原子核の相対論的平均場模型プログラムの改良

2011年2月

福井大学 工学部 物理工学科
酒井　優・西川　恵子

目 次

第1章　序論

原子 (atom) は、ギリシャ語で「これ以上分割できないもの」という意味を持つ。19世紀、物質の最小単位と考えられてきた原子は、その後の研究によりさらに小さな物質に分けられることが分かった。まず、原子は原子核と電子から構成されており、さらに原子核は陽子と中性子に分けられるのである。陽子と中性子はまとめて核子と呼ばれ、核子の間には引力が働いている。核子同士の間には、中間子を媒介して核力という力が働く。この核力によって、原子核は安定しているのである。本論文では、相対論的平均場模型を用いて、原子核の特性についてさらなる研究を進める。

相対論を用いて原子核を記述する時には、中間子を古典的な場で置き換える。相対論では、中間子は核子同士で交換されているが、非相対論では中間子は粒ではなく広がった場として考える。核子を束縛するポテンシャルがあり、これを Dirac 方程式で決定する。これを Runge-Kutta 法で数値的に求めることで、局在した核子密度が求まる。核子密度をもとに Screened-Poisson 方程式で中間子の古典場を求め、最後に結合定数倍する。このループを自己無撞着になるまで繰り返す。これが、相対論的平均場模型である。

本研究では、三和氏の修士論文 [4] の原子核の相対論的平均場プログラムを改良をした。三和氏は、波動関数を等間隔に配置したグリッド点上の値で表現し、それらを Runge-Kutta 法により決定した。しかし、十分な精度を得るにはグリッド間隔を非常に密にとらなければならず、長い時間がかかった。将来的には、速い計算速度を求められるのは必然である。又、多数の核種を計算したり、変形を考慮するとなると、計算速度は重要となる。

そこで、グリッド間隔と誤差の関係を調べ、どこで大きな誤差が発生しているのかを調べた。そして、計算速度を短縮し、さらに精度のよいプログラムにするため、非等間隔グリッドを導入し、その効果を調べた。

今回の卒業研究では、通して共同で研究を行った。

執筆担当者
第２章、第４章、第５章　西川
第３章、第６章　酒井

第2章　特殊相対性理論

この章では、特殊相対性理論について説明する。本章の執筆に際しては、主として参考文献 [1] を参考にした。

特殊相対性理論は、1905 年にアルベルト・アインシュタインが発表した。それまで正しいと考えられてきたニュートン力学に、光速に近い速度になると矛盾が出てくることを発見した。

アインシュタインの特殊相対論は光速は２つの要請から導き出すことができる。１つ目は、基本的な物理法則は、すべての慣性系で同じ数学形式で表わせるというもの。２つ目は、光速は光源や観測者の運動に無関係で一定値 c である、また、どのような信号もエネルギーも光速より速い速度で伝搬することはないというものである。

2.1　Michelson-Morley の実験

ニュートン力学では、物体も光も見かけ上の速度は観測者の速度に比例すると考えられてきた。当時、光を伝える媒質としてエーテルという質量のない何ものかが存在存在すると考えられてきた。そして、このエーテルは絶対静止空間に固定されていて、地球はその空間の中を走っているとされていた。

Albert Michelson と Edward Morley は、地球の運動と静止しているエーテルの存在について実験を行った。これが、Michelson-Morley の実験である。

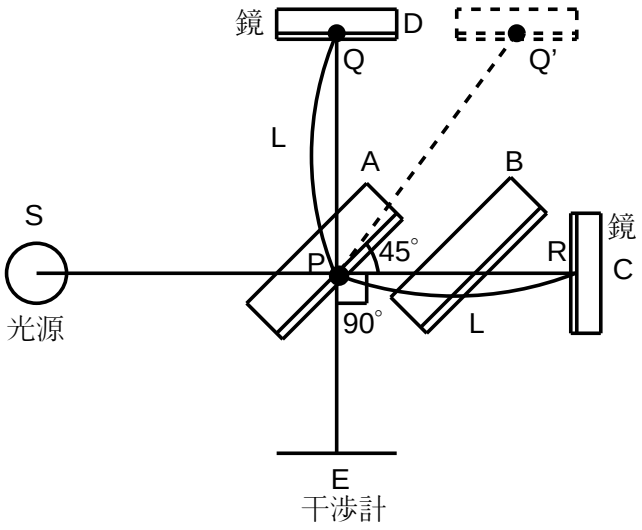


図 2.1: Michelson-Morley の実験装置

図 2.1 の A の裏面と C、D の表面には、薄く銀が塗ってある。これは、透過光と反射光が等しくなるようにするためである。A は光源 S から出た光に対して 45 °、C は垂直、D は平行となるように置かれている。PC と PD の距離は L だが、PD を往復する光はガラス A を通る分だけ光路が違ってしまうので、それを打ち消すためにガラス B をいれた。

光源 S から出た光は、一方はまず A を通過して C へ向かう。C の表面 R で反射された光は、A の裏面 P へ戻り、そこで 90 °反射されて干渉計 E へ入る。もう一方の P で反射された光は、D へ向かい Q で反射され、P に戻り直進して干渉計 E へ入る。

ここで、地球の絶対静止空間に対する速度を v, 光の速度を c とする。SR を地球の進行方向と平行にし光が PR を往復する時間を t_1 とすると、P から R に向かう光の速度は装置に対して $c-v$、R から P に向かう光の速度は $c+v$ なので、

$$t_1 = \frac{L}{c-v} + \frac{L}{c+v} = \frac{2cL}{c^2-v^2} \tag{2.1}$$

となる。一方、P から Q に向かった光は、Q に到達する間に地球が動いているため、Q は絶対静止空間内の Q′ へ移る。このため、光の PQ 方向への速度成分は、$sqrtc^2 - v^2$ となる。従って、光が PQ を往復する時間を t_2 とすると、

$$t_2 = \frac{2L}{\sqrt{c^2-v^2}} \tag{2.2}$$

となる。ここで、光速 c と比べて v が十分小さいと仮定すると、二つの光が干渉計に入る時間のずれ t_1-t_2 は、

$$t_1 - t_2 \simeq \frac{2L}{c}\left(1+\frac{v^2}{c^2}\right) - \frac{2L}{c}\left(1+\frac{v^2}{2c^2}\right) = \frac{L}{c}\left(\frac{v^2}{c^2}\right) \tag{2.3}$$

となる。地球が太陽のまわりをまわる速度は約 30km/s である。v=30km/s、L=11m（Michelson-Morley の実験では、実際に 11m であったことから）とすると、

$$t_1 - t_2 \approx 3.6 \times 10^{-16} s \tag{2.4}$$

となった。この時間差に光は 10^{-5}cm くらい進むはずなので、干渉計では、２つの光路を通った光が干渉し、その時間差が測定されるはずである。

しかし、この実験では時間差は検出されなかった。これによって、光の速度は地球が動く方向とは関係がないことが確かめられたのである。

2.2 特殊相対論

光速度が地球の運動と関係がないということは、ニュートンの力学を書き直さなければならないことになる。

ある座標系 S と、それに大して等速度 V で動く座標系 S′ を考える。S 座標系、S′ 座標系で測ったある質点の座標をそれぞれ (x, y, z)、(x', y', z') とする。古典力学の場合、この 2 つの座標には、

$$x' = x - Vt, y' = y, z' = z, t' = t \tag{2.5}$$

という関係になる。この変換を Galilei 変換という。

ニュートン力学の基本方程式には、座標の二階微分しか出てこないので、Galilei 変換をしても式の形は変わらない。式 (2.5) を時間について 2 回微分すると、

$$\ddot{x}' = \ddot{x}, \ddot{y}' = \ddot{y}, \ddot{z}' = \ddot{z}, \tag{2.6}$$

となる。これは、ある質点が持つ加速度は S と S′ の 2 つの座標系で同じである。よって、ニュートンの第に法則 $F = ma$ は、座標系 S,S′ どちらでも等しく成り立つ。もっと一般には、1 つの座標系に対して運動する座標系（慣性系）では、ニュートン力学は等しく成立し、これを古典的相対論という。

2.3 ローレンツ変換

Michelson-Morley の実験結果は、当時の物理学者に大きな衝撃を与えた。そして、その結果の説明しようと、様々な手段が考えられた。その中の 1 つが、ローレンツ変換である。

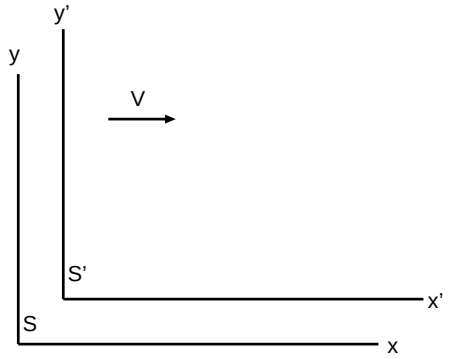


図 2.2: ローレンツ変換

2 つの座標系 S,S′ について考える。時刻 $t = 0$ に原点から放出された光は、時刻 t には半径 $r = ct$ の球面

$$x^2 + y^2 + z^2 = (ct^2) \tag{2.7}$$

を満たす点 (x,y,z) に達している。この方程式を式 (2.5) を用いて s′ 座標に変換すると、

$$(x' + Vt'^2) + y'^2 + z'^2 = (ct'^2) \tag{2.8}$$

となる。ここで、２つの座標系は同じ速度 c を持っていなければならない（Michelson-Morley の実験より）。そこで、式 (2.8) の代わりに

$$x'^2 + y'^2 + z'^2 = (ct'^2) \tag{2.9}$$

が成り立つようでなければいけない。そこで、式 (2.5) を

$$\begin{cases} x' = A(x - Vt) \\ y' = y \\ z' = z \\ t' = Cx + Dt \end{cases} \tag{2.10}$$

と置き換えることにする。式 (2.10) に式 (2.9) を代入すると、

$$(A^2 - c^2C^2)x^2 - 2(A^2V + c^2CD)xt + y^2 + z^2 = (c^2D^2 - A^2V^2)t^2 \tag{2.11}$$

となる。これを式 (2.7) に等しいとして解くと、

$$\begin{cases} A^2 - c^2C^2 = 1 \\ A^2V + c^2CD = 0 \\ c^2D^2 - A^2V^2 = c^2 \end{cases} \tag{2.12}$$

となるはずである。式 (2.12) の２番目の式を A^2 について解き、１番目の式と２番目の式に代入すると、

$$\begin{cases} C(D + VC) = -\frac{V}{c^2} \\ D(D + VC) = 1 \end{cases} \tag{2.13}$$

となる。これより

$$\frac{C}{D} = -\frac{V}{c^2} \tag{2.14}$$

という式が得られる。これを式 (2.13) に代入すると、

$$D^2 = \frac{1}{1 - \frac{V^2}{c^2}} \tag{2.15}$$

が求まる。これを、C、D について解き、平方に開いて正の値をとると

$$\begin{cases} C = -\frac{V}{c^2\sqrt{1 - \frac{V^2}{c^2}}} \\ D = \frac{1}{\sqrt{1 - \frac{V^2}{c^2}}} \end{cases} \tag{2.16}$$

を得られる。こうして、

$$\begin{cases} x' = \gamma(x - Vt) \\ y' = y \\ z' = z \\ t' = \gamma(-\frac{V}{c^2}x + t) \\ \gamma = \frac{1}{\sqrt{1 - \frac{V^2}{c^2}}} \end{cases} \tag{2.17}$$

が得られる。これが、ローレンツ変換である。

2.4　光のドップラー効果

本卒業研究では、相対論を学ぶ際に、特に光のドップラー効果について掘り下げて調べたので、以下ではそれについて述べる。

2.4.1　ドップラー効果について

救急車などのサイレンが、近づいてくるときは高く、遠ざかるときは低く聞こえることがある。これを、ドップラー効果という。これは、音源の進行方向では音の波長が短くなり、その反対方向では音の波長が長くなるためである。

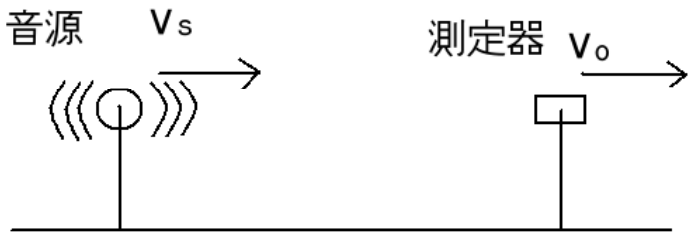


図 2.3: 音のドップラー効果

図 2.3 のように、音源、測定器それぞれ v_{S}、v_{O} で正の方向に進んでいるとする。音源は、周波数 f で周期 T ごとに音の波を出すとする。t=0 で出た音波は、t=T のとき VT だけ進むこととなる。また、音源は v_{O}T だけ進んでいるので、２つの音波の間隔を λ とすると、

$$\lambda = (V - V_{\mathrm{S}})T = \frac{V - V_{\mathrm{S}}}{f} \tag{2.18}$$

となる。測定器に届く音波の時間間隔を T′ とすると、観測器に対する波の速度は $v - v_{\mathrm{S}}$ なので、

$$T = \frac{\lambda'}{v - v_{\mathrm{S}}} = \frac{v - v_{\mathrm{S}}}{f(v - v_{\mathrm{S}})} \tag{2.19}$$

となる。周期 T=1/f なので、測定器での周波数 f′ は

$$f' = f\frac{V - v_{\mathrm{S}}}{V - v_{\mathrm{O}}} = f\frac{1 - \frac{v_{\mathrm{S}}}{V}}{1 - \frac{v_{\mathrm{O}}}{V}} \tag{2.20}$$

となる。

2.4.2　光のドップラー効果

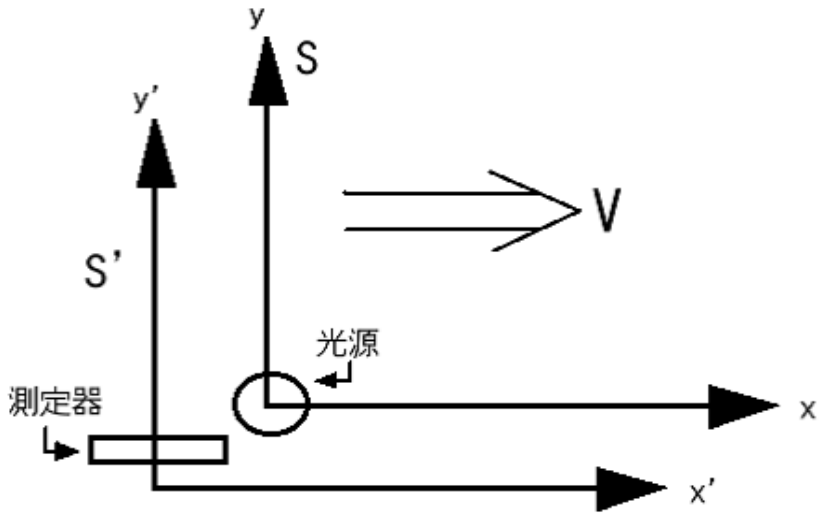


図 2.4: 光のドップラー効果

図 2.4 にように、光源が速度 V で動いているとする。光源とともに動く座標系を S とし、光源を原点とする。また、止まっている座標系を S′ とし、その原点には測定器がある。２つの原点が重なった瞬間を t=0 とする。S の時計が時刻 T_{S} のときに光のパルスを発したとする。S から S′ を見ると T'S=rT_{S} を指していた。この瞬間に S′ 系から光源を見ると、x' の位置にいたとする。光源 S から S′ に光が届くまでの時間は x'/c である。よって、s′ の原点の測定器に光が到着する時刻は

$$T_{\mathrm{S}'} = t_{\mathrm{S}'} + \frac{x'}{c} \tag{2.21}$$

となる。ここで、x' は

$$x' = V t_{\mathrm{S}'} = V T_{\mathrm{S}} \tag{2.22}$$

となる。よって、$T_{\mathrm{S}'}$ は

$$T_{\mathrm{S}'} = \gamma(1+\beta) T_{\mathrm{S}} \tag{2.23}$$

と書くことができる。
$\frac{v}{c}$ の比の値は β と示すこととする。

1つ目のパルスが発せられてから2つ目のパルスが発せられるまでの時間を、S系で測ったときはγ_{S}とし、S′系で測ったときは$\gamma_{\mathrm{S}'}$とすると、

$$\tau_{\mathrm{S}'} = \gamma(1+\beta)\tau_{\mathrm{S}} \tag{2.24}$$

となる。
振動数をνとすると、$\nu = \frac{1}{\tau}$となるので、S′系での振動数$\nu_{\mathrm{S}'}$は

$$\nu_{\mathrm{S}'} = \frac{\nu_{\mathrm{S}}}{\gamma(1+\beta)} = \nu_{\mathrm{S}}\sqrt{\frac{1-\beta}{1+\beta}} \tag{2.25}$$

となる。この式より、観測地から光源が遠ざかるときには、光の振動数が小さくなることが分かる。また、光源が近づいてくる時には速度が-vになるので

$$\nu_{\mathrm{S}'} = \nu_{\mathrm{S}}\sqrt{\frac{1+\beta}{1-\beta}} \tag{2.26}$$

となる。このような現象を光のドップラーシフトという。

2.4.3 音と光のドップラー効果の違い

これまでより、音のドップラー効果は$f\frac{1-\frac{v_{\mathrm{S}}}{V}}{1-\frac{v_{\mathrm{O}}}{V}}$、光のドップラー効果は$\nu_{\mathrm{S}}\sqrt{\frac{1-\beta}{1+\beta}}$と表されることが分かった。音のドップラー効果の式の$\frac{1-\frac{v_{\mathrm{S}}}{V}}{1-\frac{v_{\mathrm{O}}}{V}}$の部分を、$\frac{1-\beta_{\mathrm{S}}}{1-\beta_{\mathrm{O}}}$と置き換えて、2つの式を比べてみる。まず、光のドップラー効果の式$\nu_s\sqrt{\frac{1-\beta}{1+\beta}}$をテーラー展開すると、

$$\sqrt{\frac{1-\beta}{1+\beta}} = 1-\beta+\frac{1}{2}\beta^2-\frac{1}{2}\beta^3+o(\beta^4) \tag{2.27}$$

となる。同様に、音のドップラー効果の式も、測定器が止まっている場合、測定器も音源も動いている場合に分けてテーラー展開する。まず、測定器が止まっている場合の式をテーラー展開すると、

$$\frac{1}{1+\beta} = 1-\beta+\beta^2-\beta^3+o(\beta^4) \tag{2.28}$$

次に、測定器も音源も動いている場合の式をテーラー展開すると、

$$\frac{1-\frac{1}{2}}{1+\frac{1}{2}} = 1-\beta+\frac{1}{2}\beta^2-\frac{3}{4}\beta^3+o(\beta^4) \tag{2.29}$$

また、音源が止まっている場合の式は

$$1-\beta \tag{2.30}$$

である。これらの4つの式を、グラフにした。

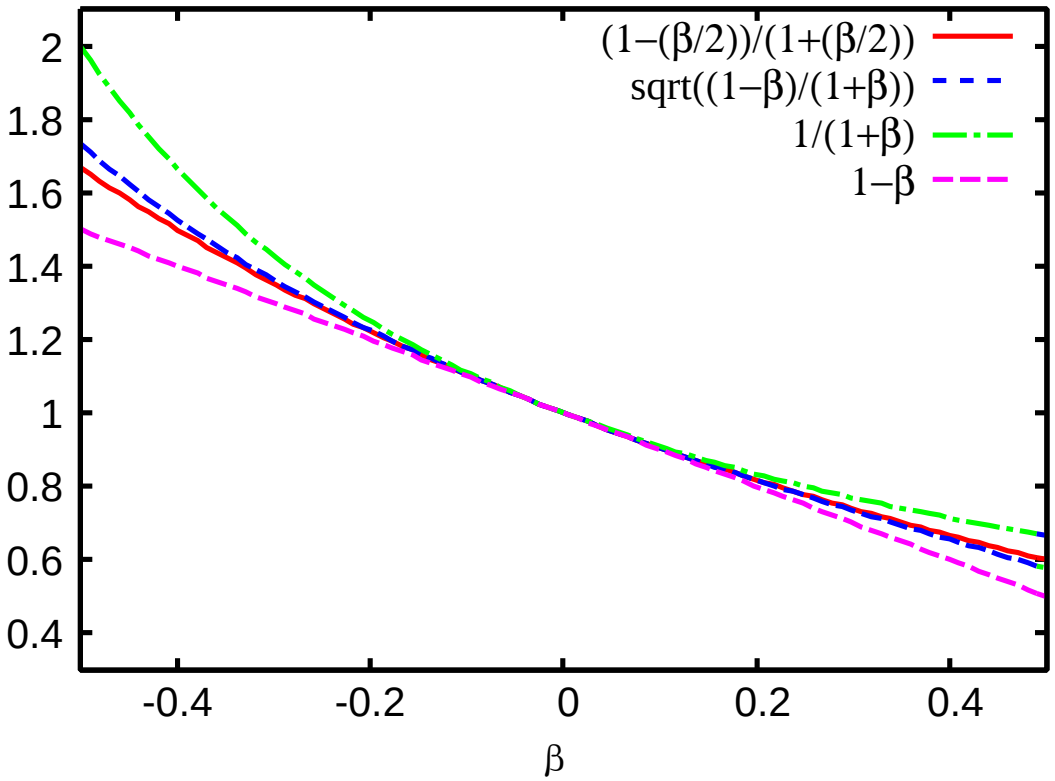


図 2.5: 音のドップラー効果と光のドップラー効果の比較

音のドップラー効果の式が光のドップラー効果の式に最も近い値を与えるのは、観測者も音源も同じ速さで反対方向に動いている場合であることが分かる。これは、式 (2.27) と、式 (2.30) の第 3 項までが一致するからである。また、その場合でも観測者と音源の相対速度が波の速さ c に比べて無視できなくなれば、両式の違いが目立ち始めることも図から見て取れる。

第3章　Dirac方程式

この章では、相対論的量子力学の方程式であるDiracの方程式について説明する。本章の執筆に際しては参考文献[1][2]を参考にした。

3.1　Dirac方程式

非相対論的な量子力学の方程式である、質量mの自由粒子に対するSchrödinger方程式は

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\nabla^2\psi \tag{3.1}$$

である。この方程式は光速度と比べゆっくりとした速さで運動する粒子に対して成り立つ量子力学の運動方程式である。

一方、相対論的なエネルギーEと運動量pの関係を書くと

$$E^2 = (pc)^2 + (mc^2)^2 \tag{3.2}$$

が成り立つ。この式で

$$E \to -\frac{\hbar}{i}\frac{\partial}{\partial t} \quad , \quad p \to \frac{\hbar}{i}\nabla$$

とおきかえ、波動関数ψに作用させると

$$-\hbar^2\frac{\partial^2}{\partial t^2}\psi = (-\hbar^2c^2\nabla^2 + m^2c^2)\psi \tag{3.3}$$

という式が得られる。この方程式はKlein-Gordon方程式と呼ばれている。

ここで、この方程式の意味をつかむために、連続の方程式を確認してみる。電荷密度ρ、電流密度$\boldsymbol{j}$は、連続の方程式

$$\frac{\partial}{\partial t}\rho + \nabla\cdot\boldsymbol{j} = 0$$

を満たす。ここでρと$\boldsymbol{j}$は次のようになる。

$$\begin{aligned}\rho &= \frac{i\hbar}{2mc^2}(\psi^*\frac{\partial}{\partial t}\psi - \phi\frac{\partial}{\partial t}\phi^*)\\ \boldsymbol{j} &= -\frac{i\hbar}{2m}(\psi^*\nabla\psi - \psi\nabla\psi^*)\end{aligned}$$

非相対論の場合には

$$\rho = \psi^*\psi = |\psi|^2$$

となり必ず正の値をとることが保証されていたが、今の場合には関数の微分がはいっていて必ずしも正の値をとることが保証されない。これは、非相対論の Schrödinger 方程式が時間についての 1 階の微分であるのに対して、Klein-Gordon 方程式が 2 階であることによるのである。

それでは、確率密度に対して連続方程式が成り立つこと、ρが正に決まっていることを条件にして、1 階の微分のみを含む方程式を作ってみよう。そのためには

$$\alpha_k\beta + \beta\alpha_k = 0, \quad \beta^2 = 1, \quad \alpha_k\alpha_j + \alpha_j\alpha_k = 2\delta_{kj} \tag{3.4}$$

を満たす行列が必要となる。最も簡単な解は、4 行 4 列の行列で

$$\alpha_k = \begin{bmatrix} 0 & \sigma_k \\ \sigma_k & 0 \end{bmatrix}, \beta = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \tag{3.5}$$

となる。ここで σ_k はパウリ行列で

$$\sigma_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \sigma_2 = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \sigma_3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.6}$$

である。I は 2 行 2 列の単位行列である。この行列を用いて

$$\{i\hbar\frac{\partial}{\partial t} + i\hbar\boldsymbol{\alpha}\cdot\nabla - \beta mc^2\}\psi = 0 \tag{3.7}$$

という方程式が導ける。この方程式が Dirac 方程式である。ここで、α や βが 4 行 4 列の行列であったことを考えると、ψ は 4 成分化され

$$\Psi(x,t) = \begin{bmatrix} \Psi_1(x,t) \\ \Psi_2(x,t) \\ \Psi_3(x,t) \\ \Psi_4(x,t) \end{bmatrix} \tag{3.8}$$

とする必要がある。

3.2 Dirac 方程式の自由粒子解

式 (3.5) の左から $-i\beta$ をかけて

$$\gamma_k = -i\beta\alpha_k = \begin{bmatrix} 0 & -i\sigma_k \\ i\sigma_k & 0 \end{bmatrix}, \quad \gamma_4 = \beta = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} \tag{3.9}$$

という行列 γ_k を定義すると、γ_k は

$$\gamma_\mu\gamma_\nu + \gamma_\nu\gamma_\mu = 2\delta_{\nu\mu}$$

を満たしている。γ_ν を用いると、Dirac 方程式は

$$\left(\gamma_1\frac{\partial}{\partial x_1} + \gamma_2\frac{\partial}{\partial x_2} + \gamma_3\frac{\partial}{\partial x_3} + \gamma_4\frac{\partial}{\partial x_4}\right)\psi = \sum_\nu \gamma_\nu\frac{\partial}{\partial x_\nu}\psi = -\frac{mc}{\hbar}\psi \tag{3.10}$$

となる。式 (3.9) を使って式 (3.10) を書き直すと

$$\begin{bmatrix} I\left(\frac{\hbar}{ic}\right)\frac{\partial}{\partial t} & -i\hbar\sigma\nabla \\ i\hbar\sigma\nabla & -I\left(\frac{\hbar}{ic}\right)\frac{\partial}{\partial t} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \psi_4 \end{bmatrix} = -mc \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \psi_4 \end{bmatrix} \tag{3.11}$$

となる。Dirac 方程式の解 $\psi(x,t)$ の各成分 $\psi_k(x,t)$ は

$$\psi_k(x,t) \;=\; u_k(p)\exp\left[i\frac{px}{\hbar} - i\frac{Et}{\hbar}\right] \tag{3.12}$$

$$E \;=\; \pm\sqrt{p^2c^2 + m^2c} \tag{3.13}$$

となる。この式を用いて式 (3.11) を変形すると

$$\begin{bmatrix} -\frac{E}{c}I & \boldsymbol{\sigma}\cdot\boldsymbol{p} \\ -\boldsymbol{\sigma}\cdot\boldsymbol{p} & \frac{E}{c}I \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = -mc \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{3.14}$$

となる。パウリ行列 σ_k の表式を使って運動量を$\boldsymbol{p}$=$p(p_1,p_2,p_3)$ とすると

$$\boldsymbol{\sigma}\cdot\boldsymbol{p} = \sigma_1 p_1 + \sigma_2 p_2 + \sigma_3 p_3 = \begin{bmatrix} p_3 & p_1 - ip_2 \\ p_1 + ip_2 & -p_3 \end{bmatrix} \tag{3.15}$$

と書ける。まず式 (3.14) を変形して

$$-\frac{E}{c}I \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \boldsymbol{\sigma}\cdot\boldsymbol{p} \begin{bmatrix} u_3 \\ u_4 \end{bmatrix} = -mc \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{3.16}$$

$$-\boldsymbol{\sigma}\cdot\boldsymbol{p} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \frac{E}{c}I \begin{bmatrix} u_3 \\ u_4 \end{bmatrix} = -mc \begin{bmatrix} u_3 \\ u_4 \end{bmatrix} \tag{3.17}$$

式 (3.16) と式 (3.17) をそれぞれ整理すると

$$\begin{bmatrix} u_3 \\ u_4 \end{bmatrix} = \frac{c\boldsymbol{\sigma}\cdot\boldsymbol{p}}{E + mc^2} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{3.18}$$

$$\begin{bmatrix} u_3 \\ u_4 \end{bmatrix} = \frac{E - mc^2}{c\boldsymbol{\sigma}\cdot\boldsymbol{p}} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{3.19}$$

を得る。式 (3.18) と式 (3.19) より

$$\left[-\frac{E}{c}I + \frac{c(\boldsymbol{\sigma}\cdot\boldsymbol{p})^2}{E + mc^2}\right] \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = -mc \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{3.20}$$

となる。ここで式 (3.15) より

$$(\boldsymbol{\sigma}\cdot\boldsymbol{p})^2 = \begin{bmatrix} p^2 & 0 \\ 0 & p^2 \end{bmatrix} = p^2 I$$

であるから、式 (3.20) は

$$\left[-E^2 + c^2p^2\right]\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = -(mc^2)^2\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

と書き直すことができる。そこで

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad , \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.21}$$

のように 2 行 1 列の行列の独立なものを選び、式 (3.14) の 2 つの独立な解として求めると

$$u^{(1)}(p) = N\begin{bmatrix} 1 \\ 0 \\ p_3c/(E+mc^2) \\ (p_1+ip_2)c/(E+mc^2) \end{bmatrix} \tag{3.22}$$

$$u^{(2)}(p) = N\begin{bmatrix} 0 \\ 1 \\ (p_1-ip_2)c/(E+mc^2) \\ -p_3c/(E+mc^2) \end{bmatrix} \tag{3.23}$$

となる。N は規格化定数である。同様に $E<0$ のときは

$$\begin{bmatrix} u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad , \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.24}$$

とおけば、独立な 2 つの解として

$$u^{(3)}((p) = N\begin{bmatrix} -p_3c/(|E|+mc^2) \\ -(p_1+ip_2)c/(|E|+mc^2) \\ 1 \\ 0 \end{bmatrix} \tag{3.25}$$

$$u^{(4)}(p) = N\begin{bmatrix} -(p_1-ip_2)c/(|E|+mc^2) \\ p_3c/(|E|+mc^2) \\ 0 \\ 1 \end{bmatrix} \tag{3.26}$$

となる。N は規格化定数である。通常

$$u^{(i)\dagger}(p)u^{(i)}(p) = \frac{E}{mc^2}$$

になるように N を決める。このとき

$$N = \sqrt{\frac{|E| + mc^2}{2mc^2}} \tag{3.27}$$

である。$E < 0$ の場合にも成り立つように、E に絶対値をつけた。

ψ の第一成分と第二成分 ψ_1 と ψ_2 は、正エネルギーの状態の主成分で、第三成分と第四成分 ψ_3 と ψ_4 は、負エネルギーの状態の主成分である。

第4章　原子核の相対論的平均場模型

4.1　原子核の構造

原子核は原子の中心にあり、陽子 (proton) と中性子 (neutron) からなる。陽子と中性子をまとめて核子と呼び、核子の間には引力が働いている。これを核力と呼ぶ。核力によって原子核は安定しており、その力は、クーロン力の約１桁強い相互作用である。陽子間には、陽子同士を遠ざけるクーロン力が働いているが、ばらばらになることがないのはこのためである。核力は、２つの核子の間で中間子を交換することにより生じる。中間子 (meson) は、核子同士を結合させている。１９３５年に湯川秀樹によって予言された。図 4.1 は、核子の間で中間子をやりとりする様子である。

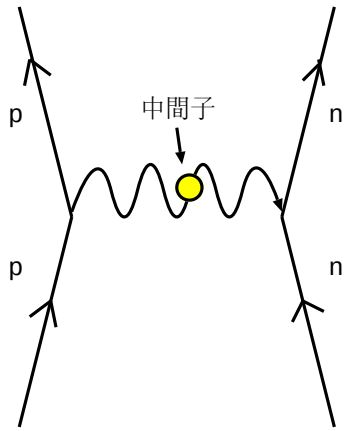


図 4.1: 核子間で交換される中間子

原子と比べると原子核は非常に小さい。原子核の半径 R は、

$$R \simeq r_o A^{\frac{1}{3}}$$

という近似で表される。r_0=1.2fm(1fm=10^{-15}m) である。例えば、^{208}Pb の場合だと、R=1.2 $\times$ $(208)^{\frac{1}{3}}$=7.1fm となる。核子の質量は、電子の質量の約 2000 倍なので、原子の質量はほとんどが原子核の質量であるといえる。

結合エネルギーとは、互いに引き合う粒子が、集まって存在する状態ととばらばらに存在する状態との間でのエネルギーの差である。核内で１核子がもつ結合エネルギーは、ほぼ 8 MeV となっている。これは、結合エネルギーは質量数にほとんどよらないことを意味しており、結合エネルギーの飽和性と呼ぶ。

4.2 相対論的平均場模型

原子核の記述において相対論を用いることには、Schrödinger 方程式とは違う大きな利点がある。まず、非相対論の場合と違いスピン・軌道結合力が自然に導かれる。又、運動エネルギーが非常に大きくなる高温・高圧下での外挿への信頼性が高い。そして、より基本的な理論（相対論的な場の理論）との関係がつく。

非相対論では普通、核子同士が直接相互作用を及ぼし合う様子を Schrödinger 方程式で表すが、相対論では Dirac 方程式で表される核子が中間子を放出したり吸収したりすることで間接的に核子同士が影響を及ぼし合う。相対論的平均場模型では、中間子を古典的な場で置き換え計算する。古典的な中間子場では、中間子は粒ではなく広がっていて、その中を核子が独立して運動している。

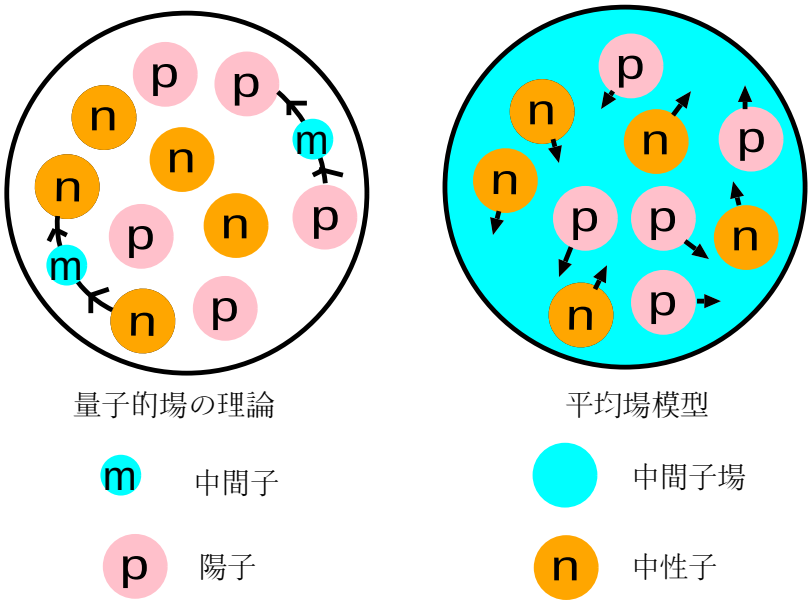


図 4.2: 中間子の様子

相対論的平均場模型においては、スカラー中間子 (J=0) とベクトル中間子 (J=1)、アイソスカラー (T=0) とアイソベクトル (T=1) 中間子を考慮することとする。

まず、核子を束縛するポテンシャルがあり、核子の波動関数は Dirac 方程式で決定される。これを Runge-Kutta 法で数値的に求め、局在した核子密度を出す。この核子密度をソースとして screened Poisson 方程式を解くと、中間子の古典場が求まる。これは、単純に数値積分で計算できる。これを結合定数場することで、また核子を束縛するポテンシャルが求まる。この Dirac 方程式と Poisson 方程式を自己無撞着になるまで繰り返す。これが、相対論的平均場模型である。

4.3 模型の定式化

この節は、文献 [4] の第 2 章と第 3 章を簡略化したものである。本論文では、核子のアイソスピン（荷電スピン）の第 3 成分 t_3 が $\frac{1}{2}$ のとき中性子、$-\frac{1}{2}$ のとき陽子であるとする。また本論文中の計算では、陽子の質量 m_p、中性子の質量 m_n は $m_p = m_n = 939(\mathrm{MeV/c^2})$ とする。

模型に取り入れる（核子間の相互作用の媒介となる）ボソン場を、表 4.1 に列挙した。中間子以外に、電磁相互作用を媒介する光子 (γ) も含まれている。S はスピン、T はアイソスピン、m は静止質量 ($\mathrm{MeV/c^2}$)、g^2 は後述の核子との結合定数の 2 乗である。

表 4.1: 中間子の種類

	S	T	$M(\mathrm{MeV/c^2})$	$\frac{g^2}{\hbar c}$ (無次元数)
σ	0	0	520	109.626
ω	1	0	783	190.431
ρ	1	1	770	16.3065
γ	1	0	0	$\frac{4\pi}{137.036}$

原子核の相対論的平均場模型で用いる有効場のラグランジアン密度は下記の通りである。

$$L = L_{\mathrm{N}} + L_\sigma + L_\omega + L_\rho + L_\gamma + L_{\mathrm{c}}.$$

核子部分は、

$$L_{\mathrm{N}} = \bar{\psi}\left(i\gamma_\mu \partial^\mu - m\right)\psi$$

で与えられる。σ 中間子部分は、

$$L_\sigma = -\frac{1}{2}\left\{(\nabla\phi_\sigma)^2 + m_\sigma^2\phi_\sigma^2\right\}$$

で与えられる。ω 中間子部分は ϕ_ω を ω 中間子場の時間成分 (第 0 成分) として

$$L_\omega = \frac{1}{2}\left\{(\nabla\phi_\omega)^2 + m_\omega^2\phi_\omega^2\right\}$$

で与えられる。ρ 中間子部分は ϕ_ρ を ρ 中間子場の時間成分で、アイソスピンの 3 軸成分が 0 の成分として

$$L_\rho = \frac{1}{2}\left\{(\nabla\phi_\rho)^2 + m_\rho^2\phi_\rho^2\right\}$$

で与えられる。光子場部分は A_0 を光子場の時間成分 (即ち、電位/e) として

$$L_\gamma = \frac{1}{2}(\nabla A_0)^2$$

で与えられる。核子場と中間子場・光子場の結合 (Coupling) は

$$L_{\mathrm{c}} = -g_\sigma \bar{\psi}\phi_\sigma\psi - g_\omega \bar{\psi}\gamma_0\phi_\omega\psi - g_\rho \bar{\psi}\gamma_0\tau_3\phi_\rho\psi - \mathrm{e}\bar{\psi}\frac{1-\tau_3}{2}\gamma_0 A_0\psi$$

で与えられる。

核子は下記の Dirac 方程式に従う。

$$\left\{-i\alpha\cdot\nabla + \beta\left(m + V_{\mathrm{S}}\right) + V_{\mathrm{V}}\right\}\psi_i = E_i\psi_i. \tag{4.1}$$

ここで、V_{S} はスカラーポテンシャルであり、下式で与えられる。

$$V_{\mathrm{S}} = -g_\sigma\phi_\sigma.$$

$V\mathrm{V}$ はベクトルポテンシャルの時間成分であり、下式で与えられる。

$$V_{\mathrm{V}} = g_\omega\phi_\omega + g_\rho\tau_3\phi_\rho + \mathrm{e}\frac{1-\tau_3}{2}A^0.$$

古典的中間子場は遮蔽された（screened） Poisson 方程式に従う。また、質量がゼロである光子の場は狭義の（proper）Poisson 方程式に従う。各々の場の源 (source term) は後述する各種の核子密度に結合定数を掛けたもので与えられる。

$$\begin{aligned}
\left(\Delta - m_\sigma^2\right)\phi_\sigma &= -g_\sigma\rho_{\mathrm{s}}, \\
\left(\Delta - m_\omega^2\right)\phi_\omega &= -g_\omega\rho_{\mathrm{V}}, \\
\left(\Delta - m_\rho^2\right)\phi_\rho &= -g_\rho\rho_3, \\
\Delta A_0 &= -\mathrm{e}\rho_{\mathrm{p}}.
\end{aligned}$$

4.3.1 球対称な系の取り扱い

球対称系では核子の波動関数は下記のような形を持つ。

$$\psi_{\varpi jmnt_3}\left(r,\theta,\varphi,m_{\mathrm{s}},m_{\mathrm{t}}\right) = \begin{pmatrix} i\frac{G_{\varpi jnt_3}(r)}{r}\Phi_{ljm}\left(\theta,\varphi,m_{\mathrm{s}}\right) \\ -\frac{F_{\varpi jnt_3}(r)}{r}\Phi_{l'jm}\left(\theta,\varphi,m_{\mathrm{s}}\right) \end{pmatrix}\delta_{t_3 m_{\mathrm{t}}},$$

但し、

$$\begin{aligned}
l &= j + \frac{1}{2}\varpi, \\
l' &= j - \frac{1}{2}\varpi, \\
\Phi_{ljm}\left(\theta,\psi,m_{\mathrm{s}}\right) &= \sum_{m_1,m_2}\left\langle l,m_1,\frac{1}{2},m_2\,|\,jm\right\rangle Y_{l,m_1}\left(\theta,\varphi\right)\delta_{m_{\mathrm{s}}m_2} \\
&= \left\langle l,m-m_{\mathrm{s}},\frac{1}{2},m_{\mathrm{s}}\,|\,jm\right\rangle Y_{l,m-m_{\mathrm{s}}}\left(\theta,\varphi\right)
\end{aligned}$$

である。ここで、$< l, m_1, \frac{1}{2}, m_2 | j, m >$はクレプシュ・ゴーダン係数、$Y_{l,m}(\theta, \varphi)$は球面調和関数である。正エネルギー解では$G$が主成分、負エネルギー解では$F$が主成分となる。

球対称系では、核子の状態は、j, ϖでラベルされる。これらのとりうる値は下記のとおりである。jは全角運動量であり、ϖはDirac方程式に特有の量子数である。

$$\begin{cases} j = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \cdots & \text{(半奇数)}. \\ \varpi = \pm 1. \end{cases}$$

この他に、エネルギー固有値を指定するには動径波動関数の主要成分Gのノード数n(0以上の整数値をとる）などが必要である。$\varpi = -1$のとき、小さい成分Fのノード数はnに等しい。$\varpi = 1$のときは、Fのノード数は$n+1$である。

さらに、方程式を簡明に表すための副次的な量として下記のものを使用する。

$$\kappa = \varpi \left(j + \frac{1}{2} \right) = \pm 1, \pm 2, \pm 3, \cdots,$$

$$\left.\begin{array}{ll} l = j + \frac{1}{2}\varpi & (G \text{の軌道角運動量}) \\ l' = j - \frac{1}{2}\varpi & (F \text{の軌道角運動量}) \end{array}\right\} = 0, 1, 2, \cdots.$$

核子の動径波動関数は下記の方程式に従う。

$$\frac{d}{dr} \begin{pmatrix} G \\ F \end{pmatrix} = \begin{pmatrix} -\frac{\kappa}{r}, & \mu + \varepsilon \\ \mu - \varepsilon, & \frac{\kappa}{r} \end{pmatrix} \begin{pmatrix} G \\ F \end{pmatrix} \tag{4.2}$$

但し、

$$\mu = m + V_{\mathrm{s}}, \quad \varepsilon = E - V_{\mathrm{V}}$$

とした。

iを軌道の番号、n_iをi番目の軌道に入っている核子の個数$(0 \leq n_i \leq 2j_{i+1})$として、下記のような各種の核子密度を定義する。

$$\begin{aligned} \rho_{\mathrm{s}}(r) &= \frac{1}{4\pi r^2} \sum_i n_i \left\{ |G_i(r)|^2 - |F_i(r)|^2 \right\} \\ \rho_{\mathrm{V}}(r) &= \frac{1}{4\pi r^2} \sum_i n_i \left\{ |G_i(r)|^2 + |F_i(r)|^2 \right\} \\ \rho_3(r) &= \frac{1}{4\pi r^2} \sum_i n_i (\tau_3)_i \left\{ |G_i(r)|^2 + |F_i(r)|^2 \right\} \\ \rho_{\mathrm{p}}(r) &= \frac{1}{4\pi r^2} \sum_i n_i \left(\frac{1 - \tau_3}{2} \right)_i \left\{ |G_i(r)|^2 - |F_i(r)|^2 \right\} \end{aligned}$$

ρ_{s}はスカラー密度、ρ_{V}はバリオン密度(核子流の時間成分、即ち、通常の密度)、ρ_3はアイソベクトル密度のアイソ空間の第3軸成分、ρ_{p}は陽子密度である。

4.3.2 核子の波動関数の求め方

境界条件を与えて Dirac 方程式の動径座標部分を 4 次の Runge-Kutta 法で解けば、動径波動関数を求めることができる。エネルギー E の値を仮定して、$r \to +0$ での正則解の漸近形から出発して r の増加する方向へ解き進めた解と、$r \to +\infty$ での正則解の漸近形から出発して r の減少する方向へ解き進めた解とを適当な点で接続し、波動関数をリスケールすることで G、F の 2 成分がともに連続に接続できるとき、仮定した E はエネルギー固有値となっているのである。

$r = 0$ でのテーラー展開は下記のとおりである。

$$G = r^{l+1} - \frac{\varepsilon^2 - \mu^2}{2(2l+3)} r^{l+3} + \mathcal{O}\left(r^{l+5}\right) \tag{4.3}$$

$$F = \frac{l+1+\kappa}{\varepsilon+\mu} r^l - \frac{l+3+\kappa}{2(2l+3)} (\varepsilon - \mu) r^{l+2} + \mathcal{O}\left(r^{l+4}\right) \tag{4.4}$$

但し、

$$\begin{cases} \varepsilon = E - V_{\mathrm{V}}(r=0) \\ \mu = m + V_{\mathrm{s}}(r=0) \end{cases}$$

とした。

$r \to +\infty$ での漸近形は下記の通りである。

$$G = \mathrm{e}^{-\sqrt{m^2-E^2}r}, \tag{4.5}$$

$$F = -\sqrt{\frac{m-E}{m+E}} \mathrm{e}^{-\sqrt{m^2-E^2}r}. \tag{4.6}$$

$G_1(r), F_1(r)$ を $r \to +0$ で正しい漸近形を持つ解、$G_2(r), F_2(r)$ を $r \to +\infty$ で正しい漸近形を持つ解とする。

2 つの解が一致する条件は、任意に取り得る matching point $r = r_{\mathrm{m}}$ にて G，F 共に連続に接続できることである。G，F は連立 1 階常微分方程式の解なので，関数値が一致すれば (連続に接続できれば) 任意の階数の微分も一致するからである。従って接続条件は

$$\frac{G_1(r_{\mathrm{m}})}{F_1(r_{\mathrm{m}})} = \frac{G_2(r_{\mathrm{m}})}{F_2(r_{\mathrm{m}})}$$

である。

$$D = G_1(r_{\mathrm{m}}) F_2(r_{\mathrm{m}}) - F_1(r_{\mathrm{m}}) G_2(r_{\mathrm{m}})$$

とすれば

$$D = 0$$

が接続条件である。

4.4 自己無撞着解を得るためのアルゴリズム

自己無撞着解を得るための手順は、まず、$V_{\rm s}$、$V_{\rm V}$ に初期値を仮定した上で以下の１〜４の手順を繰り返すことである。

1. $V_{\rm s}$、$V_{\rm V}$ に対する核子のエネルギー固有値 E_i と波動関数を求める。
2. E_i の低い状態から占拠核子数 n_i を割り当て、各種の核子密度を得る。
3. 核子密度を源とする中間子場の古典解を求め、得られた中間子場を組み合わせて $V_{\rm s}$、$V_{\rm V}$ を作る。
4. ３.で得られた $V_{\rm s}$、$V_{\rm V}$ が１.で使用した $V_{\rm s}$、$V_{\rm V}$ と異なるなら、１.に戻る。逆に差が非常に小さければ、自己無撞着解が得られたことになる。

4.5 原子核の全エネルギーの表式

自己無撞着解が得られたら、原子核の全エネルギー E は以下の表式で計算することができる

$$\begin{aligned} E &= \sum_i n_i E_i - \frac{1}{2}\int_0^\infty \{-g_\sigma\phi_\sigma(r)\,\rho_{\rm s}(r) + g_\omega\phi_\omega(r)\,\rho_{\rm v}(r) + g_\rho\phi_\rho(r)\,\rho_3(r) \\ &\quad + {\rm e}A_0(r)\,\rho_{\rm p}(r)\}\,4\pi r^2 dr. \end{aligned} \tag{4.7}$$

4.6 $V_{\rm s}, V_{\rm v}$ の初期値の設定法

空間的に一様な系では $\Delta\phi = 0$ なので

$$\left(\Delta - m^2\right)\phi = -g\rho$$

より、

$$\phi = \frac{g}{m^2}\rho$$

を得る。

$$\rho_{\rm n} = 0.16\frac{N}{A}\left({\rm fm}^{-3}\right), \quad \rho_{\rm p} = 0.16\frac{Z}{A}\left({\rm fm}^{-3}\right)$$

と仮定すれば

$$\begin{aligned} \rho_{\rm V} &= \rho_{\rm n} + \rho_{\rm p} = 0.16, \\ \rho_{\rm s} &\fallingdotseq \rho_{\rm V} \times 0.9, \\ \rho_3 &= \rho_{\rm n} - \rho_{\rm p} = 0.16\frac{N-Z}{A} \end{aligned}$$

が導かれ、ポテンシャルの深さは

$$\begin{aligned} {V_{\mathrm{V}}}^{(0)} &= \frac{g_\omega^2}{m_\omega^2}\rho_{\mathrm{V}} + \tau_3\frac{g_\rho^2}{m_\rho^2}\rho_3, \\ V_{\mathrm{s}}^{(0)} &= -\frac{g_\sigma^2}{m_\sigma^2}\rho_{\mathrm{s}}, \end{aligned}$$

となる。そこで

$$\begin{aligned} V_{\mathrm{V}}(r) &= \frac{{V_{\mathrm{V}}}^{(0)}}{1+\exp\left(-\frac{r-R}{a}\right)} + V_{\mathrm{coul}}(r) \\ V_{\mathrm{s}}(r) &= \frac{V_{\mathrm{s}}^{(0)}}{1+\exp\left(-\frac{r-R}{a}\right)} \end{aligned}$$

とすればよい。

但し、

$$\begin{aligned} a &= 0.7\ \mathrm{fm}, \\ R &= r_0 A^{\frac{1}{3}}, \\ r_0 &= 1.2\ \mathrm{fm}, \end{aligned}$$

とし、クーロンポテンシャルは陽子に対してのみ働き、

$$V_{\mathrm{coul}}(r) = \begin{cases} \frac{Z\mathrm{e}^2}{4\pi R}\left\{\frac{3}{2} - \frac{1}{2}\left(\frac{r}{R}\right)^2\right\} & (r \le R), \\ \frac{Z\mathrm{e}^2}{4\pi r} & (r > R), \end{cases}$$

と近似するとよい。

第5章　Dirac方程式の数値解法とその精度

5.1　グリッド間隔と原子核の全エネルギーの誤差の関係

グリッド間隔と全エネルギーの誤差の関係を説明するために、まず核子の波動関数について考える。核子の波動関数は (4.2) 式の解である。その解を動径座標 r 軸上に刻んだ等間隔 Δr のグリッド上で決定する。

i 番目のグリッド転を $r_i = \Delta r \cdot i$ とする。i_1 番目までのグリッド点には、動径 r=0 でのテーラー展開解 (4.3) 式と (4.4) 式を使い、その先を Runge-Kutta 法で求める。$r \geq 25fm$ は漸近解 (4.5) 式と (4.6) 式に接続させる。

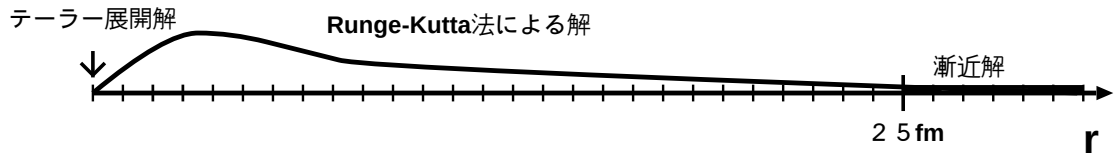


図 5.1: 核子の波動関数

原子核の全エネルギー E は、(4.7) 式で求められる。

5.1.1 結果

図 5.2 は、横軸を Δr でとったときの、核の全エネルギーの誤差 $|\Delta E|(MeV)$ のグラフである。核は、^{208}(Pb)(N = 126) である。

Δr はグリッド間隔 (fm) で、$\Delta r = 0.002$ (fm) での計算値を正確な値と見なした。テーラー展開への接続点は $r = 0.12$(fm) に固定した。傾きを計算したところ 4.7 であった。Runge-Kutta 法の誤差は $(\Delta r)^4$ に比例し、中間子場を求める積分の誤差は $(\Delta r)^5$ に比例しているが、4.7 という値はその間にある値である。このことは、プログラムの正しさの検証になっているといえる。

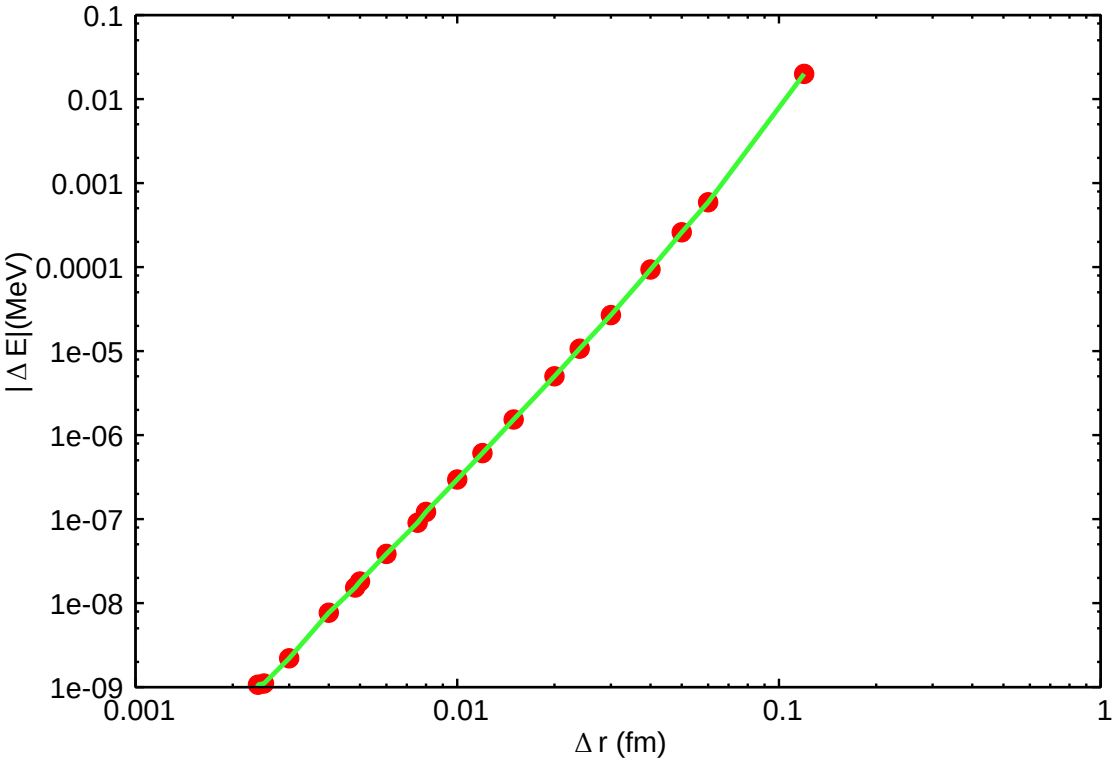


図 5.2: 全エネルギーの誤差

図 5.3 には、等間隔グリッドを用いた場合の ^{208}Pb 核の全エネルギーの誤差と Runge-Kutta 法で扱う動径グリッド点の個数の関係に対する Taylor 展開解に接続する位置の影響の結果を示した。

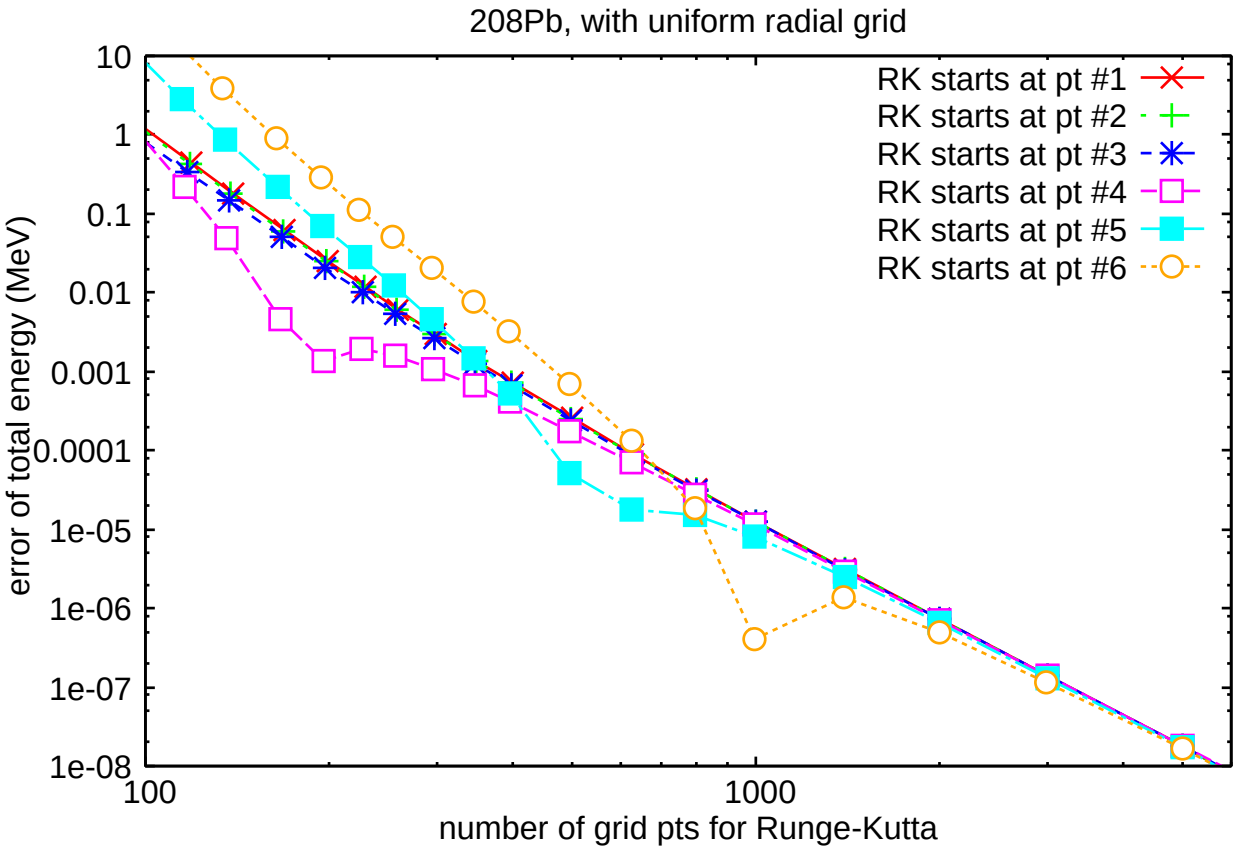


図 5.3: Taylor 展開解に接続する位置の影響

等間隔グリッドを用いる場合、座標原点 ($r = 0$) を 0 点目として、原点から i 点目で Taylor 展開による解を Runge-Kutta 法に接続するとして、i の取り方により ^{208}Pb の全エネルギーの誤差がどう変化するかを調べた結果が図 5.3 である。横軸は Runge-Kutta 法で扱う動径グリッド点の個数である。

この図より、i=1,2,3 の誤差はほとんど等しいが、$i \geq 4$ では誤差の符号が大きく変動することが分かる。これは、グリッド点の個数の増加とともに、誤差の符号が変わるようになったためと考えられる。このような不安定な振る舞いは結果の信頼性を損なうので、$i \leq 3$ を選択すべきだと思われる。

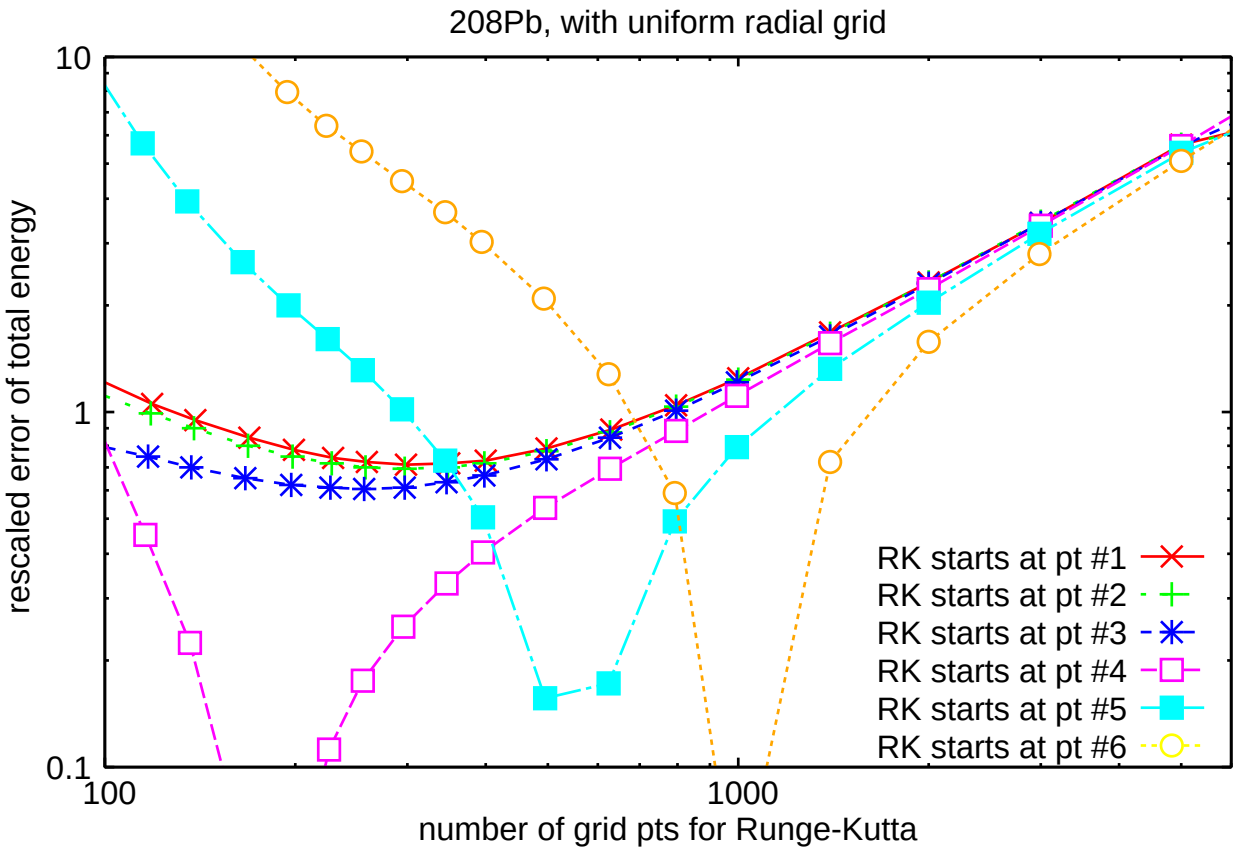


図 5.4: 図 5.3 の改良

図 5.4 は、図 5.3 の縦軸にプロットした値に、(横軸の値/100)^5 を乗じることで折れ線同士の比較を容易にしたものである。これを見ると、$i \leq 3$ の中では、$i = 3$ が最も誤差が小さいことがわかる。

これは以下の二つの要因の競合による結果と考えられる。

- Taylor 展開解は r の値が小さいほど精度が良いので、小さい r に対応する小さい i で Taylor 展開解に接続するほうが精度が高くなる。
- 動径波動関数の従う常微分方程式 (Dirac 方程式の動径部分) は $1/r$ に比例する項を含むため、r が小さいほど Runge-Kutta 法で小さな増分を使う必要があり、逆に言えば、増分が一定ならば、あまり小さい i で Taylor 展開解に接続すること精度低下につながる。

しかし、$i = 1$ と $i = 3$ の違いが小さいことを考慮して、これ以降に示す等間隔グリッドによる計算例では $i = 1$ を用いた。

5.2 r軸上のどこで大きな誤差が見られるか

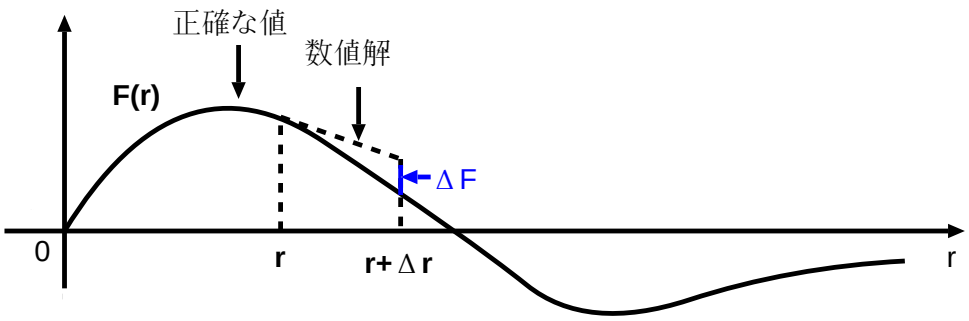


図 5.5: 波動関数の誤差を調べる

Runge-Kutta 法で Δr=0.05fm 進む際に生じる動径波動関数の誤差の大きさを r の関数としてプロットする。波道関数の誤差は $\sqrt{(\Delta F)^2 + (\Delta G)^2}$ とし、$\Delta r = 0.005$fm で得た解を正確とみなす。

5.2.1 束縛エネルギーの大きい軌道と小さい軌道の比較

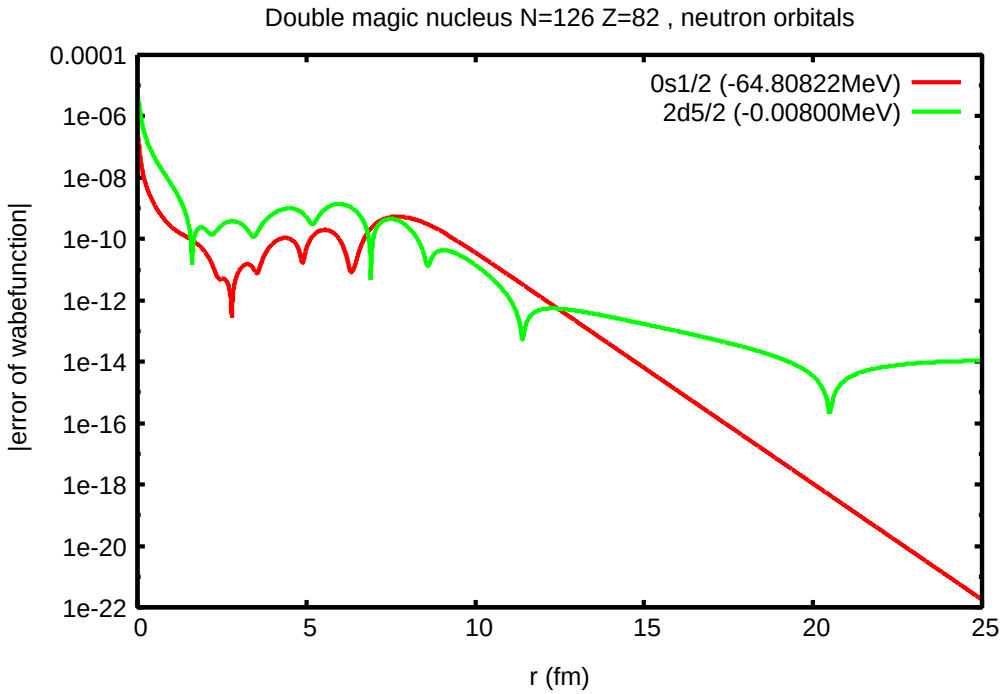


図 5.6: 束縛エネルギーの大きい軌道と小さい軌道

束縛エネルギーによらず、r の小さいところでは誤差が大きくなっていることが分かる。また、束縛エネルギーが小さい軌道では r が大きくなっても誤差は小さく、束縛エネルギーが大きい軌道では r が大きくなっても誤差が大きいことが分かる。

5.2.2 安定核 (N=126 Z=82) と、不安定核 (N=170 Z=82) の比較

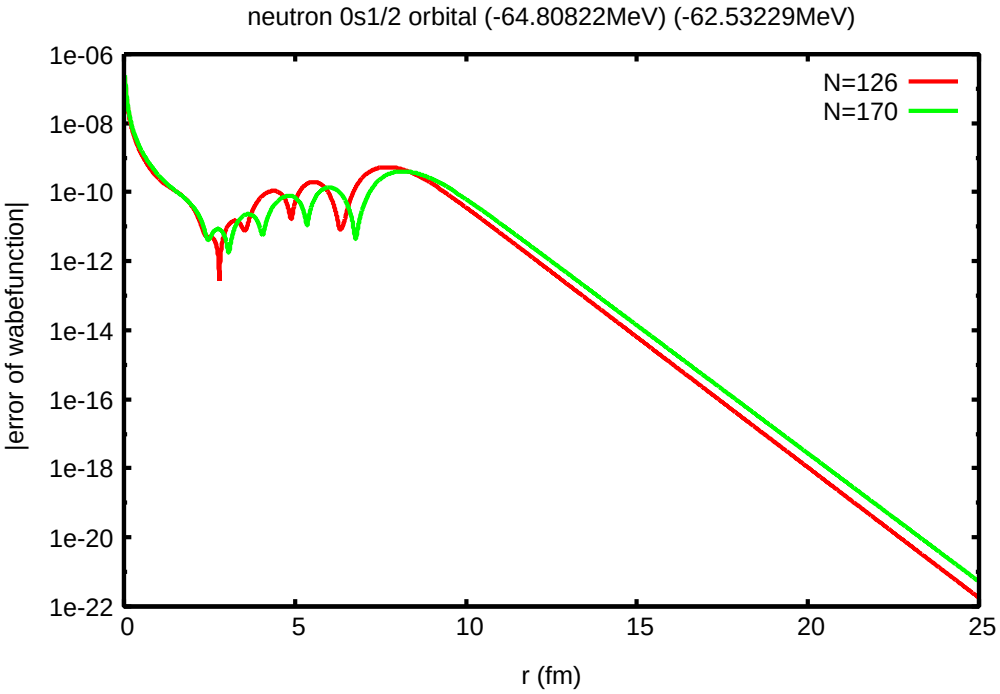


図 5.7: 束縛エネルギーの大きい軌道と小さい軌道

図 5.6 と同じ ように、r が小さいところで誤差が大きいという傾向は変わらない。また、誤差の絶対値は大きい r で不安定核のほうがやや大きいが、傾向は同じである。

5.2.3 角運動量の大きい軌道と小さい軌道の比較 (束縛エネルギーは同程度のもの)

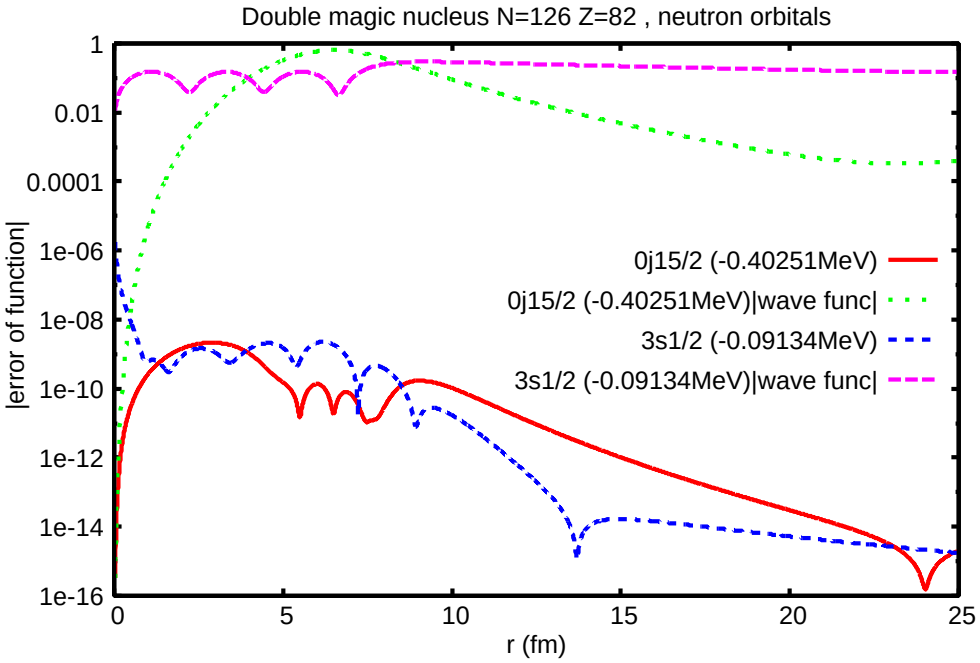


図 5.8: 束縛エネルギーの大きい軌道と小さい軌道

角運動量の大きい軌道では、rが小さいところで誤差は小さくなるが、これは遠心力ポテンシャルのせいで波動関数が小さいためである。相対誤差としてみれば、やはり小さなrで大きな誤差が発生するといえる。

5.3　非等間隔グリッドの導入

これまでの結果より、r の小さいところでは誤差が大きいことが分かった。そこで、r が小さいところでグリッド間隔 Δr を小さくとれば、全誤差を効率的に減少させることができると予想し、非等間隔グリッドを導入した。

$$\frac{r}{R_s} = f\left(\frac{\xi}{R_s}\right) \tag{5.1}$$

r は実際の半径で、r を変数 ξ に変換する。

$$\int_0^x \frac{\cosh t}{\sqrt{1+(\frac{\cosh t}{c})^2}} dt = c \,\mathrm{arcsinh}\frac{\sinh x}{\sqrt{c^2+1}} \tag{5.2}$$

変換した ξ で、微分方程式、数値積分を解いていく。

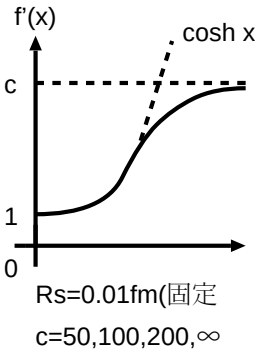


図 5.9: 非等間隔グリッド

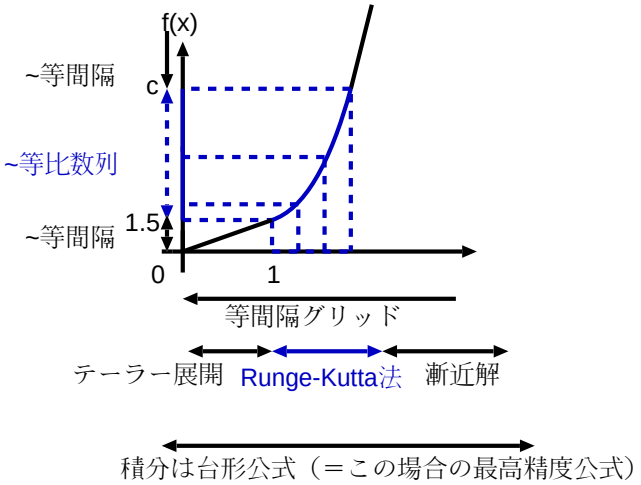


図 5.10: 非等間隔グリッド

解いた変数は図 5.9 のようになる。r の非常に小さいところと大きいところでは等間隔になっており、パラメータは c 倍だけ違う。ξ は等間隔グリッドで使うとすれば、対

応する r のグリッドは、原点付近の密な等間隔領域と粗な等間隔領域を等比数列で繋いだようなグリッドとなる。

5.3.1 結果

非等間隔グリッドを用いた場合の ^{208}Pb 核の全エネルギーの誤差と Runge-Kutta 法で扱う動径グリッド点の個数の関係に対するグリッド パラメータの積 $R_{\rm s}c$ の影響を調べた。（$R_{\rm s}c \geq 0.023$ fm の場合）

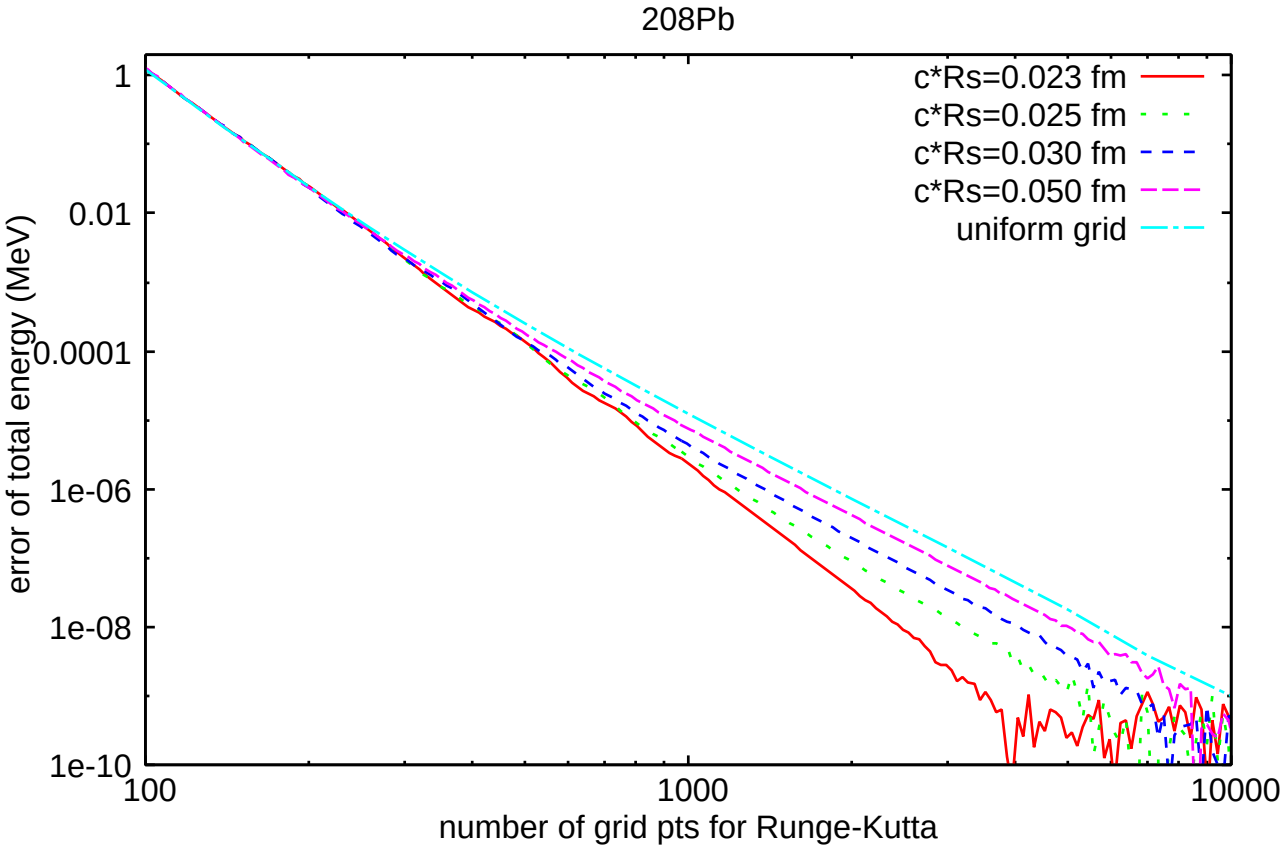


図 5.11: グリッド パラメータの積 $R_{\rm s}c$ の影響

非等間隔グリッドのパラメータ $R_{\rm s}$, c を様々に変化させた場合に、^{208}Pb の全エネルギーの誤差がどのように振る舞うかを調べたところ、積 $R_{\rm s}c$ が同じ値となる $(R_{\rm s}, c)$ の組み合わせは全て、「誤差」 対「グリッド点の総数」のグラフがほとんど重なることが判明した。

そこで、$c = 50$ に固定し、$R_{\rm s}$ の値だけを変化させて誤差の振る舞いを調べたところ、$R_{\rm s}c = 0.023$ fm のときに誤差は最小となることがわかった。（ただし計算例とした ^{208}Pb 以外の場合では値が異なる可能性は残されている。）

図 5.11 には、$R_{\rm s}c \geq 0.023$ fm の場合を示した。有限桁に起因する誤差限界 10^{-9} MeV に到達するのに要するグリッド点の個数は、等間隔グリッドで計算する場合が約 10000 点であるのに対し、$R_{\rm s}c \geq 0.023$ fm の場合は約 4000 点で十分であることがわかる。

図 5.12 は、$R_{\rm s}c \geq 0.023$ fm の場合について図 5.5 に対応させたものである。

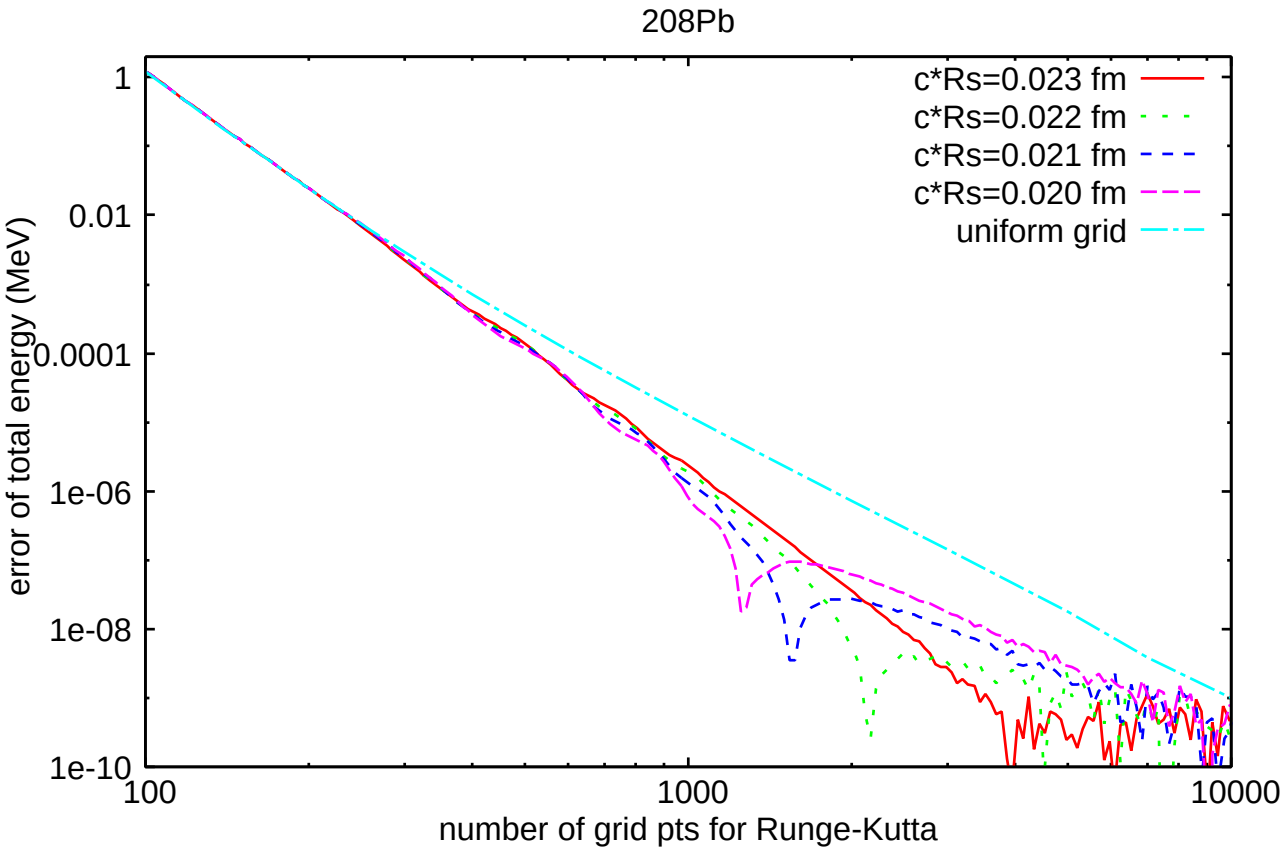


図 5.12: $R_{\rm s}c \geq 0.023$ fm の場合を図 5.5 に対応させた

図 5.12 には $R_{\rm s}c \leq 0.023$ fm の場合を示した。$R_{\rm s}c < 0.023$ fm では、誤差がグリッド点の関数としてある点で符号を変えるようになるため、そのノード付近では誤差が小さいが、限界誤差に達するのは、$R_{\rm s}c = 0.023$ fm の場合より多くのグリッド点を要することがわかる。

ただし、非等間隔グリッドのパラメータの最適値の最終的な決定には、他の核でも同様に誤差を調べる必要があり、これは、今後のこの研究を引き継ぐ者に課せられた課題である。

第6章　振動現象の解決策

この章では、自己無撞着解への収束の妨げとなる振動現象の解決策について説明する。

6.1　自己無撞着解を得るためのアルゴリズム

自己無撞着解とは、英語で self-consistent といい、平均場近似を行うときの重要な概念である。今回の相対論的平均場模型プログラムに使われている自己無撞着解を得るためのアルゴ リズムは, 以下の１～５の手順を自己無撞着するまで繰り返すことである。

1. 初期値を仮定したポテンシャルを基に核子のエネルギーと波動関数を求める。
2. 求めたエネルギーを基に各軌道に占拠核子数を割り当て配位を決定する。
3. １の波動関数と２の占拠核子数より核子密度を得る。
4. 核子密度を源とする Poisson 方程式を解いて中間子場の古典解を求める。
5. 中間子場を組み合わせてポテンシャルを作る。

理想的には、１のポテンシャルと５のポテンシャルが一致することが望ましいが、今回は差が非常に小さくなったときに、自己無撞着解が得られたとする。

6.2 占拠個数の振動現象

自己無撞着解を求める過程で、配位を決定したときに図 6.1 のように計算をさせるたびに 2 軌道が入れ替わることがある。この 2 軌道が入れ替わる現象が求めるポテンシャルに影響を与え自己無撞着させることが出来ず、解を得ることができない。これは、フェルミ面を挟む 2 軌道のエネルギー準位が非常に近い場合におこる。

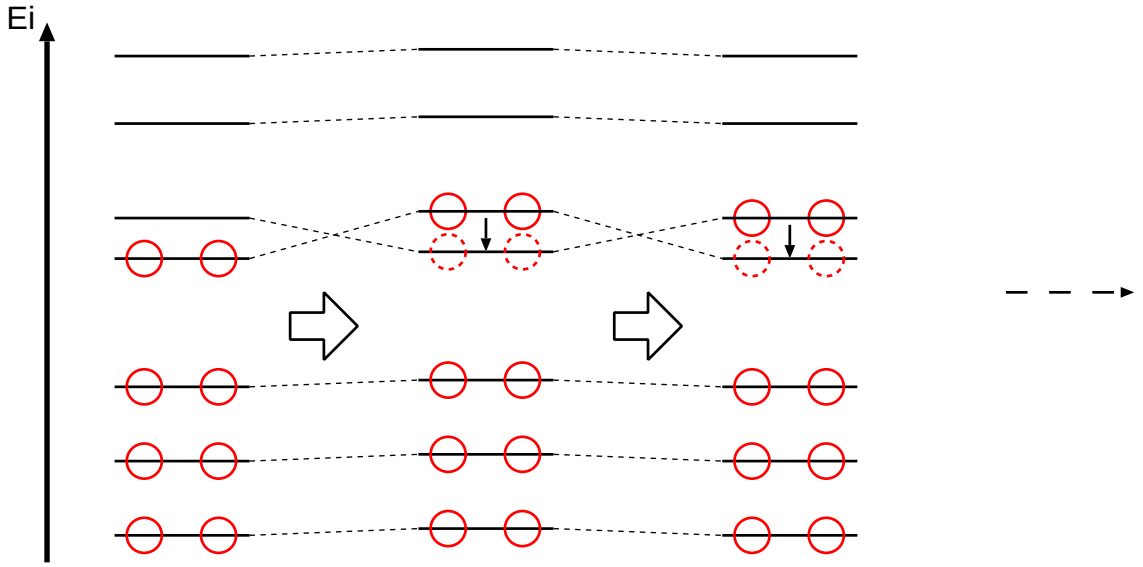


図 6.1: 振動現象の模式図

6.3 振動の具体例

振動現象の具体例を示す。中性子 N=176 陽子 Z=82 の鉛の場合に、フェルミ面を挟む 2 軌道が入れ替わっている。他の原子核でも振動現象がおこる場合は多々あるが、本研究ではこの鉛を例にとり議論を進めるとする。図 6.2、図 6.3 にエネルギーの振動現象を示した。iteration number は計算の反復回数である。図 6.3 の 1 核子のエネルギーの振動は 10KeV から 35KeV に対して、図 6.2 の全エネルギーの振動は 12MeV ほどあり、約 1000 倍にもなっていることが分かる。これは、フェルミ面を挟む 2 軌道が入れ替わることで他の軌道も変化しているためである。

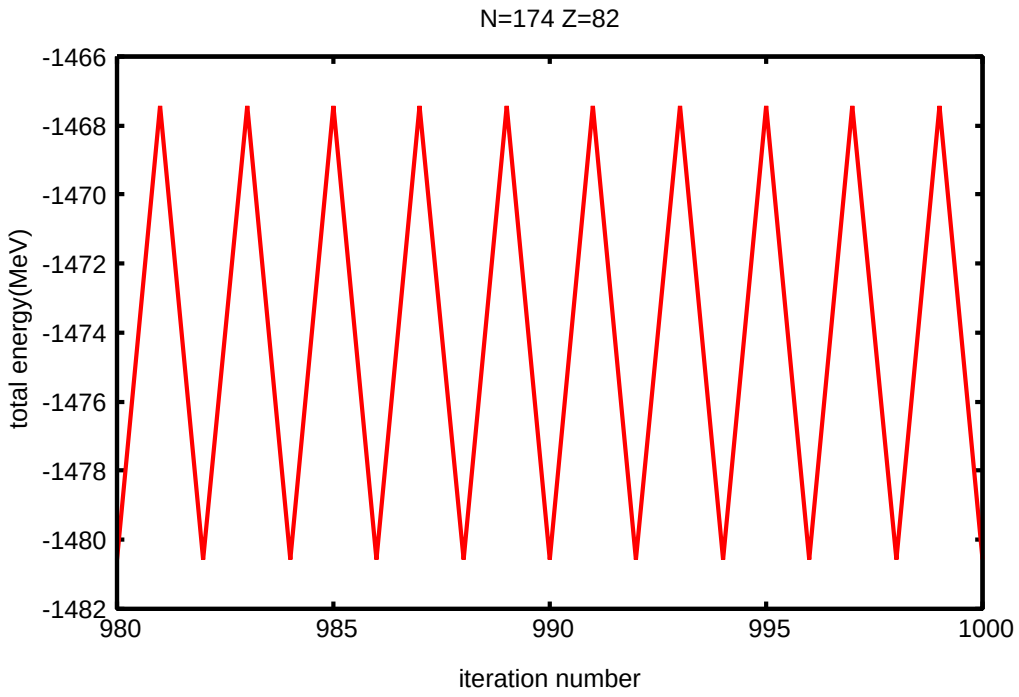


図 6.2: 全エネルギーの振動現象

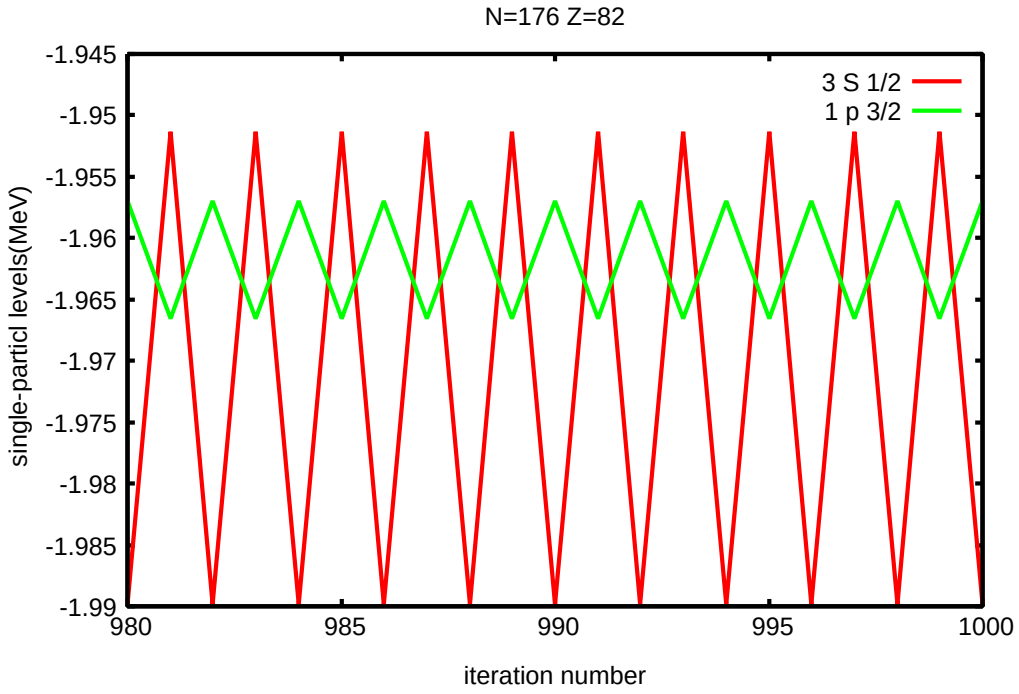


図 6.3: 1 核子準位の振動現象

6.4 減衰因子の導入

6.4.1 方法

6.1 節で説明した自己無撞着解を求めるアルゴリズムで、次の計算のために用意するポテンシャルを計算させる時に減衰因子を導入する。まず

$$\begin{aligned} V_{\mathrm{old}} &: \text{計算で使うポテンシャル} \\ V_{\mathrm{new}} &: \text{計算で求めたポテンシャル} \\ V_{\mathrm{next}} &: \text{次の計算で使うポテンシャル} \\ p &: \text{減衰因子} \end{aligned}$$

とおく。今までは単純に

$$V_{\mathrm{next}} = V_{\mathrm{new}}$$

としていたのに対して、減衰因子 p を導入して

$$V_{\mathrm{next}} = (1-p)V_{\mathrm{old}} + pV_{\mathrm{new}} \quad (0 \leq p \leq 1)$$

とおきかえてみる。これは、数値計算の上等手段として広く知られている。このように次の計算で使うポテンシャルを求めることで、徐々に新しいポテンシャルにすることができて、振動現象を抑えることができると期待される。

6.4.2 結果

結果をグラフで表した。図 6.4、図 6.5 共に横軸は計算の反復回数である。結果を見易くするため、振動の一部分を抜粋している。共に縦軸は 1 軌道のエネルギー準位である。図 6.4 は減衰因子 $p = 0.1$ の場合の結果で、図 6.5 は減衰因子 $p = 0.03$ の場合で p が十分小さい場合の結果である。この結果から、減衰因子 p を小さくしていくことで、軌道の入れ替わりがおこる回数と振動現象の振幅を抑えることができることが確認できるが、今回の目的の、振動現象を止め自己無撞着解を求めるには至らなかった。

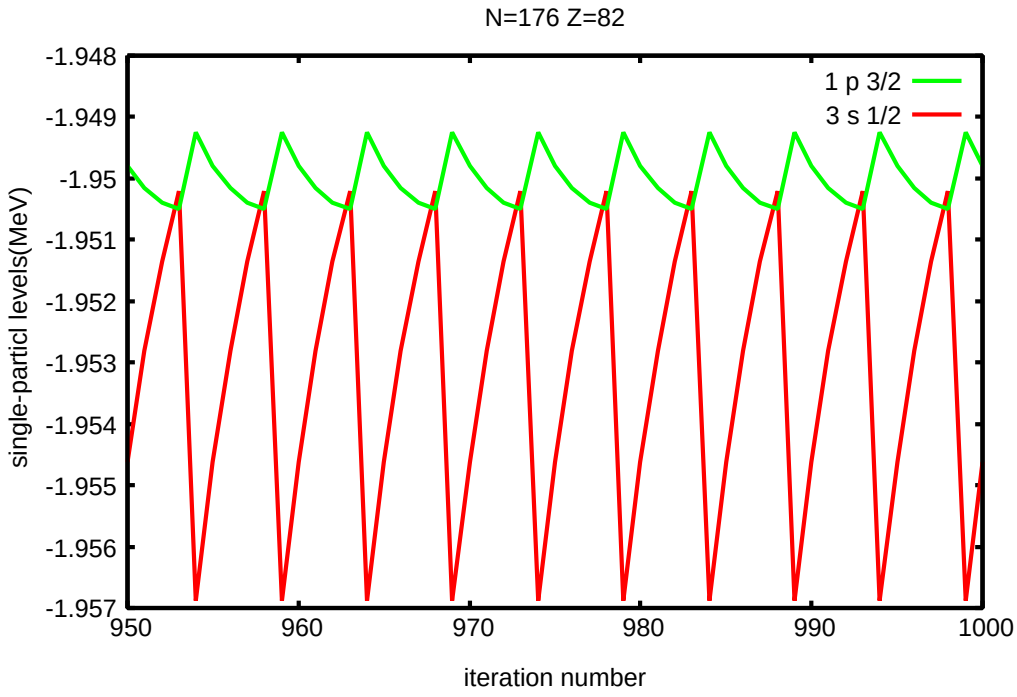


図 6.4: $p = 0.1$ の時の 1 核子準位の振動現象

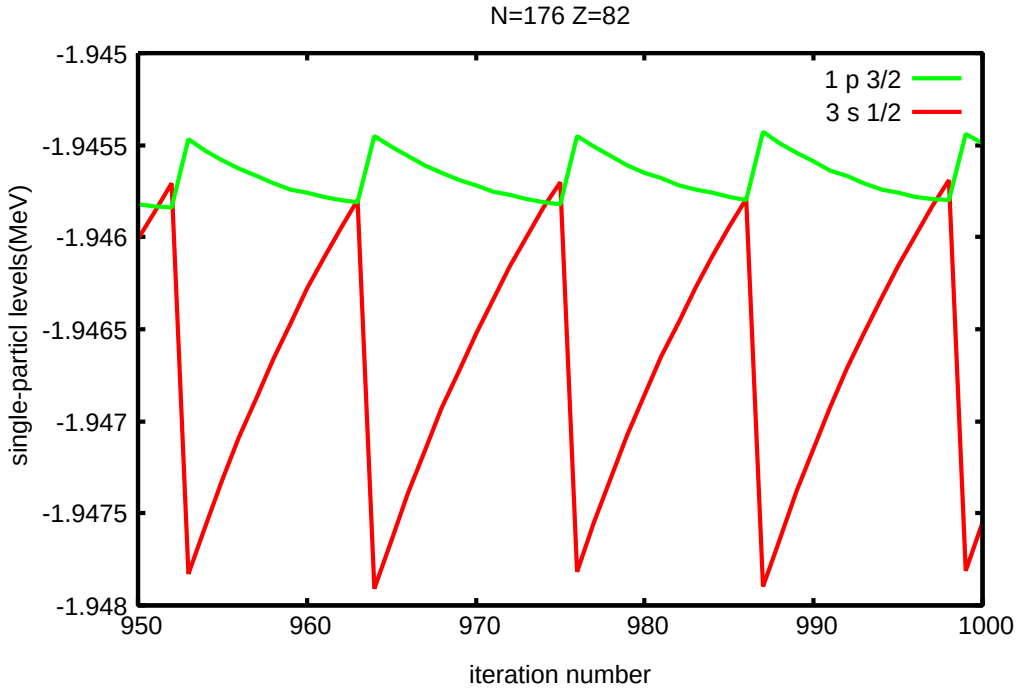


図 6.5: $p = 0.03$ の時の 1 核子準位の振動現象

6.5 有限温度化

6.5.1 理論

ここではフェルミ分布関数を導入する。

$$f(\epsilon) = \frac{1}{1 + \exp\left(\beta(\epsilon - \epsilon_F)\right)} \quad , \quad \beta = \frac{1}{k_B T}$$

但し、

$$\begin{aligned} k_B &: \text{ボルツマン定数} \\ T &: \text{温度} \\ \epsilon &: \text{一粒子準位} \\ \epsilon_F &: \text{フェルミ準位 (化学ポテンシャル)} \end{aligned}$$

とする。軌道 i の縮退度を g_i、エネルギー準位を ϵ_i、占拠粒子個数を n_i とおき、

$$n_i = g_i f(\epsilon_i)$$

となるようにとると、温度 T のときの分布になる。ここでフェルミ準位 ϵ_F は粒子の個数を N として、

$$N = \sum_i n_i$$

を満たすように決定する。$\frac{1}{\beta}$ は、反復で逆転のおきる 2 準位の間隔程度以上の大きさにとる必要がある。

このように決定したフェルミ分布関数を作用させることで、図 6.6 のようにフェルミ面を滑らかにすることができ振動現象を抑制できると期待される。

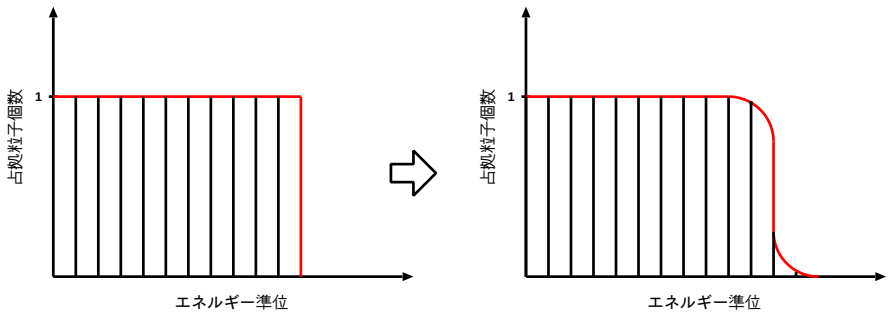


図 6.6: フェルミ面

6.5.2 結果

$\frac{1}{\beta} = 0.1(\text{MeV})$ として、計算させた結果を図 6.7 と図 6.8 に示した。これを見ると、全エネルギー、核子のエネルギーともに振動を抑制できることが分かる。次の方法で求める正しい全エネルギーとの差は約 70KeV ほどあり、これは有限温度で行ったので温度がゼロの場合の解ではないためである。

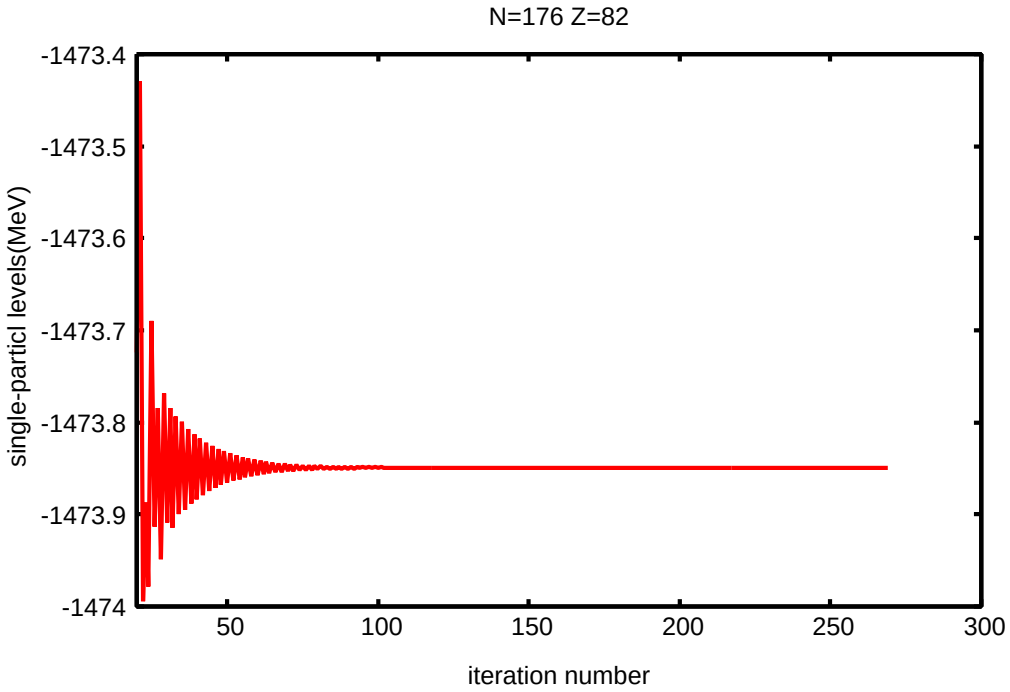


図 6.7: $\frac{1}{\beta} = 0.1$ の時の全エネルギー

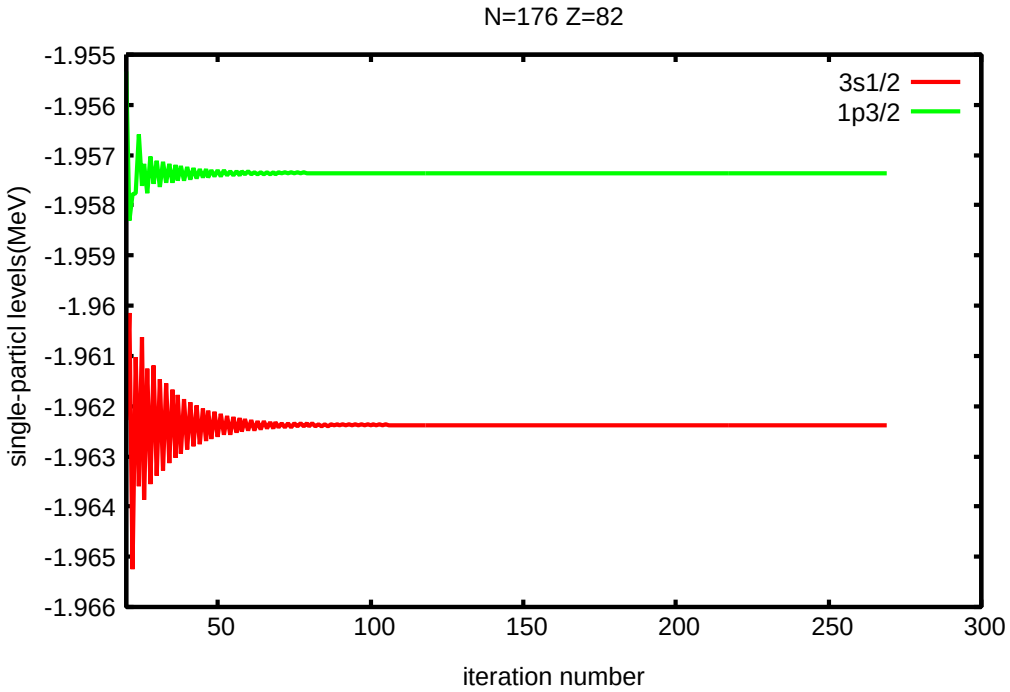
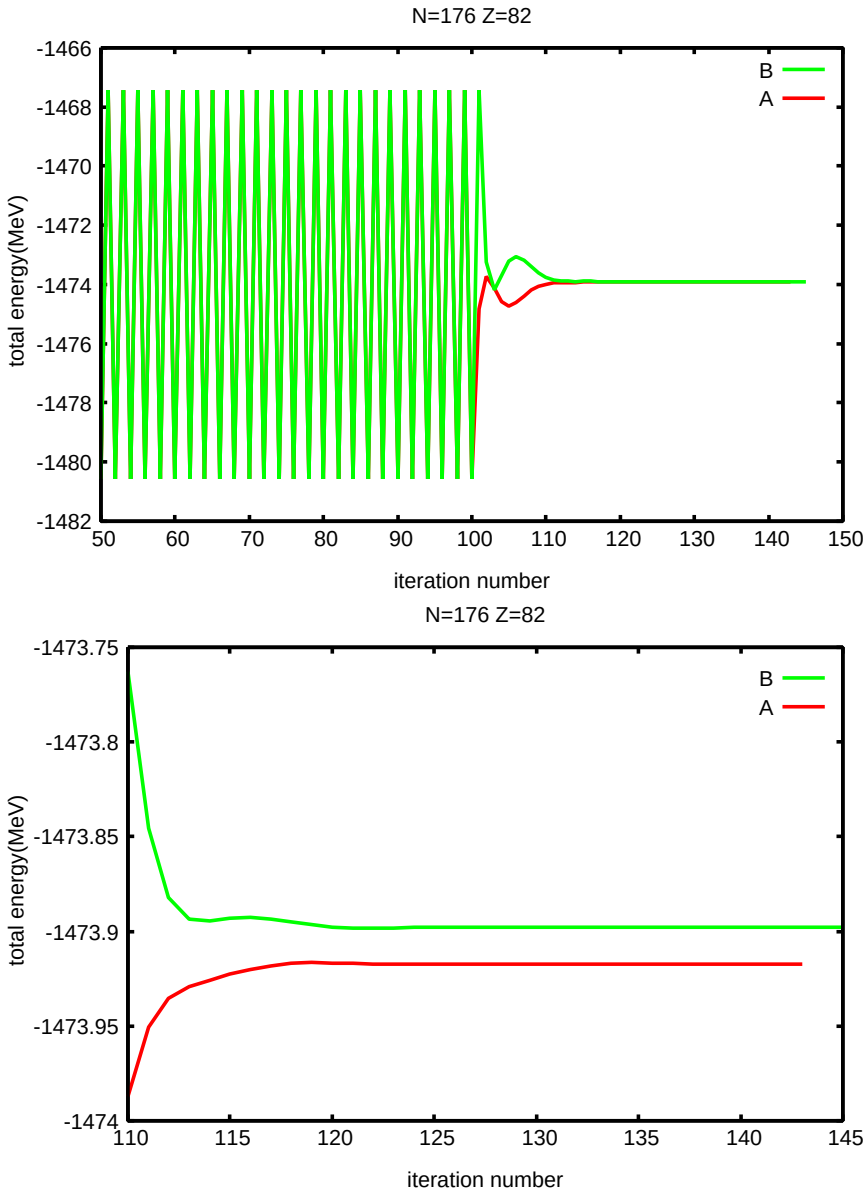


図 6.8: $\frac{1}{\beta} = 0.1$ の時の 1 核子準位のエネルギー

6.6　占拠数の固定

6.6.1　方法と結果

振動現象がおきたらその後の配位を固定させて、自己無撞着解に収束させれば振動現象を図 6.9 のように抑制できる。自動でさせるようにプログラムを修正するのがベストだが、今回はこの鉛の場合についてのみ手動で行うことにした。



A：100 回目の反復以降、配位を凍結させる
B：101 回目の反復以降、配位を凍結させる

図 6.9: 占拠数の固定

6.6.2 考察

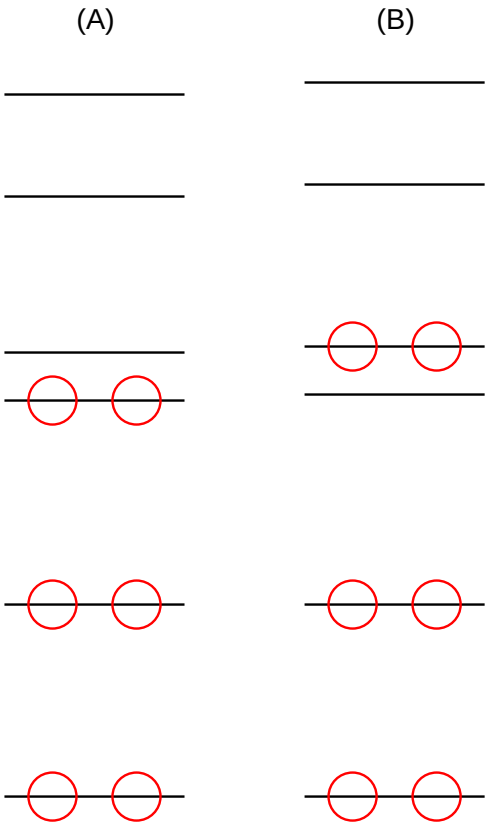


図 6.10: A と B の様子

図 6.10 のように、A の 100 回目の反復以降、配位を凍結させたものは、エネルギーが低い軌道から順に詰めた配位になっていることが分かる。B の 101 回目の反復以降、配位を凍結させたものは、間に空いた軌道の挟まった不安定な解となっていることが分かる。この結果より、A の場合の全ネネルギーが正しい値になっていることが導かれる。エネルギー準位が低い軌道は MeV 単位で変化しているのに対して、フェルミ面を挟む 2 軌道の差は 20KeV ほどしかなく非常に近くなっている。

なお、現象論的に対相関力を導入して、BCS 近似解を作っても振動を止めることができるが、本研究では模型を変更せずにできる方法についてのみ調べた。

第7章　まとめ

本論文では、三和氏の修士論文 [4] で開発された、原子核の相対論的平均場模型の解を求める数値計算プログラムについて理解を深め、正しさを検証した。続いて、動径グリッドや自己無撞着解を求める過程について見直しをして改良をした。

第２章では、特殊相対性理論の概念について説明し、非相対論である古典力学から、相対論的な考え方への変換を示した。続いて第３章では、非相対論的量子力学の方程式である Schrödinger 方程式に対して、どのように考えればスピン軌道力を自然に導入できるか議論するために、核子の従うべき相対論的量子力学の方程式である Dirac 方程式を導出を示した。

第４章では、本研究で使ったプログラムの考え方である原子核の相対論的平均場模型についての概念を説明した。ここでは、まず平均場模型に用いた２つの近似について論じた。第一の近似は、核子間の相互作用の媒介として導入された各種の中間子場のもつゆらぎを無視して平均値だけに注目するという、中間子の古典場である。第二の近似は、核子がスピン $\frac{1}{2}$ の粒子の従う相対論的な波動方程式である Dirac 方程式に従う場合にあらわれる負のエネルギーの解を無視するという no-sea 近似である。

第５章では、まずグリッド間隔と誤差の関係について調べ、最適なグリッド間隔を求めた。続いて、r 軸上のどこで大きな誤差が見られるかを明らかにし、非等間隔グリッドを導入することで誤差の軽減に成功した。

最後の第６章では、まず自己無撞着解を求めるアルゴリズムについて説明し、その計算の反復でおこることがある軌道の入れ替わりを抑制するための解決策を与えた。この振動は、フェルミ面を挟む２軌道のエネルギー準位が近いためおこることが分かっている。１つめの方法として、減衰因子の導入を行ったが結果は、振動の幅を抑えることはできたが止めることはできなかった。２つめの方法として、有限温度化を行った。ここでは、振動を抑制することが出来たが、有限温度で行ったため温度ゼロの場合の解を得られない。３つめの方法として占拠核子数の固定を行い、振動を抑制することができた。今回は中性子１７６個、陽子８２個の鉛の場合のみ検証したが、他の原子核でも同様に検証を行いプログラムで自動で振動現象をとめられるように改良するのが今後の研究を引き継ぐ者の課題である。

さらに付録として、改良をしたプログラムコードを付した。

参考文献

[1] 有馬朗人, 原子と原子核 (朝倉書店, 1982)

[2] 土岐博・保坂淳, 相対論的多体系としての原子核 (2008) (URL:http://www.rcnp.osaka-u.ac.jp/ hosaka/lecture/total.pdf)

[3] 山田昌平 福井大学工学部 物理工学科 卒業論文 「原子核の平均場模型」, 2007 年 2 月.

[4] 三和之浩 福井大学工学部 物理工学科 修士論文 「原子核の相対論的平均場模型プログラムの開発と検証」, 2008 年 2 月.

謝辞

本論文の作成にあたり、終始適切な助言を賜り、また丁寧に指導して下さった田嶋直樹先生に深く感謝いたします。また、林明久先生にも、日頃から研究の進み具合などを気にかけていただきました。ありがとうございます。

また、本研究にたくさんの意見を下さいました物理工学科の先生方にも感謝を申し上げ、謝辞とさせていただきます。

付録

以下は、等間隔グリッド版の原子核の相対論的平均場模型プログラムのソースコードリスティングである。三和氏の修士論文 [4] に掲載されたものから改良が進められ計算速度が数倍向上している。本卒業研究ではこのプログラムに様々な変更を加えて計算を行わせた。特に、関数 fermiDistribution、resetOccs、resetEs は卒業研究で新規に作成したものである。

グリッド間隔と誤差の関係を調べるために作成した修正版プログラムは掲載を割愛した。非等間隔グリッド版プログラムは改良作業を継続中なので掲載を見送った。

```
/*
Reletivistic mean-field model for spherical symmetric systems

2010/12/15      : rmf3h.c(ver.20) = starting point of rmf4a.c and rmf5a.c
2007/11/1,2,6,7 : rmf2c.c (self-consistent field calculation)
2007/10/10-23,31: rmf2b.c (for given Woods-Saxon form potential)
2007/9/3        : renamed from rmf1-z.c to rmf2a.c
2006/12/8,10-16 : created
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

// Arrays to store the energies and wavefunctions of the eigenstates
#define NEUTRON 0 // idxt = 0 for neutrons
#define PROTON 1  // idxt = 1 for protons
#define SIGMA 0   // idxb = 0 for sigma meson
#define OMEGA 1   // idxb = 1 for omega meson
#define RHO 2     // idxb = 2 for rho meson
#define COULOMB 3 // idxb = 3 for photon
#define NBOSON 4   // 0 <= idxb < NBOSON
#define IDXT 2    // idxt (index t) = 0..IDXT-1
                  //    idxt = NEUTRON (=0) : neutron
                  //    idxt = PROTON (=1) : proton
#define IDXP 2    // idxp (index p) = 0..IDXP-1
                  //    idxp = 0 : varpi=-1, L=J-1/2
                  //    idxp = 1 : varpi= 1, L=J+1/2
#define IDXJ 8    // idxj(index j)=0..IDXJ-1
                  //    J = idxj+1/2
#define NODE 5    // node=0..NODE-1
```

```
                  //    node=0 for the gournd state
#define IDXR 3001 // idxr(index r)=0..IDXR-1

                                        // "s" stands for "stored in memory"
double Es[IDXT][IDXP][IDXJ][NODE];      // energy (MeV)
double Occs[IDXT][IDXP][IDXJ][NODE];    // occupation number
double Fs[IDXT][IDXP][IDXJ][NODE][IDXR];// large component of the radial wavefunction
double Gs[IDXT][IDXP][IDXJ][NODE][IDXR];// small component of the radial wavefunction
                                        // r = idxr*dr  : idxr = 0..IDXR-1
double Vs[IDXT][2*IDXR-1];              // scalar potential (MeV) Vs[i] = Vs(r=dr*i/2)
double Vv[IDXT][2*IDXR-1];              // vector potential (MeV) Vv[i] = Vv(r=dr*i/2)
double Dens[2*IDXT][2*IDXR-1];          // Dens[0][idxrd] : neutron |F|^2+|G|^2
                                        // Dens[1][idxrd] : proton  |F|^2+|G|^2
                                        // Dens[2][idxrd] : neutron |F|^2-|G|^2
                                        // Dens[3][idxrd] : proton  |F|^2-|G|^2
                                        // r = idxrd*dr/2 : idxrd = 0..2*IDXR-2
double phi[NBOSON][2*IDXR-1];           // classical boson fields
double source[NBOSON][2*IDXR-1];        // source term of boson fields

// physical constants
const double HbC = 197.326960;          // h-bar c [MeV fm^2]
const double fineStructureConstant = 1.0/137.036000; // fine structure constant
const double Ebig  = 9.90e+30;
const double Ebigg = 9.99e+30; // must be bigger than Ebig.

// global variables
int idxtGlobal;
int NZ[IDXT]={126,82}; // the numbers of neutrons and protons (N and Z)
int imax[IDXT]; // the number of negative single-particle energy levels
int iterGlobal;
double temperature = 0.1; // [MeV], the only paramter for the Fermi distribution
double Mass[IDXT] = {939.56533, 938.27200}; // neutron and proton mass [MeV/c^2]
double BosonMass[NBOSON] = {520.0, 783.0, 770.0, 0.0}; // boson mass [MeV/c^2]
double squaredCoupling[NBOSON] = {109.626, 190.431, 16.3065, 0.0}; // [hbar*c]
                                                    // N.B.   16.3065=66.226/4
double Coupling[NBOSON]; // Coupling constant, to be defined in setParameters

double Rin;   // (fm) The radius of connection to the Taylor series around r=0.
double Rout;  // (fm)  The radius of the connection to an asymptotic solution for large r
double Rmax;  // (fm)  The maximum radius to calculate the asymptotic solution.
double Rmatch; // (fm) The radius where the forward and backward solutions meet.
double RadialGridSpacing; // (fm) Copy to a local variable "dr" and use "dr".

// paramters to construct the initial potential
double VsStr[IDXT] = {-373.0,-373.0}; // 398.5 ; // (MeV) Stength of the scalar potential
double VvStr[IDXT] = {274.0,274.0}; // 340.0 ; // (MeV) Strength of the vector potential
                 // Positive (negative) strengths mean repulsive (attractive) potentials.
double R_ws=7.0; // (fm) radius of Woods-Saxon potential, to be replaced with r0_ws*A^0.3
double r0_ws=1.2; // (fm) R=r0_ws*A^(1/3.0)
double a_ws=0.7; // (fm) surface thickness of the Woods-Saxon potential
```

```
struct RadialGridIndex {
  int i ; // inside point, where outward solution starts
  int m ; // matching point, where inward and outward solutions meet
  int o ; // outside point, where the inward solution starts
  int x ; // maximum radius point, to which the asymptotic solution is calculated
} Idxr;   // NB) IDXR is macro, Idxr is structure, idxr is int

typedef struct{
  double e ;   // enegy of this orbital
  double o ; // occupation number = the number of nucleons in this orbital
  int p ; // idxp
  int j ; // idxj
  int n ; // idxn
} orbital;

orbital e[IDXT][IDXP*IDXJ*NODE]; // properties of eigenstates of a nucleon

// prototype declarations of functions
int setParameters();
int writeParameters(FILE *FO);
int prog1();
int prog2(int idxt, orbital *e, int *imax, double *etot);
//int fermiDistribution(int idxt,int imax);
int writeWavefunction(FILE *FO,int idxt, int idxp, int idxj, int idxn);
// int swap : defined before function sort, only by which it is called.
int sort(orbital *e, int imax);
int resetOccs(int idxt);
int resetEs(int idxt);
int initialPotential(int sw);
int idx2qn(int sw,int idxp,int idxj, double *J,int *w,int *L,int *kappa);
int setRadialGrid(int sw);
double solveDirac(int idxp,int idxj,int idxn); // retuned value = Energy eigenvalue (MeV)
// int rungeBCi : defined before function runge, only by which it is called.
// int rungeBCo : defined before function runge, only by which it is called.
int runge(int sw, double E, int idxp, int idxj, int idxn, int *node, double*maco);
int Poisson(int imax,double h,double *source,double *phi);
int screenedPoisson(int imax,double h,double m,double source[],double phi[]);
int interpolate(double *y, int n);

//-------------------------------------------------------------------------------------
int main(int argc, char *argv[]){
  if(argc >= 2) NZ[NEUTRON]=atoi(argv[1]);
  if(argc >= 3) NZ[PROTON]=atoi(argv[2]);
  if(argc >= 2) printf("[ N=%d , Z=%d ]\n",NZ[NEUTRON],NZ[PROTON]);
  if(argc >= 4) temperature=atof(argv[3]);
  printf("Temperature=%f\n",temperature);
  setParameters();
  writeParameters(stdout);
  prog1();
```

```
}

//-------------------------------------------------------------------------------------------
int setParameters(){
  int idxb;
  for(idxb=0;idxb<NBOSON;idxb++){
    Coupling[idxb]=sqrt(squaredCoupling[idxb]*HbC);
  }
  Coupling[COULOMB]=sqrt(4*M_PI*HbC*fineStructureConstant);
}

//-------------------------------------------------------------------------------------------
int writeParameters(FILE *FO)
{ // writes the physical parameters of the model to a stream
  fprintf(FO,"neutron     mass=%10.4f (MeV/c^2)\n",Mass[NEUTRON]);
  fprintf(FO,"proton      mass=%10.4f (MeV/c^2)\n",Mass[PROTON]);
  fprintf(FO,"sigma meson mass=%10.4f (MeV/c^2)  g^2=%10.4f (hbar*c)\n"
                             ,BosonMass[SIGMA],squaredCoupling[SIGMA]);
  fprintf(FO,"omega meson mass=%10.4f (MeV/c^2)  g^2=%10.4f (hbar*c)\n"
                             ,BosonMass[OMEGA],squaredCoupling[OMEGA]);
  fprintf(FO,"rho   meson mass=%10.4f (MeV/c^2)  g^2=%10.4f (hbar*c)\n"
                             ,BosonMass[RHO  ],squaredCoupling[RHO  ]);
  fprintf(FO,"hbar*c=%12.6f (MeV*fm^2)\n",HbC);
  fprintf(FO,"fine structure constant=1/%12.6f\n",1.0/fineStructureConstant);
  fprintf(FO,"Coupling Constant : sigma=%.6e omega=%.6e rho=%.6e coulomb=%.6e\n"
  ,Coupling[SIGMA],Coupling[OMEGA],Coupling[RHO],Coupling[COULOMB]);
  //  fprintf(FO,"vector pot=%10.3f scalar pot=%10.3f (MeV)\n",VvStr,VsStr);
  //  fprintf(FO,"Woods-Saxon parameters: R=%10.5f a=%10.5f\n",R_ws,a_ws);
}

//-------------------------------------------------------------------------------------------
int prog1(){
/*
   Calculation of levels and densities fulfilling selfconsistency condition
*/

  int idxt,i,idxp,idxj,idxn,idxr,idxrd,degeneracy,num,iter;
  double E,sumspe[IDXT],eboson,etot;    // Energy [MeV], sum of single-particle energy
  double r,dr,drb; // radial coordinate (fm) and its grid spacing (drb=drb/2)
  int iprint;
  double diffVs[IDXT],diffVv[IDXT]; // maximum (for r) of the discrepancy in potentials
  double diff,diff2;
  double *Fsptr, *Gsptr;
  FILE *FFD, *FBD, *FPO, *FWF;

  FFD=NULL; // FFD=fopen("fdens.dat","w"); // Fermion (=nucleon) densities
  FBD=NULL; // FBD=fopen("bdens.dat","w"); // Boson densities
  FPO=NULL; // FPO=fopen("pot.dat","w");   // Potential energies
  FWF=NULL; // FWF=fopen("wf.dat","w");    // Nucleon wavefunctions
```

```
printf("[ N=%d , Z=%d ]\n",NZ[NEUTRON],NZ[PROTON]);
setRadialGrid(1);
initialPotential(1);

dr=RadialGridSpacing;
drb=dr/2;

if(FPO != NULL){
  fprintf(FPO,"# iter %d (initial potential)\n",0);
  iprint=0.1/drb; if(iprint<1) iprint=1;
  for(i=0;i<=2*Idxr.x;i++){
    if(i<=30||i % iprint == 0){
      fprintf(FPO,"%9.5f %.7e %.7e %.7e %.7e %.7e %.7e %.7e\n",i*drb
      ,Vs[NEUTRON][i],Vv[NEUTRON][i],Vs[PROTON][i],Vv[PROTON][i]
      ,0.0, 0.0, 0.0, 0.0);
    }
  }
  fprintf(FPO,"\n");
}

for(iter=1;iter<=1000;iter++){ // loop for selfconsistency : begin

  iterGlobal=iter;

  for(idxt=0;idxt<IDXT;idxt++) prog2(idxt,e[idxt],&imax[idxt],&sumspe[idxt]);
   printf("iter=%d prog2 finished. E=%e %e\n",iter,
           sumspe[NEUTRON],sumspe[PROTON]);
   // exit(1); // temporary modification
  printf("iter=%d finished\n",iter);

  printf("# J=idxj+0.5, L=idxj+idxp, varpi=2*idxp-1, kappa=varpi*(J+0.5)\n");
  for(idxt=0;idxt<IDXT;idxt++){
    num=0;
    double nocc=0;
    for(i=0;i<imax[idxt];i++){
      if(e[idxt][i].e < Ebig){
        degeneracy=2*e[idxt][i].j+2; num+=degeneracy;
        printf("%d idx(t,p,j,n)=%d %d %d %d E=m %+10.5f
                occ=%10.8f dgn=%2d sum=%3d :spl%3d\n",iter
               ,idxt,e[idxt][i].p,e[idxt][i].j,e[idxt][i].n
               ,e[idxt][i].e-Mass[idxt],e[idxt][i].o,degeneracy,num,i);
        nocc+=e[idxt][i].o;
        if(FWF != NULL){
          fprintf(FWF,"# %d %d %d %d %d\n"
                  ,idxt,i,e[idxt][i].p, e[idxt][i].j, e[idxt][i].n);
          Fsptr=Fs[idxt][e[idxt][i].p][e[idxt][i].j][e[idxt][i].n];
          Gsptr=Gs[idxt][e[idxt][i].p][e[idxt][i].j][e[idxt][i].n];
          for(idxr=0;idxr<=Idxr.x;idxr++){
            fprintf(FWF,"%9.5f %.5e %.5e\n",idxr*dr,Fsptr[idxr],Gsptr[idxr]);
          }
```

```
        fprintf(FWF,"\n");
      }

    }//>if(e)
  }//>for(i)
  if(fabs(nocc-NZ[idxt])>1.0e-12){
    printf("warning: sum of occupation number = %.8f
            error=%.3e\n",nocc,nocc-NZ[idxt]);
  }
}//>for(idxt)

if(FFD != NULL){
  for(idxrd=0;idxrd<=2*Idxr.x;idxrd++){
    r=idxrd*drb;
    fprintf(FFD,"%9.5f %.7e %.7e %.7e %.7e\n",r
    ,Dens[0][idxrd], Dens[1][idxrd], Dens[2][idxrd], Dens[3][idxrd]);
  }
  fprintf(FFD,"\n");
}

// boson field

for(i=0;i<=2*Idxr.x;i++){
  source[SIGMA][i] = (Dens[2][i]+Dens[3][i])*Coupling[SIGMA];
  source[OMEGA][i] = (Dens[0][i]+Dens[1][i])*Coupling[OMEGA];
  source[RHO][i] = (Dens[0][i]-Dens[1][i])*Coupling[RHO];
  source[COULOMB][i] = Dens[1][i]*Coupling[COULOMB];
}

screenedPoisson(2*Idxr.x+1,drb,BosonMass[SIGMA]/HbC,source[SIGMA],phi[SIGMA]);
screenedPoisson(2*Idxr.x+1,drb,BosonMass[OMEGA]/HbC,source[OMEGA],phi[OMEGA]);
screenedPoisson(2*Idxr.x+1,drb,BosonMass[RHO]/HbC,source[RHO],phi[RHO]);
Poisson(2*Idxr.x+1,drb,source[COULOMB],phi[COULOMB]);

if(FBD != NULL){
  for(i=0;i<=2*Idxr.x;i++){
    r=i*drb;
    fprintf(FBD,"%9.5f %.7e %.7e %.7e %.7e\n",r
    ,phi[SIGMA][i],phi[OMEGA][i],phi[RHO][i],phi[COULOMB][i]);
  }
  fprintf(FBD,"\n");
}

// total energy

eboson=0;
for(i=1;i<=2*Idxr.x;i++){
  eboson+=i*i*(
              -phi[SIGMA  ][i]*source[SIGMA  ][i]
              +phi[OMEGA  ][i]*source[OMEGA  ][i]
```

```
                +phi[RHO    ][i]*source[RHO    ][i]
                +phi[COULOMB][i]*source[COULOMB][i]
               );
}
eboson*=4*M_PI*drb*drb*drb;
etot=sumspe[NEUTRON]+sumspe[PROTON]-eboson/2;
printf("%d %.6f %.6f %.6f %.6f :total energy(MeV)\n",iter
  ,etot,sumspe[NEUTRON],sumspe[PROTON],eboson);

// new potential for the next iteration

for(idxt=0;idxt<IDXT;idxt++){diffVs[idxt]=0; diffVv[idxt]=0;}
if(FPO != NULL) fprintf(FPO,"# iter %d\n",iter);
for(i=0;i<=2*Idxr.x;i++){
  double Vsigma, Vomega, Vrho, Vcoul, Vsnew[IDXT], Vvnew[IDXT];
  const double p=0.5; // damping factor

  Vsigma=-Coupling[SIGMA]  *phi[SIGMA][i]  ;
  Vomega= Coupling[OMEGA]  *phi[OMEGA][i]  ;
  Vrho  = Coupling[RHO]    *phi[RHO][i]    ;
  Vcoul = Coupling[COULOMB]*phi[COULOMB][i];
  Vsnew[NEUTRON]=Vsigma;
  Vsnew[PROTON ]=Vsigma;
  Vvnew[NEUTRON]=Vomega+Vrho;
  Vvnew[PROTON ]=Vomega-Vrho+Vcoul;
  if(FPO != NULL){
    if(i<=30||i % iprint == 0){
      fprintf(FPO,"%9.5f %.7e %.7e %.7e %.7e %.7e %.7e %.7e\n",i*drb
      ,Vsnew[NEUTRON],Vvnew[NEUTRON],Vsnew[PROTON],Vvnew[PROTON]
      ,Vsigma,Vomega,Vrho,Vcoul);
    }
  }
  for(idxt=0;idxt<IDXT;idxt++){
    diff=fabs(Vsnew[idxt]-Vs[idxt][i]); if(diffVs[idxt]<diff) diffVs[idxt]=diff;
    diff=fabs(Vvnew[idxt]-Vv[idxt][i]); if(diffVv[idxt]<diff) diffVv[idxt]=diff;
    Vs[idxt][i]=(1-p)*Vs[idxt][i]+p*Vsnew[idxt];
    Vv[idxt][i]=(1-p)*Vv[idxt][i]+p*Vvnew[idxt];
  }
}

diff  = diffVs[NEUTRON] ;
diff2 = diffVs[PROTON]  ; if ( diff2 > diff ) diff = diff2 ;
diff2 = diffVv[NEUTRON] ; if ( diff2 > diff ) diff = diff2 ;
diff2 = diffVv[PROTON]  ; if ( diff2 > diff ) diff = diff2 ;

printf("max (for r) potential
        inconsistency(Vs(n),Vv(n),Vs(p),Vv(p),all) (MeV)\n");
printf("%e %e %e %e :pd\n"
  ,diffVs[NEUTRON],diffVv[NEUTRON],diffVs[PROTON],diffVv[PROTON],diff);
if(FPO != NULL) fprintf(FPO,"\n");
```

```
    if(diff<1.0e-8) {
      printf("selfconsistency fulfilled: iter %d,
              potential difference< %e\n",iter,diff);
      break;
    }

  } // for(iter): loop for selfconsistency : end

  if(FFD != NULL || FBD != NULL){
    for(i=0;i<=2*Idxr.x;i++){
      r=i*drb;
      if(FFD != NULL) fprintf(FFD,"%9.5f %.7e %.7e %.7e %.7e\n",r
      ,Dens[0][i], Dens[1][i], Dens[2][i], Dens[3][i]);
      if(FBD != NULL) fprintf(FBD,"%9.5f %.7e %.7e %.7e %.7e\n",r
      ,phi[SIGMA][i],phi[OMEGA][i],phi[RHO][i],phi[COULOMB][i]);
    }
    if(FFD != NULL) fprintf(FFD,"\n");
    if(FBD != NULL) fprintf(FBD,"\n");
  }

  if(FFD != NULL) fclose(FFD);
  if(FBD != NULL) fclose(FBD);
  if(FPO != NULL) fclose(FPO);
  if(FWF != NULL) fclose(FWF);

  printf("%d %d %d %d %d %f %f %f %25.15f :final\n"
         ,NZ[NEUTRON],N[PROTON]
         ,Idxr.i,Idxr.m,Idxr.o,Rin,Rout,RadialGridSpacing,etot);

  return 0;
}

//-------------------------sort energy and calculate density-----------------------------
int prog2(int idxt, orbital *e, int *imaxPtr, double *etot){
  int idxp, idxj, idxn;
  int i,n,fn,idxr,idxrd;
  double r, dr, E, *Fsptr, *Gsptr;

  idxtGlobal=idxt;

  // energy sweep (for a test of this code)
  if(0){
    int mxi=200, node ; double maco;
    int idxp=0; int idxj=0; int idxn=0;
    for(i=0;i<mxi;i++){
      E=Mass[idxt]-mxi+i;
      runge(0,E,idxp,idxj,idxn,&node,&maco);
    }
  }
```

```
if(iterGlobal<=100){

// calculation of the energies and wavefunctions of the eigenstates
// for all the combinations of the quantum numbers
resetEs(idxt);
i=0;
for(idxp=0;idxp<IDXP;idxp++){
  for(idxj=0;idxj<IDXJ;idxj++){
    for(idxn=0;idxn<NODE;idxn++){
      E=solveDirac(idxp, idxj, idxn);
      if(E >= Ebig) break;
      e[i].e=E;
      e[i].p=idxp;
      e[i].j=idxj;
      e[i].n=idxn;
      i++;
    } //>for(idxn)
  } //>for(idxj)
} //>for(idxp)
*imaxPtr=i;

// sorting the eigenstates in the asceding order of energy
sort(e,*imaxPtr);

// determination of the occupation number

//if(iterGlobal<=101){

resetOccs(idxt);
fn=NZ[idxt];
for(i=0;i<*imaxPtr;i++){
  if(fn>0) {
    n=2*e[i].j+2;
    if(fn<n){
      n=fn;
    }
    e[i].o=n;
    Occs[idxt][e[i].p][e[i].j][e[i].n]=n;
    fn-=n;
  }
  else {
    e[i].o=0;
    Occs[idxt][e[i].p][e[i].j][e[i].n]=0;
  }
}//>for(i)
if(fn>0){
  fprintf(stderr,"error in calculating density:
          insufficient number of levels\n");
  exit(1);
```

```
  }
}

  //fermiDistribution(idxt,*imaxPtr);

  // calculation of the densities
  double f6 =  4.0/3.0;
  double f7 = -1.0/3.0;
  dr=RadialGridSpacing/2;

  for(idxrd=0;idxrd<=2*Idxr.x;idxrd++){
    Dens[idxt][idxrd]=0.0;
    Dens[idxt+IDXT][idxrd]=0.0;
  }

  *etot=0; // sum of single-particle energies
  for(i=0;i<*imaxPtr;i++){double occ;
    occ=e[i].o;
    if(occ == 0.0) continue;
    *etot+=occ*e[i].e;
    Fsptr=Fs[idxt][e[i].p][e[i].j][e[i].n];
    Gsptr=Gs[idxt][e[i].p][e[i].j][e[i].n];
    for(idxr=1;idxr<=Idxr.x;idxr++){double Ft,Gt;
      idxrd=2*idxr;
      r=idxrd*dr;
      Ft=Fsptr[idxr];
      Gt=Gsptr[idxr];
      Dens[idxt     ][idxrd]+=(Ft*Ft+Gt*Gt)*occ/(4.0*M_PI*r*r);
      Dens[idxt+IDXT][idxrd]+=(Ft*Ft-Gt*Gt)*occ/(4.0*M_PI*r*r);
    }

    Dens[idxt][0]=f6*Dens[idxt][2]+f7*Dens[idxt][4];
    Dens[idxt+IDXT][0]=f6*Dens[idxt+IDXT][2]+f7*Dens[idxt+IDXT][4];

  }//>for(i)
  // printf("%d %f\n",NZ[idxt],Mass[idxt]);
  *etot-=NZ[idxt]*Mass[idxt];
  // printf("%f\n",*etot);

  interpolate(Dens[idxt],2*Idxr.x+1);
  interpolate(Dens[idxt+IDXT],2*Idxr.x+1);

  // fprintf(stderr,"%f %f %f %f #3#\n"
             ,Dens[0][0],Dens[1][0],Dens[2][0],Dens[3][0]);

  //return 0;
}

//-----------------------------------------------------------------------------
/*int fermiDistribution(int idxt,int imax)
```

```
{
  int i,j,fn;
  double n,nsum,m,l;
  double beta=1/temperature;
  double eflo,efhi,ef;
  const int debug = 0;

  eflo=-100.0;
  if(eflo > e[idxt][0].e) eflo = e[idxt][0].e;
  efhi=0.0;
  if(efhi < e[idxt][imax-1].e) efhi = e[idxt][imax-1].e;
  efhi+=1.0+temperature*37;
  ef=0.0;
  fn=NZ[idxt];

  // resetOccs(idxt); // removed line

  if(debug) printf("%d %f\n",NZ[idxt],beta);

  for(j=0;j<54;j++){ //bisectionMethod

      ef=(eflo+efhi)/2;
      nsum=0.0;

    for(i=0;i<imax;i++){

      n=(2*(e[idxt][i].j)+2)*(1/(1+exp(beta*(((e[idxt][i].e)-Mass[idxt])-ef))));
      m=(1/(1+exp(beta*(((e[idxt][i].e)-Mass[idxt])-ef))));
      l=(2*(e[idxt][i].j)+2);
      nsum+=n;
      if(debug) printf("%d %d n=%f nsum=%f m=%f l=%f\n",i,j,n,nsum,m,l);
    }

    if(debug) printf("ef=%f\n",ef);

    if(nsum<fn){eflo=ef;}
    else if(nsum>fn){efhi=ef;}
    else {break;}

  }

  ef=(eflo+efhi)*0.5; // modified

  resetOccs(idxt);

  nsum=0;
  for(i=0;i<imax;i++){

    n=(2*(e[idxt][i].j)+2)*(1/(1+exp(beta*(((e[idxt][i].e)-Mass[idxt])-ef))));
        //fermidistribution
```

```
    m=(1/(1+exp(beta*(((e[idxt][i].e)-Mass[idxt])-ef))));
    l=(2*(e[idxt][i].j)+2);

    e[idxt][i].o=n;
    Occs[idxt][e[idxt][i].p][e[idxt][i].j][e[idxt][i].n]=n;
    nsum+=n;
    if(debug) printf("%1d %3d %1d %1d %1d"
             ,idxt,i,e[idxt][i].p,e[idxt][i].j,e[idxt][i].n); // for debug
    if(debug) printf(" E= %9.4f %.1e occ=%13.10f %.1e l=%4.1f nsum=%14.10f\n"
    ,e[idxt][i].e-Mass[idxt]
    ,e[idxt][i].e-Es[idxt][e[idxt][i].p][e[idxt][i].j][e[idxt][i].n]
    ,e[idxt][i].o
    ,e[idxt][i].o-Occs[idxt][e[idxt][i].p][e[idxt][i].j][e[idxt][i].n]
    ,l
    ,nsum); // for debug
  }
    //exit(1);
}
*/
//-------------------------------------------------------------------------------
int writeWavefunction(FILE *FO, int idxt, int idxp, int idxj, int idxn)
{ // writes the wavefunction to a stream
  int i;
  double r1, F, G, *Fsptr, *Gsptr;

  Fsptr=Fs[idxt][idxp][idxj][idxn]; Gsptr=Gs[idxt][idxp][idxj][idxn];
  F=0;G=0;
  for(i=0;i<=Idxr.x;i++){
    if(F<fabs(Fsptr[i]))F=fabs(Fsptr[i]);
    if(G<fabs(Gsptr[i]))G=fabs(Gsptr[i]);
  }
  fprintf(FO,"# (idxp,idxj,idxn)=(%d %d %d) E=%16.8f\n"
          ,idxp,idxj,idxn,Es[idxt][idxp][idxj][idxn]);
  fprintf(FO,"# r(fm) F/%12e G/%12e Vve Vsc\n",F,G);
  F=1.0/F; G=1.0/G;
  for(i=0;i<=Idxr.x;i++){
    r1=RadialGridSpacing*i;
    fprintf(FO,"%8.4f %16.8e %16.8e %16.8e %16.8e\n"
           ,r1,Fsptr[i]*F,Gsptr[i]*G,Vv[idxt][2*i],Vs[idxt][2*i]);
  }
}

//-------------------------------------------------------------------------------
inline int swap(orbital *e, int i, int j){  // called only by function "sort"
  orbital tmp;
  tmp  = e[i];
  e[i] = e[j];
  e[j] = tmp;
}
```

```
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
int sort(orbital *e, int imax){
  int idxt, i, j;

  for(i=0;i<imax-1;i++){
    for(j=i+1;j<imax;j++){
      if(e[i].e>e[j].e){
          swap(e, i, j);
      }
    }
  }
}

//-------------------------------------------------------------------------------------
int resetOccs(int idxt_arg){
// Zero clear a global array Occs for the first index is equal to idxt_arg.
// If idxt_arg >= IDXT, the zero clear is done for all the values of idxt.
  int idxt,idxp,idxj,idxn;
  for(idxt=0;idxt<IDXT;idxt++){
    if(idxt == idxt_arg || idxt_arg >= IDXT){
      for(idxp=0;idxp<IDXP;idxp++){
        for(idxj=0;idxj<IDXJ;idxj++){
          for(idxn=0;idxn<NODE;idxn++){
            Occs[idxt][idxp][idxj][idxn]=0.0;
          }
        }
      }
    }
  }
}

//-------------------------------------------------------------------------------------
int resetEs(int idxt_arg){
  int idxt,idxp,idxj,idxn;
  for(idxt=0;idxt<IDXT;idxt++){
    if(idxt == idxt_arg || idxt_arg >= IDXT){
      for(idxp=0;idxp<IDXP;idxp++){
        for(idxj=0;idxj<IDXJ;idxj++){
          for(idxn=0;idxn<NODE;idxn++){
            Es[idxt][idxp][idxj][idxn]=Ebigg; // means not calculated
          }
        }
      }
    }
  }
}

//-------------------------------------------------------------------------------------
int initialPotential(int sw) {
// 0th compoment of vector potential (MeV) and scalar potential (MeV)
```

```
  int i,idxt;
  double r,dr;
  double A,R,rhov,rhos,rho3,t1,t2,Vv0[IDXT],Vs0[IDXT],Vc0; // local variable
  // external variables
  // r0_ws,a_ws,NZ[]

  A=NZ[NEUTRON]+NZ[PROTON];
  R=R_ws; // =r0_ws*pow(A,1/3.0), though optimal value may be slightly different...
  rhov=0.16;
  rhos=rhov;
  rho3=rhov*(NZ[NEUTRON]-NZ[PROTON])/A;
  t1=pow(Coupling[OMEGA]*HbC/BosonMass[OMEGA],2)*rhov;
  t2=pow(Coupling[RHO]*HbC/BosonMass[RHO],2)*rho3;
  Vv0[NEUTRON]=t1+t2;
  Vv0[PROTON ]=t1-t2;
  Vs0[NEUTRON]=-pow(Coupling[SIGMA]*HbC/BosonMass[SIGMA],2)*rhos;
  Vs0[PROTON ]=Vs0[NEUTRON];
  Vc0=NZ[PROTON]*fineStructureConstant*HbC;
  dr=RadialGridSpacing/2;
  for(idxt=0;idxt<IDXT;idxt++){
    for(i=0;i<=2*Idxr.x;i++){
      r=i*dr;
      Vv[idxt][i]=Vv0[idxt]/((1.0+exp((r-R)/a_ws))*(1.0+exp((-r-R)/a_ws)));
      Vs[idxt][i]=Vs0[idxt]/((1.0+exp((r-R)/a_ws))*(1.0+exp((-r-R)/a_ws)));
      if(idxt == PROTON){
        if(r<=R){
          Vv[idxt][i]+=0.5*Vc0/R*(3-pow(r/R,2));
        }
        else {
          Vv[idxt][i]+=Vc0/r;
        }
      }
    }
    if(sw>0){
      printf("vector potential: t,strength,R,a=%d %f %f %f\n"
        ,idxt,VvStr[idxt],R_ws,a_ws);
      printf("scalar potential: t,strength,R,a=%d %f %f %f\n"
        ,idxt,VsStr[idxt],R_ws,a_ws);
    }
  }
  return 0;
}

//-----------------------------------------------------------------------------------------
// A set of functions to solve the Dirac eigenvalue equation with spherically symmetric
// vector and scalar potentials which are regular at radius zero.
//   * idx2qn
//   * setRadialGrid
//   * solveDirac
//   * rungeBCi
```

```
//   * rungeBCo
//   * (rungeFun1 and rungeFun2 :commented out)
//   * runge
//-------------------------------------------------------------------------------------
int idx2qn(int sw,int idxp,int idxj, double *J,int *w,int *L,int *kappa){
// converts the indices into quantum numbers
  *J=idxj+0.5; // J = angular momentum
  if(idxp == 0){*w=-1;} else {*w=1;} // w = quantum number "varpi"
  *L=idxj+(1+ *w)/2; // L=J+varpi/2 : orbital angular momentum
  *kappa=*w *(idxj+1); // kappa = varpi*(J+1/2)
  if(sw > 0){
     fprintf(stderr,"quantum numbers: J=%3d/2  varpi=%d  L=%d  kappa=%d\n"
             ,(int)(2*(*J)),*w,*L,*kappa);
   return 0;
  }
}

//-------------------------------------------------------------------------------------
int setRadialGrid(int sw)
{ // sets up the parameters of the radial grid to express the radial wavefunctions
  double dr, A;

  Idxr.i=1;  // For radial grid number <= Idxr.i, rungeBCi is used.
  Idxr.o=500; // For radial grid number >= Idxr.o, rungeBCo is used.
  Rout=25.0;  // (fm) corresponding to Idxr.o
  Rmax=30.0;  // (fm) corresponding to Idxr.x

  if(1 > Idxr.i || Idxr.i > Idxr.o || Idxr.o >= IDXR || Rout <= 0.0){
    fprintf(stderr,"setRadialGrid: error: Idxr.i,Idxr.o,Rout %d %d %f\n"
            ,Idxr.i,Idxr.o,Rout);
    exit(1);
  }

  A=NZ[NEUTRON]+NZ[PROTON];
  R_ws=r0_ws*pow(A,1/3.0);

  dr=Rout/Idxr.o; RadialGridSpacing=dr;
  Rin=Idxr.i*dr;
  Idxr.m=(R_ws+3*a_ws)/dr+0.5;
  if(Idxr.m < Idxr.i) Idxr.m = Idxr.i;
  if(Idxr.m > Idxr.o) Idxr.m = Idxr.o;
  Rmatch=Idxr.m*dr;
  Idxr.x = floor(Rmax/dr+0.5);
  if(Idxr.x > IDXR-1) Idxr.x=IDXR-1;
  Rmax=Idxr.x*dr;

  if(sw>0){
    printf("   R(in,match,out,max)=%9.5f %9.5f %9.4f %9.4f (fm)
               \n",Rin,Rmatch,Rout,Rmax);
    printf("idxr(in,match,out,max)=%9d %9d %9d %9d dr=%15.10f (fm)\n"
```

```
          ,Idxr.i,Idxr.m,Idxr.o,Idxr.x,dr);
  }
}

//-------------------------------------------------------------------------------------
double solveDirac(int idxp,int idxj,int idxn)
{ // calculates the energy and the wavefunction of the eigenstate
  int i,idxt,node;
  double maco,maco1,maco2; // matching condition is "maco=0"
  double E,E1,E2,dE, Eprec=1.0e-14; // energy (MeV)
  idxt = idxtGlobal;

  runge(0,Mass[idxt]-0.001,idxp,idxj,idxn,&node,&maco);

  //  printf("solveDirac: runge done.\n"); // #1#

  if(node < idxn || node == idxn && maco < 0){
    // fprintf(stderr,"no bound states: For E=Mass
                , node=%d maco=%e\n",node,maco);
    E=Ebig;
    goto fin;
  }

  E1=Mass[idxt]-30.0; dE=10.0;
  for(;;){
    runge(0,E1,idxp,idxj,idxn,&node,&maco);

    //    printf("solveDirac: runge done in for(;;). E1=%f dE=%f node=%d maco=%f\n"
    //   ,E1,dE,node,maco); // #2#

    if(node < idxn || node == idxn && maco < 0) break;
    E1=E1-dE; dE*=1.25;
  }
  // fprintf(stderr,"lower bound (E,node,maco)=(%f %d %e)\n",E1,node,maco);

  E2=E1*0.8+Mass[idxt]*0.2;
  for(;;){
    runge(0,E2,idxp,idxj,idxn,&node,&maco);
    if(node > idxn || node == idxn && maco > 0) break;
    E2=E2*0.8+Mass[idxt]*0.2;
    if(Mass[idxt]-E2 < 0.001) { // Actually,this condition was alread checked.
      fprintf(stderr,"no bound states(2): For E=Mass
              , node=%d maco=%e\n",node,maco);
      E=Ebig;
      goto fin;
    }
  }
  // fprintf(stderr,"upper bound (E,node,maco)=(%f %d %e)\n",E2,node,maco);

  // bisection method
```

```
  for(i=0;i<60;i++){
    if(E2-E1<=Mass[idxt]*Eprec) break;
    E=(E1+E2)*0.5;
    runge(0,E,idxp,idxj,idxn,&node,&maco);
    // fprintf(stderr,"%12f %5d %15.12f %12f %12f\n",E,node,maco,E1,E2);
    if(node < idxn || node == idxn && maco < 0){E1=E;} else {E2=E;}
  }
  E=(E1+E2)*0.5;
  runge(1,E,idxp,idxj,idxn,&node,&maco);
fin:
  Es[idxt][idxp][idxj][idxn]=E;
  // fprintf(stderr,"iter=%d E=%f\n",i,Es[idxt][idxp][idxj][idxn]);
  return E;
}

//-------------------------------------------------------------------------------------------
int rungeBCi(double r, double E, int idxp, int L, double *F, double *G){
// Boundary Condition for r -> 0
  double vv,vs,epsilon,mu,rL,rL1,rL2,rL3;
  int idxt;
  idxt=idxtGlobal; // A global variable is copied to a local variable.

  if(r <= 0.0){
    *F=0; *G=0;
    if(r == 0) return 0; else return 1;
  }
  vv=Vv[idxt][0]; vs=Vs[idxt][0];
  epsilon=(E-vv)/HbC;
  mu=(Mass[idxt]+vs)/HbC;
    {//input : int L, double r,  output : double rL=pow(r,L)
      int pt=L; double xt=r, rt=1.0;
      while(pt != 0){if(pt & 1) rt*=xt; xt*=xt; pt>>=1;} rL=rt;
    }
  rL1=rL*r; rL2=rL1*r; rL3=rL2*r;
  *F=rL1-(epsilon*epsilon-mu*mu)/(4*L+6)*rL3;
  if(idxp == 0){ // varpi=-1, L=J-1/2=0,1,2,...
    *G=-(epsilon-mu)/(2*L+3)*rL2;
  }
  else { // varpi=1, L=J+1/2=1,2,3,...
    *G=(2*L+1)/(epsilon+mu)*rL-(epsilon-mu)/(2*L+3)*rL2;
  }
  return 0;
}

//-------------------------------------------------------------------------------------------
int rungeBCo(double r, double E, double *F, double *G){
// Boundary Condition for r -> infinity
  double Mass1;
  Mass1=Mass[idxtGlobal];
```

```
  *F=exp(-r*sqrt(Mass1*Mass1-E*E)/HbC);
  *G=-sqrt((Mass1-E)/(Mass1+E))*exp(-r*sqrt(Mass1*Mass1-E*E)/HbC);
  return 0;
}

//-------------------------------------------------------------------------------------
#define RUNGE_OPTION 3

#if RUNGE_OPTION == 1
  // - - - - - - - - - - - - - - - function version - - - - - - - - - - - - - -
  inline double rungeFun1(double rinv,double F,double G,double E
                          ,int kappa,double V){
    return (E-V+Mass[idxtGlobal])*G*(1.0/HbC)-kappa*F*rinv;
  }
  inline double rungeFun2(double rinv,double F,double G,double E
                           ,int kappa,double V){
    return -(E-V-Mass[idxtGlobal])*F*(1.0/HbC)+kappa*G*rinv;
  }
#elif RUNGE_OPTION == 2
  // - - - - - - - - - - - - - - - - macro version - - - - - - - - - - - - - - -
  #define rungeFun1(rinv, F, G, E, kappa, V)\
    ((-(kappa))*(rinv)*(F)+(1.0/HbC)*((E)-(V)+Mass[idxtGlobal])*(G))
  #define rungeFun2(rinv, F, G, E, kappa, V)\
    ((1.0/HbC)*(-((E)-(V)-Mass[idxtGlobal]))*(F)+(kappa)*(rinv)*(G))
#elif RUNGE_OPTION == 3
  // - - - - - - - - - - - - - - - - inline version - - - - - - - - - - - - - - -
  //   Inline version uses neither functions nor macros.
#endif

int runge(int sw, double E, int idxp, int idxj, int idxn, int *node, double*maco){
// Runge-Kutta method for radial wave fn.
// sw & 0x01 > 0 --> the wavefunciton iwws stored in the global arrays
// sw & 0x02 > 0 --> fprintf(stderr) some information

/*

used global variables
     Fs, Gs, Mass, idxtGlobal, RadialGridSpacing
used functions
    rungeBCi, rungeBCo, idx2qn

*/

  int i,is,ii,i2,i3;
  int w,L,kappa;
  int forbac; // 0 for forward solution, 1 for backward solution
  int idxt;
  double *Fsa,*Gsa;
  double r1,r1inv;
  double k1,l1,k2,l2,k3,l3,k4,l4,k,l;
```

```
  double F,F2,F3,F4,oldF, G,G2,G3,G4;
  double dr, pmdr, halfpmdr; // pmdr = plus or minus dr
  double J, vv, vs;
  double Fm[2], Gm[2];
  double c1,c2,c3,c4,c5,c6,c7;

  idxt=idxtGlobal;
  Fsa=Fs[idxt][idxp][idxj][idxn];
  Gsa=Gs[idxt][idxp][idxj][idxn];

  idx2qn(0,idxp,idxj, &J,&w,&L,&kappa);

  dr=RadialGridSpacing;

  if(sw & 0x01){
    for(i=0;i<=Idxr.i;i++){
      r1=i*dr;
      rungeBCi(r1,E,idxp,L,&Fsa[i],&Gsa[i]);
    }
    for(i=Idxr.x;i>=Idxr.o;i--){
      r1=i*dr;
      rungeBCo(r1,E,&Fsa[i],&Gsa[i]);
    }
  }

  *node=0;

  for(forbac=0; forbac<2; forbac++){

    if(forbac == 0){
      rungeBCi(Idxr.i*dr,E,idxp,L,&F,&G);
      pmdr=dr; is=Idxr.i; ii=1;
    }
    else {
      rungeBCo(Idxr.o*dr,E,&F,&G);
      pmdr=-dr; is=Idxr.o; ii=-1;
    }

    c1=pmdr*kappa;
    c2=pmdr*(1.0/HbC);
    c3=E+Mass[idxt];
    c4=E-Mass[idxt];
    halfpmdr=pmdr*0.5;

    r1=is*dr; r1inv=1.0/r1; c5=c1*r1inv;
    i2=is*2; vs=Vs[idxt][i2]; vv=Vv[idxt][i2];
    for(i=is+ii ; ; i+=ii){
      i2=i*2;
#if ( RUNGE_OPTION == 1 ) || ( RUNGE_OPTION == 2 )
  //----------------------- function and macro versions -----------------------
```

```
      k1=rungeFun1(r1inv,F,G,E,kappa,vv-vs)*pmdr;
      l1=rungeFun2(r1inv,F,G,E,kappa,vv+vs)*pmdr;
        r1=r1+halfpmdr; r1inv=1.0/r1;
        i3=i2-ii; vs=Vs[idxt][i3]; vv=Vv[idxt][i3];
        F2=F+0.5*k1; G2=G+0.5*l1;
      k2=rungeFun1(r1inv,F2,G2,E,kappa,vv-vs)*pmdr;
      l2=rungeFun2(r1inv,F2,G2,E,kappa,vv+vs)*pmdr;
        F3=F+0.5*k2; G3=G+0.5*l2;
      k3=rungeFun1(r1inv,F3,G3,E,kappa,vv-vs)*pmdr;
      l3=rungeFun2(r1inv,F3,G3,E,kappa,vv+vs)*pmdr;
        r1=dr*i; r1inv=1.0/r1;
        vs=Vs[idxt][i2]; vv=Vv[idxt][i2];
        F4=F+k3; G4=G+l3;
      k4=rungeFun1(r1inv,F4,G4,E,kappa,vv-vs)*pmdr;
      l4=rungeFun2(r1inv,F4,G4,E,kappa,vv+vs)*pmdr;
#elif RUNGE_OPTION == 3
  //-----------------------------  inline version ------------------------------
      k1=c2*(c3-vv+vs)*G-c5*F;
      l1=c5*G-c2*(c4-vv-vs)*F;
        r1=r1+halfpmdr; c5=c1/r1;
        i3=i2-ii; vs=Vs[idxt][i3]; vv=Vv[idxt][i3];
        F2=F+0.5*k1; G2=G+0.5*l1;
        c6=c2*(c3-vv+vs);
        c7=-c2*(c4-vv-vs);
      k2=c6*G2-c5*F2;
      l2=c5*G2+c7*F2;
        F3=F+0.5*k2; G3=G+0.5*l2;
      k3=c6*G3-c5*F3;
      l3=c5*G3+c7*F3;
        r1=dr*i; c5=c1/r1;
        vs=Vs[idxt][i2]; vv=Vv[idxt][i2];
        F4=F+k3; G4=G+l3;
      k4=c2*(c3-vv+vs)*G4-c5*F4;
      l4=c5*G4-c2*(c4-vv-vs)*F4;
#endif
      k=(k1+2*k2+2*k3+k4)*(1.0/6.0);
      l=(l1+2*l2+2*l3+l4)*(1.0/6.0);
      oldF=F;
      F=F+k;
      G=G+l;
      if(F*oldF<0) (*node)++;
      if(sw & 0x01){
        Fsa[i]=F;
        Gsa[i]=G;
      }
      if(i==Idxr.m) break;
    }
    Fm[forbac]=F; Gm[forbac]=G;

  } // end of for(forbac)
```

```
  *maco=Fm[0]*Fm[1]*(Fm[0]*Gm[1]-Gm[0]*Fm[1]);

  //fprintf(stderr,"%12f %5d %15e E,node,maco\n",E,*node,*maco);

  //---------------- normalization of the wavefunction : begin ----------------
  if(sw & 0x01)
  { double s,s1,s2,t1,t2,mf,nf1,nf2;

    s1=0.0;
    for(i=0;i<Idxr.m;i++){
      t1=Fsa[i];
      t2=Gsa[i];
      s1+=t1*t1+t2*t2;
    }
    s2=0.0;
    for(i=Idxr.m;i<=Idxr.x;i++){
      t1=Fsa[i];
      t2=Gsa[i];
      s2+=t1*t1+t2*t2;
    }
    mf=Fm[0]/Fm[1]; // It is preferable to choose between Fin/Fout and Gin/Gout
                    // the one which suffers from less numeircal error.
    s=dr*(s1+s2*mf*mf);
    nf1=1/sqrt(s);  // sign of the tail for r-> infinity agrees with rungeBCo
    if(mf<0.0) nf1*=-1;
    nf2=nf1*mf;
    for(i=0;i<Idxr.m;i++){
      Fsa[i]*=nf1;
      Gsa[i]*=nf1;
    }
    for(i=Idxr.m;i<=Idxr.x;i++){
      Fsa[i]*=nf2;
      Gsa[i]*=nf2;
    }
  }
  //---------------- normalization of the wavefunction : end ------------------

  return 0;

}

// block 2 : begin
//   This is an independently available part.
//  ================================================================================
//                      proper/screened Poisson equation solver
//                          No global variables are referred.
//  ================================================================================

//------------------------ Poisson equation solver -------------------------
```

```
int Poisson(int imax,double h,double *source,double *phi){
  double r0,r1;
  double I1,I2;
  double rhol;
  int i,j,k,l,a1;
  double intcoef[6]={ 11.0/1440.0, -93.0/1440.0, 802.0/1440.0,
                      802.0/1440.0, -93.0/1440.0,  11.0/1440.0};

  //r=0
  I2=0;
  for(j=0;j<imax-1;j++){
    for(k=0;k<6;k++){
      l=j+k-2;
      r1=l*h;
      a1=abs(l);
      if(a1<imax) rhol=source[a1];
      else rhol=0;
      I2+=intcoef[k]*rhol*r1;
    }
  }
  phi[0]=h*I2;

  //r<0
  for(i=1;i<imax;i++){
    r0=i*h;
    I1=0;
    for(j=0;j<i;j++){
      for(k=0;k<6;k++){
        l=j+k-2;
        r1=l*h;
        a1=abs(l);
        if(a1<imax) rhol=source[a1];
        else rhol=0;
        I1+=intcoef[k]*rhol*r1*r1/r0;
      }
    }
    I2=0;
    for(k=0;k<6;k++){
      for(j=i;j<imax-1;j++){
        l=j+k-2;
        r1=l*h;
        a1=abs(l);
        if(a1<imax) rhol=source[a1];
        else rhol=0;
        I2+=intcoef[k]*rhol*r1;
      }
    }
    phi[i]=h*(I1+I2);
  }
  return 0;
```

```
}

//------------------ screened Poisson equation solver -------------------------
int screenedPoisson(int imax, double h, double m, double *source, double *phi){
  double D;          // the value of Green's function
  double r0,r1;      // variables r and r' of Green's function
  double s,s1,s2,exp1,exp2,sourcel;
  int i,j,k,l,n,al,nexp;
  double intcoef[6]={ 11.0/1440.0, -93.0/1440.0, 802.0/1440.0,
                      802.0/1440.0, -93.0/1440.0,  11.0/1440.0};
#define NEXP 100000
  double expmmr[NEXP];
  int expsft = 3;

  nexp=NEXP;
  exp1=exp(-m*h);
  for(k=0;k<NEXP;k++){
    if(k % 20 == 0) expmmr[k]=exp(-m*h*(k-expsft));
    else expmmr[k]=expmmr[k-1]*exp1;
    if(expmmr[k]<1.0e-32) {
      nexp=k+1;
      break;
    }
  }
  if(expmmr[nexp-1]>1.0e-16) {
    fprintf(stderr,"screenedPoisson: warning: You have to increase NEXP\n");
    fprintf(stderr,"expmmr[%d]=%e\n",NEXP,expmmr[NEXP-1]);
  }
#undef NEXP

  // calculation of phi(r) at r=0
  s2=0;
  for(j=0;j<imax-1;j++){  // integral over [0,(imax-1)*h]
    for(k=0;k<6;k++){
      l=j+k-2;
      r1=l*h;
      n=l+expsft; if(n<nexp) exp1=expmmr[n]; else exp1=0;
      al=abs(l); if(al<imax) sourcel=source[al]; else sourcel=0;
      s2+=intcoef[k]*sourcel*r1*exp1;
    }
  }
  phi[0]=h*s2;

  //  fprintf(stderr,"phi[0]=%.7f\n",phi[0]);

  // calculation of phi(r) for r>0
  for(i=1;i<imax;i++){
    r0=i*h;
    s1=0;
    for(j=0;j<i;j++){  // integral over [j*h,(j+1)*h], (j+1)*h<=r0
```

```
      for(k=0;k<6;k++){
         l=j+k-2;
         r1=l*h;
         n=i-l+expsft; if(n<nexp) exp1=expmmr[n]; else exp1=0;
         n=i+l+expsft; if(n<nexp) exp2=expmmr[n]; else exp2=0;
         al=abs(l); if(al<imax) sourcel=source[al]; else sourcel=0;
         s1+=intcoef[k]*sourcel*r1*(exp1-exp2);
      }
    }
    s2=0;
    for(j=i;j<imax-1;j++){  // integral over [j*h,(j+1)*h], j*h>=r0
      for(k=0;k<6;k++){
         l=j+k-2;
         r1=l*h;
         n=l-i+expsft; if(n<nexp) exp1=expmmr[n]; else exp1=0;
         n=l+i+expsft; if(n<nexp) exp2=expmmr[n]; else exp2=0;
         al=abs(l); if(al<imax) sourcel=source[al]; else sourcel=0;
         s2+=intcoef[k]*sourcel*r1*(exp1-exp2);
      }
    }
    phi[i]=h*(s1+s2)/(2*m*r0);
  }
  return 0;
}
// block 2 : end

// block 3 : start
//   No global variables are referred.
//-----------------------------------------------------------------------------/
int interpolate(double *y, int n){
/*
  double y[n],
  y[0],y[2],y[4],...,y[n-5],y[n-3],y[n-1] : given.
  y[1],y[3],y[5],...,y[n-6],y[n-4],y[n-2] : to be interpolated.
  n must be an odd integer not less than 7.
  y[-i]=y[i] assumed.
  y[i]=0 assumed for i >= n.
*/
  int i;
  double f0=  3.0/256.0;
  double f1=-25.0/256.0;
  double f2= 75.0/128.0;
  double f3= 75.0/128.0;
  double f4=-25.0/256.0;
  double f5=  3.0/256.0;

  if(n%2==0||n<7){
    exit(1);
  }
```

```
  y[1]  =f0*y[4]  +f1*y[2]  +f2*y[0]  +f3*y[2]  +f4*y[4]  +f5*y[6]  ;
  y[3]  =f0*y[2]  +f1*y[0]  +f2*y[2]  +f3*y[4]  +f4*y[6]  +f5*y[8]  ;

  for(i=5;i<=n-6;i+=2){
    y[i]=f0*y[i-5]+f1*y[i-3]+f2*y[i-1]+f3*y[i+1]+f4*y[i+3]+f5*y[i+5];
  }

  // NB) density beyond the maximum radius is approximated with zero.
  //  A better treatment is to construct 5-pt and 4-pt formulae
  // and use them for these points.
  y[n-4]=f0*y[n-9]+f1*y[n-7]+f2*y[n-5]+f3*y[n-3]+f4*y[n-1]          ;
  y[n-2]=f0*y[n-7]+f1*y[n-5]+f2*y[n-3]+f3*y[n-1]                    ;

}
// block 3 : end
```