.'])
def get_min(inp):
    out = []
    if len(inp) is 1:
        return inp
    for i in inp:
        if i.c == 'no':
            continue
    emin = i
    for j in inp:
        if i is not j:
            if i.z == j.z and i.n == j.n:
                if float(j.e) < float(emin.e):
                    emin = j
if emin not in out:
    out.append(emin)
for i in out:
    if i.bn > 1:
        for j in inp:
            if i.z == j.z and i.n == j.n - 1 and i.d == j.d:
            out
blocklist = j.bl
break
for b in range(0, len(blocklist)):
    txt = " "
    for bt in blocklist[b]:
        txt += bt
    blocklist[b] = txt
n = 0
found_lowest = False
while found_lowest is False:
    for l in inp:
        if l.i is not i:
            if l.z == i.z and l.n == i.n and l.d == i.d:
                i.ediff = float(i.e) - float(l.e)
                found_lowest = True
                break
            n += 1
        if n >= len(blocklist):
            break
massfile = open(‘masses.dat’, “w”)
for i in out:
    i.m = (i.z∗(m_p + m_e) + i.n∗m_n + float(i.e))
    i.dm = (i.m - (i.z + i.n)∗amu)∗1000
massprint = [i.z, i.n, i.m, i.dm]
massfile.write(" { >5} { >5} { >22} { >22}".format(*massprint) + "n")
massfile.close()
return sort_results(out)
def sort_results(inp):
    inp = sorted(inp, key = lambda nuclide: (nuclide.z, nuclide.n, nuclide.d))
    return inp
def print_results(inppt, filename):
    text_file = open(filename, ‘w’)
text_file.write("{: >5} {: >5} {: >10} {: >8} {: >12} {: >22} {: >22} {: >8}"
                .format(*textprint) + "n"
                )
datafile = open(‘pairingbeta.dat’, ‘w’)
datafile.write("Z + * + N* + *

        + *pairing energy *+

        + *beta* + *n"
                )
for ip in inppt:
    textprint = [str(ip.z), str(ip.n), str(ip.d), str(ip.h), str(ip.bu),
                str(ip.e), str(ip.ediff), ip.c]
text_file.write("{: >5} {: >5} {: >10} {: >8} {: >12} {: >22} {: >22} {: >8}".format(*textprint) + "n"
                )
datafile.write(str(ip.z) + " " + str(ip.n) + " 

        " + str(ip.e) + " 

        "
                )
for k in range(0, len(ip.pe)):
    datafile.write(str(ip.pe[k]) + " *=")
datafile.write(\"n\n")
datafile.close()
text_file.close()
def print_separation_energies(inp): #must be sorted
    text_file = open(‘separation_energies.dat’, ‘w’)
    for i in range(0, len(inp)):
        E = 0
        if inp[i].n % 2 is 0 and i > 1:
            if inp[i].s is inp[i-2].x:
                E = float(inp[i-2].e) - float(inp[i].e)
            text_file.write(str(inp[i].x) + " * + str(inp[i].n) + * +

        str(E) + "n"
        inp[i].s = E
        elif inp[i].n % 2 is 0 and i > 0:
            if inp[i].s is inp[i-1].x:
                E = float(inp[i-1].e) - float(inp[i].e)
            text_file.write(str(inp[i].x) + " * + str(inp[i].n) + * +

        str(E) + "n"
        inp[i].s = E
text_file.close()
def print_neutron_driplines(inp):
    one_file = open(‘one_neutron_driplines.dat’, ‘w’)
two_file = open(‘two_neutron_driplines.dat’, ‘w’)
for i in range(0, len(inp) - 2):
    if inp[i].n % 2 is not 0:
        if inp[i].x > 0 and inp[i+1].x < 0 and inp[i].z is inp[i+1].z:
            one_file.write(str(inp[i].x) + " * + str(inp[i].n) + * +

        str(E) + "n"
        elif inp[i].n % 2 is 0:
            if inp[i].x > 0 and inp[i+1].x < 0 and inp[i].z is inp[i+1].z:
                two_file.write(str(inp[i].x) + " * + str(inp[i].n) + "

        one_file.close()
two_file.close()
def check_outputfiles(inp): #checks for corrupted and empty files
    create_start = timer()
    filelist = create_nucliset(inp)
    create_stop = timer()
    print("Create nucliset time: " + str(create_stop – create_start))
corrupted = 0
    empty = 0
    checked_files = ‘ ‘
    checkdir = str(subprocess.check_output(‘ls ’))
if "corrupted" not in checkdir:
    subprocess.call(['mkdir', 'corrupted'])
if "empty_files" not in checkdir:
    subprocess.call(['mkdir', 'empty_files'])

i = 0
if "checked_files.dat" in checkdir:
    with open('checked_files.dat') as file:
        checked_filelist = file.read()
else:
    checked_filelist = ''
for i in filelist:
    text = ''
    if i.f in checked_filelist:
        continue
    with open(i.f) as file:
        text = file.read()
    k = find_nth(text, '|HFB + THO>', 2)
    if "is corrupted!" in text and i.n % 2 is not 0 and \
        find_nth(text, "tEnergy: ehfb", 4) >= 0:
        if k >= 0:
            corrupted = 1
            print("Corrupted file moved to corrupted/ for analysis: " + i.f)
        continue
    elif "hel is missing!" in text and i.n % 2 is not 0 and \
        find_nth(text, "tEnergy: ehfb", 4) >= 0:
        if k >= 0:
            corrupted = 1
            print("Corrupted file moved to corrupted/ for analysis: " + i.f)
        continue
    if "tEnergy" not in text or "|HFB + THO>" not in text:
        subprocess.call(["mv", i.f, "empty_files"])
    if "FORTRAN 95 CODE" in text and find_nth(text, "tEnergy: ehfb", 4) < 0:
        print("Corrupted file or file with .hel missing moved to empty_files:
            " + i.f)
        empty = 1
        subprocess.call(["mv", i.f, "empty_files"])
    filelist.append(i)
    checked_files += i.f

text_file = open('checked_files.dat', 'w')
text_file.write(checked_files)
text_file.close()
if corrupted is 1:
    print("Corrupted files have been moved to corrupted/
        fixing files . . . . .")
    analyze_corrupted('corrupted')
if empty is 1:
    print("Empty files have been moved to empty_files/*")
return filelist

def create_nuclei(inp, rat = ' '):
    allnuclei = []
    oldfilelist = '
    findnum = 1
    findnum2 = 1
    for char in inp:
        if char is ":":
            findnum += 1
        if char is "/":
            findnum2 += 1
    filelist = make_filelist(inp)
for i in allnuclei:
    oldfilelist += i.f
j = 0
x = 0
n = 0
deformation = []
blocklevel = []
Energy = []
k = 0

while k < len(filelist):
    if filelist[k] in oldfilelist:
        del filelist[k]
        k -= 1
    else:
        filelist[k] = inp + '/\' + filelist[k]
        k += 1
    for i in filelist:
        Energy = []
        blocklist = []
        if "_def_" in i:
            x = int(i[find_nth(i, '_', 1), findnum2 + 1 : find_nth(i, '"', findnum)])
        else:
            x = int(i[0 : find_nth(i, '"', 1)])
        n = int(i[find_nth(i, '"', findnum) + 1 : find_nth(i, '|', findnum + 1)])
        nuc = str(x) + '"' + str(n)
        if "level" in i:
            deformation = i[find_nth(i, '"', findnum + 2) + 1 : find_nth(i, '"', findnum + 3)]
            blocklevel = i[find_nth(i, '|level|', 1) + 6 : find_nth(i, '|.dat|', 1)]
        else:
            deformation = i[find_nth(i, '"', findnum + 2) + 1 : find_nth(i, '|.dat|', 1)]
            blocklevel = ""
with open(i) as file:
    (Energy, conv, Ediffiter, itt, t) = get_Energy(file, rat)
if n % 2 is 0:
    n += 1
    allnuclist.append(nuclide(z, n, deformation, blocklevel, Energy, i, blocklist, conv, Ediffiter, itt, t))
allnuclist = sort_results(allnuclist)
for i in allnuclist:
    if i.n % 2 is 0:
        for j in allnuclist:
            if j.n % 2 is not 0:
                if j.z == i.z and j.n == i.n + 1 and j.d == i.d:
                    for k in range(0, len(i.bl)):
                        if j.b[k] == i.bl[k]
                            if j.bn == 0:
                                j.bn = k + 1

with open(inp + '/nuclitelist.dat', 'w') as text_file:
    for i in allnuclist:
        text_file.write(str(i.z) + ' ' + str(i.n) + ' ' + str(i.d) + ' ' + str(i.b) + ' ' + str(i.bn) + ' ' + str(i.e) + ' ' + str(i.f[find_nth(i.f, '/', findnum2) + 1:]) + ' ' + str(i.c) + ' ' + str(i.eit) + ' 
')
        for j in range(0, len(i.bl)): text_file.write(i.bl[j] + ' ')
        text_file.write('
')
    text_file.close()
return allnuclist
def make_filelist(inp):
    filelist = str(subprocess.check_output(["ls", inp]))
    filelist = filelist[1:] + filelist[:1]
    filelist = filelist.strip('"
')
    filelist[0] = filelist[0][1:]
    del filelist[len(filelist) - 1]
    j = 0
    while j < len(filelist):
        if not def_ not in filelist[j]:
            def filelist[j]:
                j += 1
        return filelist
def fileexists(nuclist, filename):
    for i in nuclist:
        findnum = 0
        for char in i.f:
            if char is '/':
                findnum += 1
                if i.f[find_nth(i.f, '/', findnum) + 1:] == filename:
                    return True
        return False
def replace_empty_files():
    directory = '"empty_files"'
    nuclist = create_nuclist(directory)
    allnuclist = create_nuclist('"
')
    calcall = 0
    answer = '"startvalue"'
    splitrun = 0
    for i in nuclist:
        run = 0
        text = ""
        filename = str(i.z) + '"" + str(int(i.n - 1)) + '"_def_" + str(i.d) + '".dat"
        if not fileexists(allnuclist, filename) is False:
            continue
        with open(filename) as file:
            for line in file:
                text += line
        blocking = read_blockinfo(text)
        for ab in range(0, len(allnuclist)):
            if allnuclist[ab].n % 2 is not 0:
                if splitrun == 0:
                    allnuclist[ab].b = allnuclist[ab].b.split('"
')
                    splitrun = 1
                    if i.z != allnuclist[ab].z or i.n != allnuclist[ab].n or i.d != allnuclist[ab].d:
                        allnuclist[ab].d =
                        r = 0
                        badblock = '""
                        for m in range(0, len(allnuclist[ab])):
                            block1 = m
                            block2 = m
                            for n in range(0, len(blocking[1])):
                                block1 += blocking[1][n]
block2 += allnuclist[ab].bn

if n < len(blocking[r]) - 1:
    block2 += ",

if block2 == block2 or block2 == badblock:
    del blocking[r]
    r -= 1

r += 1

if answer is not "na":
    print("Calculations could not be performed: blocking candidate could not be found")
    return

if answer is "y" or answer is "ya":
    calcall = 1
    if i.n < 10:
        sn = "s00" + str(i.n - 1)
    elif i.n < 100:
        sn = "s0" + str(i.n - 1)
    else:
        sn = "s" + str(i.n - 1)

    else:
        sn = "s" + str(i.x)

    ifile = sn + "+" + str(i.d) + "_level_" + blocktext + ".dat"
    print(ifile)
    subprocess.call(['cp', ifile, '.'])

while run >= 0 and run < len(blocking):
    set_blocking(blocking[run])
    blocktext = ''
    for p in blocking[run]:
        blocktext += p
    filename2 = str(i.x) + "+" + str(i.x) + "_def_" + str(i.d) + "_level_" + blocktext + ".dat"

set_params([i.z, i.n - 1], i.d, [-287.85, -287.85, 'T'])

while run >= 0 and run < len(blocking):
    set_blocking(blocking[run])
    blocktext = ''
    for p in blocking[run]:
        blocktext += p
    filename2 = str(i.x) + "+" + str(i.x) + "_def_" + str(i.d) + "_level_" + blocktext + ".dat"
    subprocess.call(['cp', ifile, '.'])
    subprocess.call(['mv', ifile, 'empty_files/processed_empty_files'])

run += 1

if answer is 'na':
    calcall = 1
    if i.n < 10:
        sn = 's00' + str(i.n - 1)
    elif i.n < 100:
        sn = 's0' + str(i.n - 1)
    else:
        sn = 's' + str(i.n - 1)

    ifile = sn + "+" + str(i.d) + "_hel"
    print(ifile)
    subprocess.call(['cp', ifile, '.'])
    subprocess.call(['mv', ifile, 'empty_files/processed_empty_files'])

def set_iterations(it):
    with open('hfbtho_NAMELIST.dat') as file:
        textstr = ''
        for line in file:
            if 'number_iterations' in line:
                textstr += line
        else:
            textstr += line
        text_file = open('hfbtho_NAMELIST.dat', 'w')
        text_file.write(textstr)
        text_file.close()
def count_def_block():
    allnuc = create_nuclis(1)
    return allnuc
else:
    return []

def compare_with_exp():
    nuclist = create_nuclis(1)
    nuclist = get_min(nuclist)
    print_separation_energies(nuclist)
    exp = file.readlines()
    for exp in exp:
        rowlist = get_list_from_row(i)
        if "1000" in rowlist[5]:
            continue
        n = int(rowlist[0])
        z = int(rowlist[1])
        bea = float(rowlist[4])
        a = int(rowlist[2])
        exp[append(nuclide(z, n, 0, 0, bea * a / 1000, 0, [], "yes", 0))]
    exp = sort_result(explist)
    comparefile = open("expcompare.dat", "w")
    printlist = ["Z", "N", "AME2016", "HFB+THO", "Difference", "Error % from exp value", "Total binding energy (MeV)""]
    comparefile.write("{:<5} {:<5} {:<20} {:<10} {:<20} {:<24} {:<24}")
    for i in nuclist:
        for j in exp:
            if i.z == j.z and i.n == j.n:
                diff = -j.e - float(i.e)
                printlist = [i.z, i.n, j.e, str(float(i.e)), diff, diff/j.e]
                comparefile.write("{:<5} {:<5} {:<20} {:<10} {:<20} {:<24} {:<24}")
                break
    comparefile.close()

def get_data(nuclist):
    spnum = 50
    spfiles = str(subprocess.check_output(["ls"]))
    for i in range(0, len(nuclist)):
        spfile = str(nuclist[i].z) + "_" + str(nuclist[i].n) + "_spe.dat"
        fermi = int(round(nuclist[i].n/2.0 - 0.5))
        sp = []
        spp = []
        with open(nuclist[i].f) as file:
            text = file.read()
            text1 = text.split("|HFB + THO> ")[1].split(" dipole moment")
            [0].split("n")
        for j in text1:
            if "pairing energy" in j:
                pair = get_list_from_row(j)
            elif "deformation beta2" in j:
                beta = get_list_from_row(j)
            nuclist[i].pe = [pair[3], pair[4], pair[5]]
            nuclist[i].beta = [beta[2], beta[3], beta[4]]
        neutronsp = text.split("|HFB + THO> ")[1].split("quasiparticle energies neutrons")[1]
        .split("labels")[1].split("all")[0].split("n")
        protonsp = text.split("|HFB + THO> ")[1].split("quasiparticle energies protons")[1]
        .split("labels")[1].split("all")[0].split("n")
    del neutronsp[-1]
    del neutronsp[0]
    del protonsp[-1]
    del protonsp[0]
    for j in neutronsp:
        add = get_list_from_row(j)
        sp.append(add)
    for k in protonsp:
        addp = get_list_from_row(k)
        spp.append(addp)
        for e in range(0, len(sp)):
            sp[e] = [float(sp[e][3]), sp[e][9:]]
        for ep in range(0, len(spp)):
            spp[ep] = [float(spp[ep][3]), spp[ep][9:]]
        sp = sorted(sp)
        spp = sorted(spp)
    newsp = []
    text_file = open(str(nuclist[i].z) + "_" + str(nuclist[i].n) + "_nspe.dat", "w")
    for f in range(0, len(sp)):
        if abs(fermi - f) <= 50:
            newsp.append(sp[f])
            text_file.write(str(sp[f][0]) + " * + str(sp[f][1]) + "\n")
    text_file.close()
    newsp = []
    text_file = open(str(nuclist[i].z) + "_" + str(nuclist[i].n) + "_nspe.dat", "w")
    for f in range(0, len(spp)):
        if abs(fermi - f) <= 50:
            newsp.append(spp[f])
            text_file.write(str(spp[f][0]) + " * + str(spp[f][1]) + "\n")
    text_file.close()
    nuclist[i].sp = newsp
    nuclist[i].spn = newsp

def count_def_block():
    allnuc = create_nuclis(1)
    return allnuc
else:
    return []
nuc = 0
countfile = open('count_d_b.dat', 'w')
countwrite = ['Z', 'N', '#def', '#block [-0.1, -0.2, (-0.3), 0.0, 0.1, 0.2, 0.3]', 'Total calc']
countfile.write('{:>5} {:>5} {:>8} {:>60} {:>15}').format(*countwrite) + '\n'
cdef = 0
cblock = 0
while nuc < len(allnuc):
cblocktext = ''
totblock = 0
if nuc == 0:
cdef += 1
nuc += 1
continue
while nuc < len(allnuc) and allnuc[nuc].n == allnuc[nuc - 1].n and allnuc[nuc].z == allnuc[nuc - 1].z:
    cblocktext += ' ' + str(cblock)
cdef += 1
totblock += cblock
cblock = 1
else:
cblock += 1
nuc += 1
totcount = totblock*cdef + cblock*cdef
if allnuc[nuc].n % 2 != 0:
countwrite = [str(allnuc[nuc - 1].z), str(allnuc[nuc - 1].n), str(cdef), str(cblocktext) + ' ' + str(cblock), str(totcount)]
countfile.write('{:>5} {:>5} {:>8} {:>60} {:>15}').format(*countwrite) + '\n'
cdef = 1
cblock = 1
cblocktext = ''
countfile.close()

def print_exp_count():
countprint = open('countprint.dat', 'w')
with open('expcompare.dat') as file:
exptext = file.readlines()
del exptext[0]
with open('count_d_b.dat') as file:
counttext = file.readlines()
for e in range(0, len(exptext)):
    for c in counttext:
        if get_list_from_row(exptext[e])[0] == get_list_from_row(c)[0] and 
        get_list_from_row(exptext[e])[1] == get_list_from_row(c)[1]:
            if float(get_list_from_row(c)[1]) % 2 == 0:
                countprint.write(get_list_from_row(c)[-1] + '\n')
                break
countprint.close()
m_p = 938.271998
m_e = 0.5109989
m_n = 939.565330
amu = 931.4940954
option = 'none'
while option is not 'q':
option = input('(1) Check files 
(2) Analyze Corrupted files 
(3) Print results: ')
References


