Dirac-Fock法による原子の基底状態の電子配置

2008年1月30日

福井大学 工学部 物理工学科
16年度入学 1番　秋山大樹、16番　上川暢介

目 次

序章

原子とは 1 個の原子核と何個かの電子とからなる安定な物理系である。Z 個の電子を持つ原子では、電子の電荷の総和は e を電気素量として $-Ze$、原子核の持つ電荷は $+Ze$ である。Z は正の整数で原子番号と呼ばれる。

我々は原子核研究室に所属しているが、その内の 1 人（秋山）が大学院進学後は原子のスペクトルの研究を行うため研究の対象を原子核ではなく原子にした。研究室が原子核の研究室であるため、原子を扱う研究なら将来的には原子核の有限サイズの効果も論じられる理論的枠組みで研究を行いたいと指導教員が志望した（本研究では核の有限サイズの効果は論じない）。ところが、電子を原子核のサイズに押し込めると位置の不確定さが減るので、不確定性関係によって運動量の不確定さが増し、期待値が大きくなる。電子の質量は陽子や中性子と比べると 1/2000 程度と軽いので、速度が速くなり非相対論での扱いが不適切となるため、相対論の効果を取り入れた Dirac 方程式による扱いが必要だとわかった。

このような経緯で原子の電子状態や電子配位を Dirac 方程式および、多電子の場合は自己無撞着場（Dirac-Fock 法と呼ばれることが多い）の方法で研究するというテーマ設定となった。第 1 章は水素様原子について記した。第 2 章は第 1 章に基づいて多電子電子について記した。

なお、本論文の理論の説明の部分の多くは関連図書 [2] の記述をもとにして書いたものであるが、筆者が理解に手間取った部分を中心に独自の説明を付加することで学部学生の読者に対し、よりわかりやすいものに変えたつもりである。今後、この分野の研究を始める初学者の役に立つことがあれば幸いである。

執筆担当者
第 1 章　上川
第 2 章　秋山

第１章　水素様原子

1.1　非相対論的取り扱い

水素様原子は核と電子の２粒子系である。系のエネルギーは、２粒子の運動エネルギーと２粒子間のクーロン引力のポテンシャルの和であるので、その系の Schrödinger 方程式は

$$\left\{-\frac{\hbar^2}{2M}\vec{\nabla}_{\rm n}^2-\frac{\hbar^2}{2m_{\rm e}}\vec{\nabla}_{\rm e}^2-\frac{Ze^2}{4\pi\epsilon_0 r}\right\}\psi(\vec{r_{\rm n}},\vec{r_{\rm e}})=E\psi(\vec{r_{\rm n}},\vec{r_{\rm e}}) \tag{1.1}$$

となる。ただし $\vec{\nabla_{\rm n}}^2$、$\vec{\nabla_{\rm e}}^2$ はそれぞれ核と電子の位置ベクトル $\vec{r_{\rm n}}$、$\vec{r_{\rm e}}$ の成分に関する微分演算子である。即ち

$$\vec{r_{\rm n}}=(x_{\rm n},y_{\rm n},z_{\rm n}) \tag{1.2}$$

$$\vec{r_{\rm e}}=(x_{\rm e},y_{\rm e},z_{\rm e}) \tag{1.3}$$

とすると

$$\vec{\nabla}_{\rm n}=(\frac{\partial}{\partial x_{\rm n}},\frac{\partial}{\partial y_{\rm n}},\frac{\partial}{\partial z_{\rm n}}) \tag{1.4}$$

$$\vec{\nabla}_{\rm e}=(\frac{\partial}{\partial x_{\rm e}},\frac{\partial}{\partial y_{\rm e}},\frac{\partial}{\partial z_{\rm e}}) \tag{1.5}$$

と定義する。M は核の質量、$m_{\rm e}$ は電子の質量、r は核と電子の位置ベクトルの差の絶対値とする。説明を簡略化するためスピン等の内部自由度は省いてある。ここで $\vec{r_{\rm n}}$、$\vec{r_{\rm e}}$ のかわりに系の重心位置ベクトル $\vec{R}$ と相対ベクトル $\vec{r}$ を

$$\vec{R}=\frac{M\vec{r_n}+m_e\vec{r_e}}{M+m_e},\quad \vec{r}=\vec{r_e}-\vec{r_n} \tag{1.6}$$

で導入すると、運動エネルギーの部分が

$$-\frac{\hbar^2}{2M}\vec{\nabla}_{\rm n}^2-\frac{\hbar^2}{2m_{\rm e}}\vec{\nabla}_{\rm e}^2=-\frac{\hbar^2}{2M_{\rm t}}\vec{\nabla}_{\rm R}^2-\frac{\hbar^2}{2\mu}\vec{\nabla}^2, \tag{1.7}$$

$$M_{\rm t}=M+m_{\rm e},\quad \mu=\frac{m_{\rm e}M}{m_{\rm e}+M} \tag{1.8}$$

のように重心運動のエネルギーと相対運動のエネルギーに分離できる。ただし、$\vec{\nabla_{\rm R}}$、$\vec{\nabla}$ はそれぞれ $\vec{R}$、$\vec{r}$ の成分に関するナブラ演算子である。このようにして (1.1) は２つの独立な方程式に分離される。

$$-\frac{\hbar^2}{2M_{\rm t}}\vec{\nabla}_{\rm R}^2\psi_{\rm g}(\vec{R})=E_{\rm g}\psi_{\rm g}(\vec{R}), \tag{1.9}$$

$$\left\{-\frac{\hbar^2}{2\mu}\vec{\nabla}^2-\frac{Ze^2}{4\pi\epsilon_0 r}\right\}\psi_{\rm s}(\vec{r})=E_{\rm s}\psi_{\rm s}(\vec{r}), \tag{1.10}$$

$$\psi = \psi_{\rm g}(\vec{R})\psi_{\rm s}(\vec{r}), \quad E_{\rm g} + E_{\rm s} = E. \tag{1.11}$$

最初の式が重心運動 (添字 g は center of gravity を意味する) の方程式で第 2 式が相対運動 (添字 s は soutai を意味する) の方程式となっている。重心運動の解は平面波であり、$\vec{K}$ を波数ベクトルとして

$$\psi_{\rm g} = \frac{1}{\sqrt{L^3}} e^{i\vec{K}\cdot\vec{R}}, \quad E_{\rm g} = \frac{\hbar^2\vec{K}^2}{2M_{\rm t}} \tag{1.12}$$

と表される。ただし、波動関数の規格化は 1 辺の長さが L の立方体内の存在確率が 1 になるとして行った。基底状態（最もエネルギーの低い状態）は $\vec{K} = 0$ であり、そのときのエネルギーは $E_{\rm g} = 0$ である。内部運動を記述する第 2 の方程式 (2.8) の固有値は解析的に求めることができ、結果は

$$E_n = -\frac{\mu e^4 Z^2}{2n^2(4\pi\epsilon_0)^2\hbar^2} = -\frac{Z^2}{n^2}{\rm R_y} \tag{1.13}$$

となる。これが水素様原子のエネルギーである。即ち $E = E_{\rm s} + E_{\rm g} = E_{\rm s} = E_{\rm n}$ である。n は主量子数と呼ばれる量子数で正の整数値をとる。$\rm R_y$ はリュードベリ定数と呼ばれ、

$${\rm R_y} = \frac{\rm m_e e^4}{2(4\pi\epsilon_0\hbar)^2} = 13.60570{\rm eV} \tag{1.14}$$

である。ここで個々のエネルギー準位にどのような角運動量の状態が属するかを説明する。l は方位量子数、m は磁気量子数を表し、それぞれ $0 \le l \le n-1$、$-l \le m \le l$ という範囲の整数値をとる。また l は数字のかわりにアルファベットの小文字を用いて $l = 0$ は s、$l = 1$ は p、$l = 2$ は d、$l = 3$ は f、$l = 4$ は g、$l = 5$ は h、$\cdots$ とも表される。

n の数字に続けて l を表すアルファベットを書いて軌道を表すことが多い。例えば $n = 1$ には 1s、$n = 2$ には 2s、2p、$n = 3$ には 3s、3p、3d という軌道がある。

電子のスピンの自由度を考慮すると状態の個数は 2 倍となる。主量子数 n の殻には

$$\sum_{l=0}^{n-1} 2 \times (2 \times l + 1) = 2n^2 \tag{1.15}$$

個の電子が入りうる。また、主量子数が n 以下の殻に入る電子の最大個数は

$$\sum_{n'=1}^{n} 2n'^2 = \frac{1}{3}n(n+1)(2n+1) \tag{1.16}$$

個である。

1.2 相対論的取り扱い

前節で論じた Schrödinger 方程式は相対論的効果を無視している。Schrödinger 方程式を Dirac 方程式に置き換えれば、相対論的効果を完全に取り入れることになる。

Dirac 方程式は

$$\left[c\vec{\alpha}\cdot\vec{p} + m_{\rm e}c^2\beta\right]\psi = i\hbar\frac{\partial\psi}{\partial t} \tag{1.17}$$

と表される。ただし、ψ は 4 つの成分を持ち、

$$\vec{\alpha} = \begin{bmatrix} 0 & \vec{\sigma} \\ \vec{\sigma} & 0 \end{bmatrix}, \quad \beta = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix}, \tag{1.18}$$

$$\sigma_{\rm x} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_{\rm y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_{\rm z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{1.19}$$

である。ちなみに $\sigma_{\rm x},\sigma_{\rm y},\sigma_{\rm z}$ は Pauli 行列と呼ばれる。I は 2 行 2 列の単位行列である。方程式 (1.17) の定常解を求めるために

$$\psi = \chi(\vec{r})e^{-\frac{iEt}{\hbar}} \tag{1.20}$$

とおくと、時間を含まない方程式

$$E\chi(\vec{r}) = \left[c\vec{\alpha}\cdot\vec{p} + m_{\rm e}c^2\beta\right]\chi(\vec{r}) \tag{1.21}$$

が得られる。いま χ の 4 成分を

$$\chi(\vec{r}) = \begin{bmatrix} \mu(\vec{r}) \\ \omega(\vec{r}) \end{bmatrix}, \quad \mu(\vec{r}) = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \omega(\vec{r}) = \begin{bmatrix} \omega_3 \\ \omega_4 \end{bmatrix}, \tag{1.22}$$

のように 2 つの 2 成分関数 μ と ω に分けると、(1.21) は

$$\begin{cases} E\mu = & c\vec{p}\cdot\vec{\sigma}\omega & + & m_{\rm e}c^2\mu \\ E\omega = & c\vec{p}\cdot\vec{\sigma}\mu & - & m_{\rm e}c^2\omega \end{cases} \tag{1.23}$$

となる。運動量 $\vec{p}$ が z 方向を向いているように座標系を選ぶと、規格化因子を除いて次のような 4 つの独立な解が得られる。ただし、$E_{\rm p} = \sqrt{(m_{\rm e}c^2)^2 + \vec{p}^2c^2}$ である。

$$\chi_1 \propto \begin{bmatrix} 1 \\ 0 \\ \frac{E_{\rm p}-m_{\rm e}c^2}{cp} \\ 0 \end{bmatrix}, \chi_2 \propto \begin{bmatrix} 0 \\ 1 \\ 0 \\ -\frac{E_{\rm p}-m_{\rm e}c^2}{cp} \end{bmatrix}, \chi_3 \propto \begin{bmatrix} -\frac{E_{\rm p}-m_{\rm e}c^2}{cp} \\ 0 \\ 1 \\ 0 \end{bmatrix}, \chi_4 \propto \begin{bmatrix} 0 \\ \frac{E_{\rm p}-m_{\rm e}c^2}{cp} \\ 0 \\ 1 \end{bmatrix}. \tag{1.24}$$

なお、$E_{\rm p} = -\sqrt{(m_{\rm e}c^2)^2 + \vec{p}^2c^2}$ に対応する解も存在するが、Dirac の空孔理論によれば $E_{\rm p} < 0$ の解は真空状態において既に全て占拠されていると考える。したがって 1 電子のとる状態としては $E_{\rm p} > 0$ の状態だけを考えればよい。$E_{\rm P}$ の表式を $p^2/(m_{\rm e}^2c^2)$ のべき級数の形に展開すると

$$\begin{aligned} E_{\rm p} &= \sqrt{(m_{\rm e}c^2)^2 + p^2c^2} && (1.25)\\ &= m_{\rm e}c^2\sqrt{1+\frac{p^2}{m_{\rm e}^2c^2}} && (1.26)\\ &= m_{\rm e}c^2\left\{1 + \frac{p^2}{2m_{\rm e}^2c^2} - \frac{1}{8}\left(\frac{p^2}{m_{\rm e}^2c^2}\right)^2 + \cdots\right\} && (1.27)\\ &= m_{\rm e}c^2 + \frac{p^2}{2m_{\rm e}} - \frac{p^2}{8m_e}\left(\frac{p}{m_ec}\right)^2 + \cdots && (1.28) \end{aligned}$$

となる。右辺の第 1 項 $m_{\rm e}c^2$ は静止質量のエネルギーであり、第 2 項が非相対論的な運動エネルギーと同一のものである。水素原子の基底状態では $p^2/2m_{\rm e} = 13.6{\rm eV}$ であるのに対し $m_{\rm e}c^2 = 5.0\times10^5{\rm eV}$ であり、その大きさに約 4×10^4 倍もの違いがある。即ち、非相対論的計算で 13.6eV という 3 桁の精度の値を得るためには、相対論的計算では $(510998.9 + 13.6){\rm eV}$ という 7 桁の精度の値を求める必要があることになる。ここに数値計算の観点からみた Dirac 方程式の難しさが現れている。

次に外場がある時の Dirac 方程式について考える。外場がある時の方程式は、自由粒子のときの運動量 $\vec{p}$、エネルギー E をそれぞれ

$$\vec{p} \to \vec{p} - q\vec{A}, \quad E \to E - q\phi \tag{1.29}$$

と置き換えれば得られることがわかっている。q は粒子の持つ電荷、$\vec{A},\phi$ は電磁場のベクトルおよびスカラーポテンシャルである。(1.17) にこの置き換えを行うと、電子に対する電荷 $q = -e$ を代入し、

$$\left[c\vec{\alpha}\cdot(\vec{p}+e\vec{A}) - e\phi + \beta m_{\rm e}c^2\right]\psi = i\hbar\frac{\partial\psi}{\partial t} \tag{1.30}$$

が得られる。

以下は、$E > 0$, したがって $\psi = \begin{bmatrix} \mu \\ \omega \end{bmatrix} exp(\frac{-iEt}{\hbar})$ の μ が大きく、ω が小さい成分である時について考える。自由粒子の (1.23) に相当する連立方程式を (1.30) から導き、$E = m_{\mathrm{e}}c^2 + E'$ とおくと

$$\begin{cases} E'\mu = c(\vec{p} + e\vec{A}) \cdot \vec{\sigma}\omega - e\phi\mu \\ (E' + 2m_{\mathrm{e}}c^2)\omega = c(\vec{p} + e\vec{A}) \cdot \vec{\sigma}\mu - e\phi\omega \end{cases} \tag{1.31}$$

となる。ここで $E', e\phi$ とも絶対値が十分小さいとすると、第 2 式から近似的に

$$\omega(\vec{r}) = \frac{1}{2m_{\mathrm{e}}c}(\vec{p} + e\vec{A}) \cdot \vec{\sigma}\mu(\vec{r}) \tag{1.32}$$

が得られる。これを第 1 式に代入し、μ だけの式にすると

$$E'\mu(\vec{r}) = \frac{1}{2m_{\mathrm{e}}} \left[(\vec{p} + e\vec{A}) \cdot \vec{\sigma}\right]^2 \mu(\vec{r}) - e\phi\mu(\vec{r}) \tag{1.33}$$

となる。Pauli のスピン行列の間には

$$\begin{cases} \sigma_{\mathrm{x}}^2 = \sigma_{\mathrm{y}}^2 = \sigma_{\mathrm{z}}^2 = 1 \\ \sigma_{\mathrm{x}}\sigma_{\mathrm{y}} = -\sigma_{\mathrm{y}}\sigma_{\mathrm{x}} = i\sigma_{\mathrm{z}} \\ \sigma_{\mathrm{y}}\sigma_{\mathrm{z}} = -\sigma_{\mathrm{z}}\sigma_{\mathrm{y}} = i\sigma_{\mathrm{x}} \\ \sigma_{\mathrm{z}}\sigma_{\mathrm{x}} = -\sigma_{\mathrm{x}}\sigma_{\mathrm{z}} = i\sigma_{\mathrm{y}} \end{cases} \tag{1.34}$$

の関係があることから、スピンに直接関係しない任意の 2 つのベクトル量 $\vec{A}, \vec{B}$ に対して

$$(\vec{A} \cdot \vec{\sigma})(\vec{B} \cdot \sigma) = (\vec{A} \cdot \vec{B}) + i(\vec{A} \times \vec{B}) \cdot \sigma \tag{1.35}$$

が成り立つ。(1.33) にこの公式をあてはめ、また $\vec{p}$ を微分演算子 $-i\hbar\vec{\nabla}$ に書き直せば

$$E'\mu = \left[\frac{1}{2m_e}(-i\hbar\vec{\nabla} + e\vec{A})^2 + \frac{e\hbar}{2m_e}(\vec{\sigma} \cdot \vec{B}) - e\phi\right]\mu \tag{1.36}$$

となる。ただし、$\vec{B} = \vec{\nabla} \times \vec{A}$ は磁束密度を表す。v/c の 2 次以上の項を無視して、水素様原子では $\vec{A} = 0$ であり、$\phi = \frac{Ze}{4\pi\epsilon_0 r}$ なので、時間を含まない Dirac 方程式 (1.21) は

$$E\chi = \left[c\vec{\alpha} \cdot \vec{p} + V(r) + m_e c^2\right]\chi, \quad V(r) = -\frac{Ze^2}{4\pi\epsilon_0 r} \tag{1.37}$$

となる。また、v/c の 1 次までの近似による非相対論的近似方程式 (1.36) は

$$E'\mu = (-\frac{\hbar^2}{2m_{\mathrm{e}}}\vec{\nabla}^2 + V(r))\mu \tag{1.38}$$

となり、これは Schrödinger 方程式と同一の方程式である。Dirac 方程式と Schrödinger 方程式の違いが現れるには v/c の 2 次以上の項まで考慮する必要がある。結果として得られるハミルトニアンは

$$H = -\frac{\hbar^2}{2m_{\mathrm{e}}}\vec{\nabla}^2 + V(r) - \frac{1}{8m_{\mathrm{e}}^3c^2}\vec{p}^4 + \frac{\hbar^2}{2m_{\mathrm{e}}^2c^2}\frac{1}{r}\frac{dV}{dr}\vec{l} \cdot \vec{s} - \frac{\hbar^2}{4m_{\mathrm{e}}^2c^2}\vec{\nabla}V \cdot \vec{\nabla} \tag{1.39}$$

となる。ただし、ここで $\vec{l} = \vec{r} \times \vec{p}$、$\vec{s} = \frac{1}{2}\vec{\sigma}$ と書いた。右辺第 3 項は運動エネルギーを p の 2 次式で近似することへの補正、第 4 項はスピン軌道間相互作用という意味をもつ。第 4 項の存在により、原子中の電子のエネルギー準位は主量子数 n だけでは決まらず、後述のとおり全角運動量 j にも依存するようになる。ここで

$$\vec{j} = \vec{l} + \vec{s} \tag{1.40}$$

であり、

$$j = l \pm \frac{1}{2} \tag{1.41}$$

である。

$$\vec{j}^2 = (\vec{l} + \vec{s})^2 = \vec{l}^2 + 2(\vec{l} \cdot \vec{s}) + \vec{s}^2, \tag{1.42}$$

$$\vec{j}^2 = j(j+1), \quad \vec{l}^2 = l(l+1), \quad \vec{s}^2 = s(s+\frac{1}{2}) = \frac{3}{4} \tag{1.43}$$

より

$$\vec{l} \cdot \vec{s} = \frac{1}{2}\left\{ j(j+1) - l(l+1) - \frac{3}{4} \right\} \tag{1.44}$$

となるので、(1.39) の近似のレベルでは j の値に加えて l が $j + \frac{1}{2}$ か $j - \frac{1}{2}$ かによってエネルギーが変化するのだが、Dirac 方程式を近似なく扱えば後述のとおり l への依存性は消失する。

(1.37) の固有値は解析的に求めることができて、$k = j + 1$ として

$$E_{n,j} = \frac{m_e c^2}{\sqrt{1 + \left(\frac{\alpha Z}{n - k + \sqrt{k^2 - (\alpha Z)^2}} \right)^2}} \tag{1.45}$$

となる。ただし

$$\alpha = \frac{e^2}{4\pi\epsilon_0 \hbar c} \simeq \frac{1}{137.03599} \tag{1.46}$$

で α は微細構造定数と呼ばれている。Schrödinger 方程式の固有値にそろえて、電子が核から無限に引き離されて静止しているときのエネルギーを 0 にするには、(1.45) の $E_{n,j}$ が含んでいる電子の静止エネルギー $m_e c^2$ を引いて

$$E'_{n,j} = E_{n,j} - m_e c^2 \tag{1.47}$$

とする必要がある。α の値が小さいので、Z が大きくなければ αZ も 1 に比べて小さいことを利用してテーラー展開すると

$$E'_{n,j} = -\frac{Z^2}{n^2}\mathrm{R_y} - \frac{\alpha^2 Z^4}{4n^4}\left(\frac{4n}{k} - 3 \right)\mathrm{R_y} + O(\alpha^6 Z^6) \tag{1.48}$$

となる。(1.48) の右辺の第 1 項が (1.13) の右辺と等しいので第 2 項が相対論による主要な補正となっている。k が小さいほど、即ち j が小さいほどエネルギー準位は大きく下がることがわかる。(1.45) は主量量子数 n の他に j によってもエネルギーがかわる。このように同じ n、異なった j でエネルギー準位がいくつかに分離することを微細構造と呼ぶ。下に $Z = 1$ と $Z = 100$ でともに主量子数 $n = 2$ の状態のエネルギー準位の図を示す。

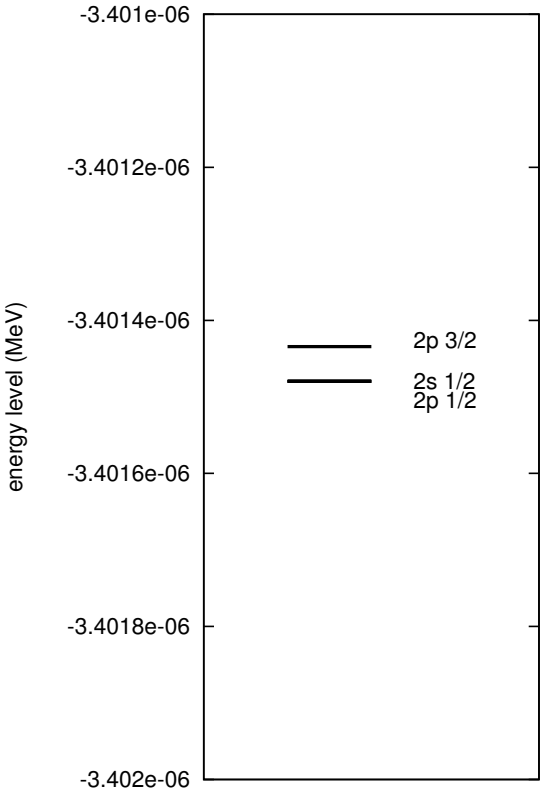


図 1.1: Z=1 の場合の $n = 2$ 殻の準位

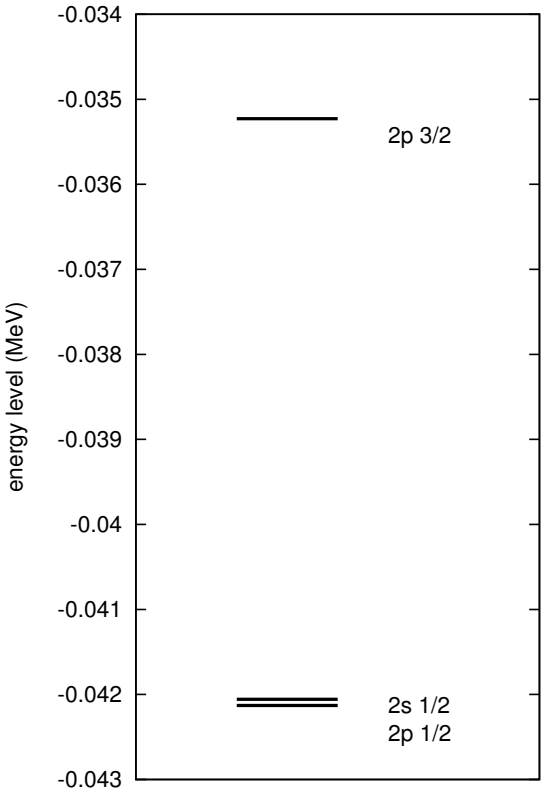


図 1.2: Z=100 の場合の $n = 2$ 殻の準位

図 1.1 は $Z = 1$(水素原子核) のまわりを 1 個の電子が運動する場合のエネルギー固有値である。エネルギーの大きさ (約 3.4eV) に対して、相対論的効果による $n = 2$ 殻の分裂幅は約 75000 分の 1(約 4.5×10^{-5}eV) しかない。一方 $Z = 100$ 核のまわりを 1 個の電子が運動する場合は、分裂幅 (約 6.8×10^{3}eV) はエネルギー大きさ (約 0.04MeV) の 6 分の 1 程度にまで大きくなっている。これは (1.48) 式の右辺第 1 項が Z^2 に比例するのに対し、第 2 項が Z^4 に比例するため、Z が増大すると第 2 項の第 1 項に対する大きさの比が Z^2 に比例して増大するためである。ただし、$\alpha \fallingdotseq 10^{-2}$ であるから、$Z = 1$ のときは第 2 項は第 1 項の $\alpha \fallingdotseq 10^{-4}$ 倍程度の大きさしかない。しかし、$Z = 100$ の場合は $\alpha^2 Z^2 \fallingdotseq 1$ となるため第 2 項の効果が顕著に現れるようになるのである。なお、図 1.2 においては $2s_{1/2}$ と $2p_{1/2}$ がわずかに (71eV) 分裂している。これは描いたレベルが (1.45) 式の値ではなく、次章で述べる Dirac 方程式の数値解法において (1.52) 式のとおり原子核の有限の大きさを考慮して計算した結果を描いたためである。原子核のサイズがゼロの極限で縮退する s 軌道と p 軌道は原子核のサイズが有限になると、原点 ($r = 0$) で波動関数がゼロとならない s 波のみが影響を受けてエネルギーが上昇するため、縮退が破れるのである。

1.3 相対論での動径波動関数の数値解法

第 2 章で扱う多電子系では電位が $1/r$ に比例するような単純な数式では表せないので解析的に解くことができない。よって動径波動関数の数値解法によってエネルギーを求める。

原子中の電子の扱いの難しい点として局所的な振舞のエネルギースケールが原子核の近傍と核から遠く離れた地点では何桁も違うことがあげられる。計算精度を上げるためには $r \to 0$ 付近でグリッドを密にしなければならないので、r を別の変数 ξ に変数変換することによってこの問題を解決する。
Dirac 方程式の動径部分は

$$\begin{cases} \frac{d}{dr}G_{(r)} = -\frac{\kappa}{r}G_{(r)} + (E - V_{(r)} + mc^2)F_{(r)} \\ \frac{d}{dr}F_{(r)} = \frac{\kappa}{r}F_{(r)} - (E - V_{(r)} - mc^2)G_{(r)} \end{cases} \tag{1.49}$$

である。なおこの節では自然単位系 ($\hbar = c = 1$) を用いる。また、

$$\varpi = 2(l - j), \quad \kappa = \varpi(j + \frac{1}{2}) \tag{1.50}$$

は波動関数の角度部分を指定するための量子数であり、全角運動量 j とは

$$j = \begin{cases} l - \frac{1}{2} & (l \geq 1), \quad \varpi = 1, \quad \kappa = l \\ l + \frac{1}{2} & (l \geq 0), \quad \varpi = -1, \quad \kappa = -(l + 1) \end{cases} \tag{1.51}$$

という関係にある。ポテンシャルエネルギーは原子核のサイズが有限であることを考慮して

$$V(r) = \begin{cases} -\frac{Ze^2}{R}\left\{\frac{3}{2} - (\frac{r}{R})^2\right\} & (r \leq R) \\ -\frac{Ze^2}{r} & (r \geq R) \end{cases} \tag{1.52}$$

の形を採用する。ここで R は核の半径で $1.2 \times$ (質量数)$^{\frac{1}{3}}$ fm である。ここで

$$f_{11} = -\frac{\kappa}{r}, \quad f_{12} = \frac{\kappa}{r} \tag{1.53}$$

$$f_{21} = (E - V_{(r)} + mc^2), \quad f_{22} = -(E - V_{(r)} - mc^2) \tag{1.54}$$

とおくと (1.49) は

$$\frac{d}{dr}\begin{pmatrix} G \\ F \end{pmatrix} = \begin{pmatrix} f_{11(r)} & f_{12(r)} \\ f_{21(r)} & f_{22(r)} \end{pmatrix}\begin{pmatrix} G \\ F \end{pmatrix} \tag{1.55}$$

となる。ここで r から ξ に変数変換する。即ち

$$r = r(\xi) \tag{1.56}$$

とすると

$$\frac{d}{d\xi}\begin{pmatrix} G \\ F \end{pmatrix} = \frac{dr}{d\xi}\frac{d}{dr}\begin{pmatrix} G \\ F \end{pmatrix} \tag{1.57}$$

なので (1.55) は

$$\frac{d}{d\xi}\begin{pmatrix} G \\ F \end{pmatrix} = \frac{dr}{d\xi}\begin{pmatrix} f_{11(r)} & f_{12(r)} \\ f_{21(r)} & f_{22(r)} \end{pmatrix}\begin{pmatrix} G \\ F \end{pmatrix} \tag{1.58}$$

になる。$\begin{pmatrix} f_{11(r)} & f_{12(r)} \\ f_{21(r)} & f_{22(r)} \end{pmatrix}$ は $1/r$ に比例する項を含むので $\frac{dr}{d\xi}$ を r に比例させれば $1/r$ の項を打ち消すことができる。即ち、

$$\frac{dr}{d\xi} = \frac{r}{a} \tag{1.59}$$

(a は比例定数) とすればよい。この微分方程式の解は

$$\xi = a \ln r + \xi_0 \tag{1.60}$$

である。即ち ξ の関数としての r は

$$r = e^{\frac{\xi - \xi_0}{a}} \tag{1.61}$$

とすればよい。この結果、動径グリッドは

$$\xi_i = \xi_0 + hi, \quad 0 \leq i \leq N, \tag{1.62}$$

$$r_i = (e^{\frac{h}{a}})^i \tag{1.63}$$

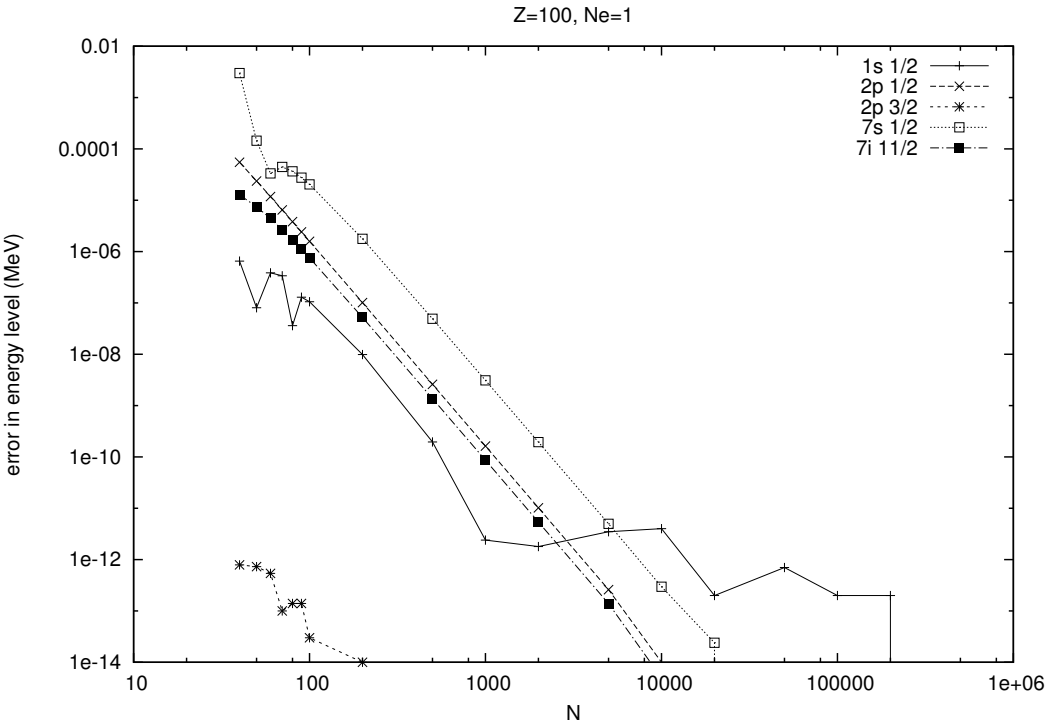
となる。(1.62) は等間隔となり (1.63) は等比数列になっている。しかし、

$$\int_0^r dr \to \int_{-\infty}^{\xi_0} d\xi + \int_{\xi_0}^{\infty} d\xi \tag{1.64}$$

において、矢印の右側の第 1 項の積分の扱い方を別途考える必要があるのが面倒である。そこで、以下のように $r = 0$ での正則な変換に置き換えた。

$$\begin{cases} \frac{\xi}{a} = \log \left| \frac{r}{a} + \sqrt{(\frac{r}{a})^2 + 1} \right| \\ \frac{r}{a} = \sinh \left(\frac{\xi}{a} \right) \end{cases} \tag{1.65}$$

1.3.1 計算精度の確認



上のグラフは、動径波動関数の数値解法によって求めたエネルギー固有値の誤差を描いたものである。$N = 10^6$ での計算結果を誤差なしと見なしている。縦軸がエネルギーの誤差で、横軸はグリッド点 N 数となっている。

$Z = 100$ の場合、解析的に得られたエネルギーは $1s_{1/2}$ 状態で約 1.6×10^{-1}(MeV)、$7s_{1/2}$ 状態で約 3.0×10^{-3}(Mev) である。グラフによると、グリッド数を $N = 10000$ にとった時に誤差が約 10^{-12}(MeV) となっている。これは核の有限サイズの効果 (71eV) より 8 桁も高い精度である。また、別の言い方をすれば相対誤差 $10^{-12}/1.6 \times 10^{-1} \doteq 10^{-11}$ を達成できていることになるが、これは計算に用いた倍精度実数が 10^{-15} 程度の精度であることを考えると計算可能な最高精度に近い精度を達成していると言うことになる。$N = 10000$ という非常に少ないグリッド点数でこれだけ高い精度が得られたのは、等比数列のグリッド点を採用したことによるのである。

第2章　多電子原子

2.1　多電子系の電子配位

2.1.1　元素の電子配位

電子配位とは、各原子の持つ電子をどの軌道に何個入れるかの配分のことである。
表 2.1 に実験により決定された電子配置を原子番号 $18 \leq Z \leq 37$ の原子について示す。
これらは文献 [1] からの引用である。

表 2.1: 元素の電子配置

エネルギー準位	1s	2s	2p	3s	3p	3d	4s	4p	4d	4f	5s	5p	5d
18Ar	2	2	6	2	6								
19K	2	2	6	2	6		1						
20Ca	2	2	6	2	6		2						
21Sc	2	2	6	2	6	1	2						
22Ti	2	2	6	2	6	2	2						
23V	2	2	6	2	6	3	2						
24Cr	2	2	6	2	6	5	1						
25Mn	2	2	6	2	6	5	2						
26Fe	2	2	6	2	6	6	2						
27Co	2	2	6	2	6	7	2						
28Ni	2	2	6	2	6	8	2						
29Cu	2	2	6	2	6	10	1						
30Zn	2	2	6	2	6	10	2						
31Ga	2	2	6	2	6	10	2	1					
32Ge	2	2	6	2	6	10	2	2					
33As	2	2	6	2	6	10	2	3					
34Se	2	2	6	2	6	10	2	4					
35Br	2	2	6	2	6	10	2	5					
36Kr	2	2	6	2	6	10	2	6					
37Rb	2	2	6	2	6	10	2	6			1		

水素様原子のエネルギー準位は、同じ主量子数 n を持ち、異なる方位量子数 l の状態は縮退している。しかし、多電子原子の場合は後述のようにこの縮退が破られ、n だけでなく l によってもエネルギー準位が変化する。

パウリの原理によって、一つの軌道にはスピンの向きが反対になった 2 つの電子しか入ることができない。n と l が決まっても、磁気量子数 m の異なる $(2 \times l + 1)$ 個の軌道が存在するので、一つの決まった n, l の軌道には全部で $2 \times (2 \times l + 1)$ 個の電子が入ることができる。

n が小さいうちは、n が異なる状態間のエネルギー差に比べて、同じ n で l が異なるときのエネルギー

差の方が小さい。しかし、n が大きくなると l による差が大きくなる。そのような理由で、3s,3p が閉殻になった後、3d ではなく 4s に電子が入っている。

4s に電子が詰まった後 3d に入る区間では、4s の電子が 3d に入れ替わっている所もあるが、これは他の電子からの影響なども考慮して全体として安定な状態をとるように配位が選ばれるからである。

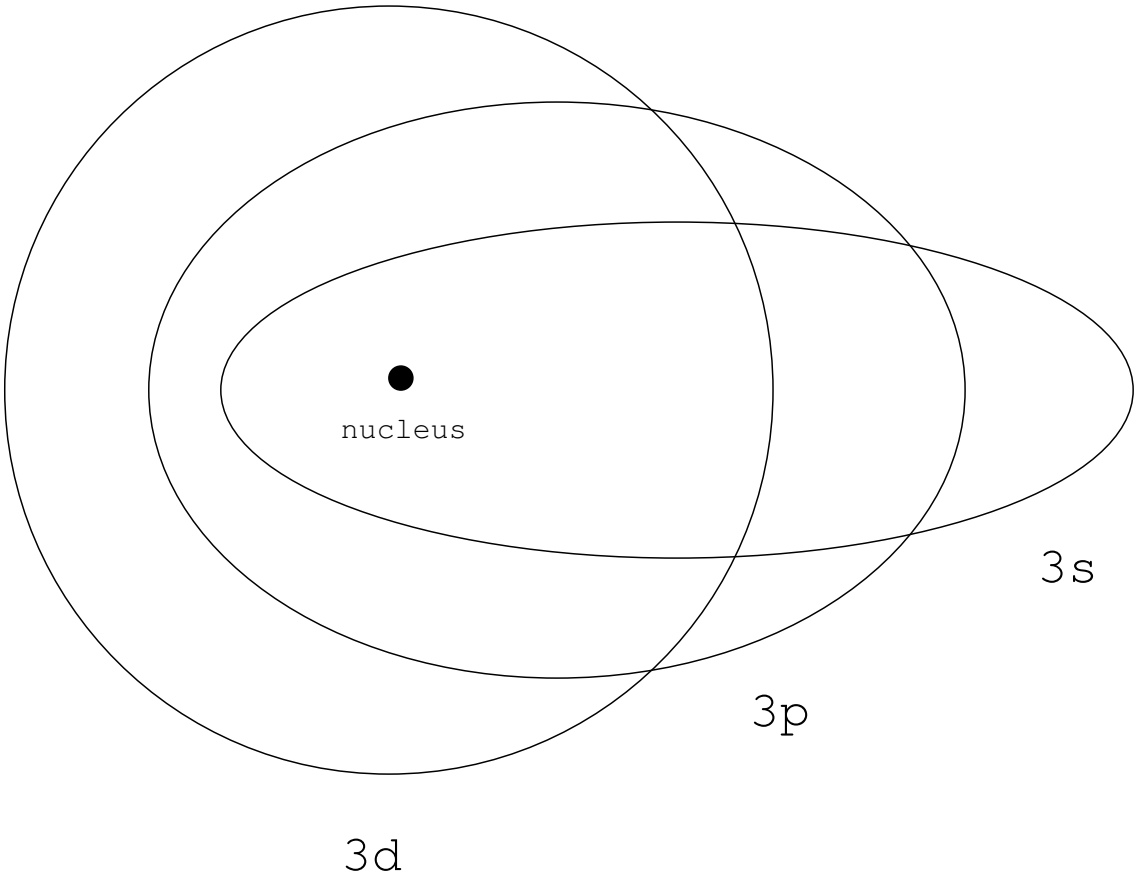


図 2.1: 同じ殻に属する副殻のイメージ図 (主量子数 n=3 の場合)

水素様原子の場合は 3s,3p,3d のエネルギーは同じだが、多電子原子では、角運動量によって軌道は変わってくる。3d 軌道は、円軌道になっているため、内側の電子による核電荷の遮蔽によって、核から受けるクーロン引力が減少するためエネルギー準位が上昇する。一方、3s 軌道は、内側まで入り込み遮蔽の影響を受けにくいのでエネルギーは 3d 軌道の場合ほど高くはならない。

このため 3s,3p,3d の順にエネルギーが高くなる。同様に $n = 4$ に属する 4s は 4p,4d,4f よりエネルギーが低くなる。その結果、3d 軌道より 4s 軌道の方がエネルギーが低くなり、4s,3d の順に電子が詰まっていく。

2.2 イオン化エネルギー

イオン化エネルギーとは、原子から電子を取り除くのに必要なエネルギーのことである。
数式による定義は以下の通りである。

$$I_{\rm e}(Z, N_{\rm e}) = E(Z, N_{\rm e} - 1) - E(Z, N_{\rm e}) \ . \tag{2.1}$$

上式で $I_{\rm e}(Z, N_{\rm e})$ および $E(Z, N_{\rm e})$ は原子番号 Z、電子数 $N_{\rm e}$(中性原子なら $N_{\rm e} = Z$) の原子の基底状態のイオン化エネルギーおよびエネルギー (束縛エネルギーの -1 倍) である。$E(Z, N_{\rm e} - 1)$ は、電子が一個少ない系の基底状態のエネルギーである。図 2.2 に文献 [1] からとった実験的に測定された中性原子のイオン化エネルギーを原子番号 $1 \leq Z \leq 36$ の原子について示した。

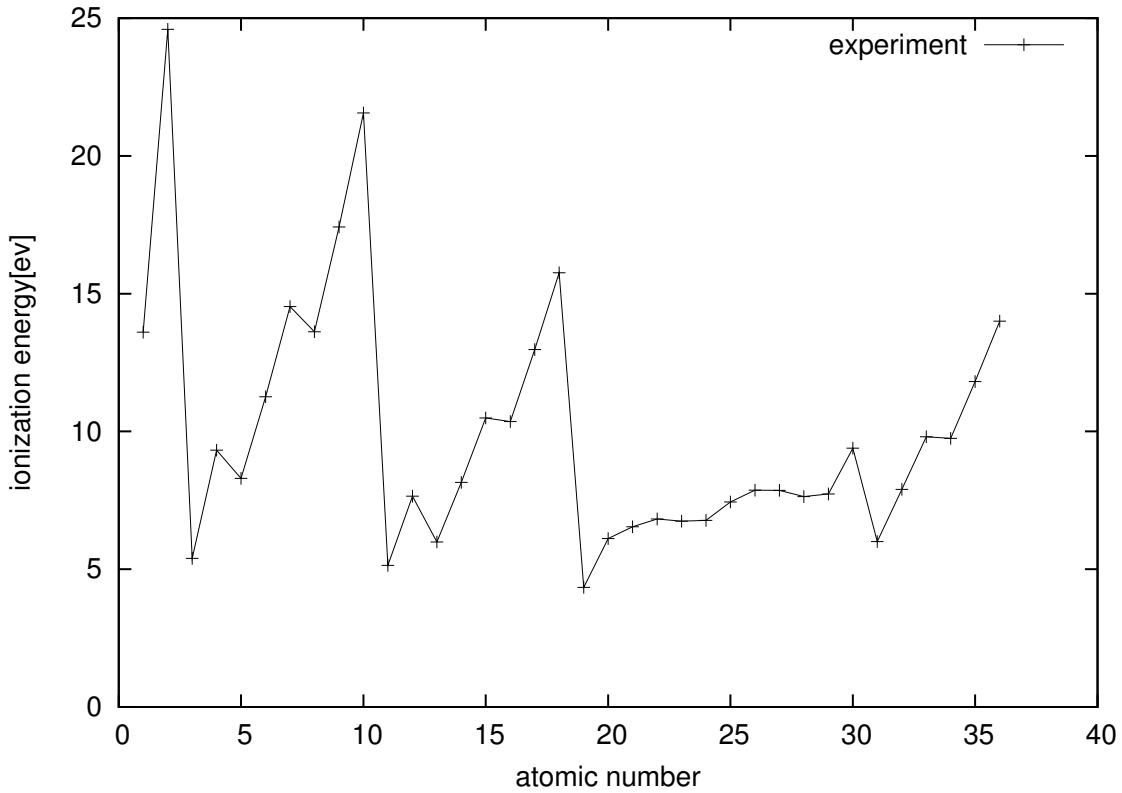


図 2.2: イオン化エネルギー

希ガスは、閉殻なので安定しておりイオン化エネルギーが大きい。閉殻の外に 1 つだけ電子を持っている Li,Na,K などのアルカリ金属では、その電子の束縛が弱いのでイオン化エネルギーが小さい。
表 2.1 を見ると原子番号 $21 \leq Z \leq 29$ の間は、4s に 1、2 個の電子が入った状態で 3d に充填していく区間なので、化学的に似た性質になっており、図 2.2 のようにイオン化エネルギーも $6 \sim 8$eV の範囲に収まっている。($39 \leq Z \leq 47$ の区間も同様)

2.3 多電子系のハミルトニアン

ここでは、説明を分かり易くするためによく知られた非相対論的な Schrödinger 方程式に基づいて説明していく。ただし実際の数値計算は相対論的な Dirac 方程式に基づいて行う。まず最初に N 電子系のハミルトニアンは、

$$H = \sum_{i=1}^{N} \left[-\frac{\hbar^2}{2m} {\vec{\nabla}_i}^2 - \frac{Ze^2}{4\pi\epsilon_0 r_i} \right] + \sum_{i<j}^{N} \frac{e^2}{4\pi\epsilon_0 r_{ij}} \tag{2.2}$$

で表される。右辺の第一項が運動エネルギー、第二項が原子核と電子のクーロン引力エネルギー、第三項が電子同士のクーロン斥力部分となっている。第三項で $i < j$ となっているのは、異なる番号の組み合わせについての和を表している。
時間を含まないシュレーディンガー方程式は (2.2) 式で定義された H を用いて、

$$H\Phi(\vec{\tau_1}, \vec{\tau_2} \ldots, \vec{\tau_N}) = E\Phi(\vec{\tau_1}, \vec{\tau_2} \ldots, \vec{\tau_N}) \tag{2.3}$$

と表される。これを解いて固有値 E を求めるのだが、r_{ij} を含んでいるので各電子毎に変数分離して独立に解くことはできない。しかし、N 電子の電子相関を含む完全な波動関数を求めることは実際上は不可能なほど大規模な計算となる。そこで Hartree-Fock 法を使って近似的な波動関数を求めることにする。

2.4 Hartree-Fock 法

2.4.1 Slater 行列式

スレーター行列式とは、

$$\Phi = \frac{1}{\sqrt{N!}} \begin{vmatrix} \phi_1(\vec{\tau_1}) & \cdots & \phi_N(\vec{\tau_1}) \\ \vdots & \ddots & \vdots \\ \phi_1(\vec{\tau_N}) & \cdots & \phi_N(\vec{\tau_N}) \end{vmatrix} \tag{2.4}$$

の様な形の波動関数のことである。$\vec{\tau}$ は、スピンと空間、両方の座標を含んだ変数である ($\vec{\tau} = (\vec{r}, \sigma)$)。スレーター行列式は、2 粒子の交換に対して反対称になっており、パウリの原理も満たしている。この式で、2 粒子を交換するということは、行を入れ替えるということなので、行列式の性質から符号が変わる。このことから、スレーター行列式が反対称性を持っていることが分かる。

N 電子系の波動関数を、この様なスレーター行列式に限定して、変分問題を解くというのが、これから説明する Hartree-Fock 法である。

2.4.2 Hartree-Fock 法 (He 様原子の場合)

まず初めに計算の簡単な He 様原子 (即ち $N = 2$ の場合) について Hartree-Fock 法を導く。一般の N の場合については、導出法は説明せず $N = 2$ の場合の方程式を含んだ一般化になっていることで納得してもらいたい。$N = 2$ の場合の導出は文献 [2] の説明をほぼ忠実に採録したものである。まず (2.4) 式のスレーター行列式は、

$$\Phi = \frac{1}{\sqrt{2!}} \begin{vmatrix} \phi_1(\vec{\tau_1}) & \phi_2(\vec{\tau_1}) \\ \phi_1(\vec{\tau_2}) & \phi_2(\vec{\tau_2}) \end{vmatrix} \tag{2.5}$$

となる。(2.5) の波動関数を使って、

$$\int \phi_i^* \phi_j d\tau = \delta_{ij} \tag{2.6}$$

という規格直交条件の下で、ハミルトニアンの期待値を求める。

$$\int\int \Phi^* H \Phi d\tau_1 d\tau_2 = \int\int \frac{1}{\sqrt{2}}(\phi_1(\tau_1)\phi_2(\tau_2) - \phi_1(\tau_1)\phi_2(\tau_2))^* H \frac{1}{\sqrt{2}}(\phi_1(\tau_1)\phi_2(\tau_2) - \phi_1(\tau_1)\phi_2(\tau_2))$$

$$= \frac{1}{2}\int\int \phi_1^*(\tau_1)\phi_2^*(\tau_2) H \phi_1(\tau_1)\phi_2(\tau_2) d\tau_1 d\tau_2 \tag{2.7}$$

$$+\frac{1}{2}\int\int \phi_1^*(\tau_2)\phi_2^*(\tau_1) H \phi_1(\tau_2)\phi_2(\tau_1) d\tau_1 d\tau_2 \tag{2.8}$$

$$-\frac{1}{2}\int\int \phi_1^*(\tau_1)\phi_2^*(\tau_2) H \phi_1(\tau_2)\phi_2(\tau_1) d\tau_1 d\tau_2 \tag{2.9}$$

$$-\frac{1}{2}\int\int \phi_1^*(\tau_2)\phi_2^*(\tau_1) H \phi_1(\tau_1)\phi_2(\tau_2) d\tau_1 d\tau_2 \tag{2.10}$$

次にハミルトニアンの一電子部分を、h_1,h_2 とすると、ハミルトニアンは

$$H = h_1 + h_2 + \frac{e^2}{4\pi\epsilon_0 r_{12}} \tag{2.11}$$

となる。これを代入すると、まず (2.7) 式は、

$$\begin{aligned}(2.7) = &\frac{1}{2}\int\int \phi_1^*(\tau_1)\phi_2^*(\tau_2) h_1 \phi_1(\tau_1)\phi_2(\tau_2) d\tau_1 d\tau_2 \\ &+\frac{1}{2}\int\int \phi_1^*(\tau_1)\phi_2^*(\tau_2) h_2 \phi_1(\tau_1)\phi_2(\tau_2) d\tau_1 d\tau_2 \\ &+\frac{1}{2}\int\int \phi_1^*(\tau_1)\phi_2^*(\tau_2) \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_1(\tau_1)\phi_2(\tau_2) d\tau_1 d\tau_2\end{aligned}$$

となる。h_1, h_2 はそれぞれ τ_1, τ_2 にのみ作用するので、

$$= \frac{1}{2}\int \phi_1^*(\tau_1) h_1 \phi_1(\tau_1) d\tau_1 \tag{2.12}$$

$$+\frac{1}{2}\int \phi_2^*(\tau_2) h_2 \phi_2(\tau_2) d\tau_2 \tag{2.13}$$

$$+\frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_1^*(\tau_1)\phi_2^*(\tau_2)\phi_1(\tau_1)\phi_2(\tau_2) d\tau_1 d\tau_2 \tag{2.14}$$

となる。(2.8) 式も同様に、

$$(2.8) = \frac{1}{2}\int \phi_2^*(\tau_1) h_1 \phi_2(\tau_1) d\tau_1 \tag{2.15}$$

$$+\frac{1}{2}\int \phi_1^*(\tau_2) h_2 \phi_1(\tau_2) d\tau_2 \tag{2.16}$$

$$+\frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_1^*(\tau_2)\phi_2^*(\tau_1)\phi_1(\tau_2)\phi_2(\tau_1) d\tau_1 d\tau_2 \tag{2.17}$$

となる。次に (2.9) 式は、

$$\begin{aligned}(2.9) = &-\frac{1}{2}\int\int \phi_1^*(\tau_1)\phi_2^*(\tau_2) h_1 \phi_1(\tau_2)\phi_2(\tau_1) d\tau_1 d\tau_2 \\ &-\frac{1}{2}\int\int \phi_1^*(\tau_1)\phi_2^*(\tau_2) h_2 \phi_1(\tau_2)\phi_2(\tau_1) d\tau_1 d\tau_2 \\ &-\frac{1}{2}\int\int \phi_1^*(\tau_1)\phi_2^*(\tau_2) \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_1(\tau_2)\phi_2(\tau_1) d\tau_1 d\tau_2\end{aligned}$$

$$
\begin{aligned}
&= -\frac{1}{2}\int \phi_2^*(\tau_2)\phi_1(\tau_2)d\tau_2 \int \phi_1^*(\tau_1)h_1\phi_2(\tau_1)d\tau_1 \\
&\quad -\frac{1}{2}\int \phi_1^*(\tau_1)\phi_2(\tau_1)d\tau_1 \int \phi_2^*(\tau_2)h_2\phi_1(\tau_2) \\
&\quad -\frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_1)\phi_2^*(\tau_2)\phi_1(\tau_2)\phi_2(\tau_1)d\tau_1 d\tau_2
\end{aligned}
$$

となり、(2.6) 式により第一項、二項が消えて

$$
(2.9) = -\frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_1)\phi_2^*(\tau_2)\phi_1(\tau_2)\phi_2(\tau_1)d\tau_1 d\tau_2 \tag{2.18}
$$

となる。(2.10) も同様に、

$$
(3.9) = -\frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_2)\phi_2^*(\tau_1)\phi_1(\tau_1)\phi_2(\tau_2)d\tau_1 d\tau_2 \tag{2.19}
$$

となる。
これらを全て足し合わせると、

$$
\int\int \Phi^* H \Phi d\tau_1 d\tau_2 = H_{11} + H_{22} + J_{12} - K_{12} \tag{2.20}
$$

となる。ここで H_{11},H_{22},J_{12},K_{12} は、

$$
\begin{aligned}
H_{11} &= (2.12) + (2.16) \\
&= \frac{1}{2}\int \phi_1^*(\tau_1)h_1\phi_1(\tau_1)d\tau_1 + \frac{1}{2}\int \phi_1^*(\tau_2)h_2\phi_1(\tau_2)d\tau_2
\end{aligned}
$$

$$
= \int \phi_1^*(\vec{\tau})\left[-\frac{\hbar^2}{2m}\vec{\nabla}^2 - \frac{Ze^2}{4\pi\epsilon_0 r}\right]\phi_1(\vec{\tau})d\tau\ , \tag{2.21}
$$

$$
\begin{aligned}
H_{22} &= (2.13) + (2.15) \\
&= \frac{1}{2}\int \phi_2^*(\tau_2)h_2\phi_2(\tau_2)d\tau_2 + \frac{1}{2}\int \phi_2^*(\tau_1)h_1\phi_2(\tau_1)d\tau_1
\end{aligned}
$$

$$
= \int \phi_2^*(\vec{\tau})\left[-\frac{\hbar^2}{2m}\vec{\nabla}^2 - \frac{Ze^2}{4\pi\epsilon_0 r}\right]\phi_2(\vec{\tau})d\tau\ , \tag{2.22}
$$

$$
\begin{aligned}
J_{12} &= (2.14) + (2.17) \\
&= \frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_1)\phi_2^*(\tau_2)\phi_1(\tau_1)\phi_2(\tau_2)d\tau_1 d\tau_2 \\
&+ \frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_2)\phi_2^*(\tau_1)\phi_1(\tau_2)\phi_2(\tau_1)d\tau_1 d\tau_2
\end{aligned}
$$

$$
= \int\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\left|\phi_1(\tau_1)\right|^2\left|\phi_2(\tau_2)\right|^2 d\tau_1 d\tau_2\ , \tag{2.23}
$$

$$
\begin{aligned}
-K_{12} &= (2.18) + (2.19) \\
&= -\frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_1)\phi_2^*(\tau_2)\phi_1(\tau_2)\phi_2(\tau_1)d\tau_1 d\tau_2 \\
&\quad -\frac{1}{2}\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_2)\phi_2^*(\tau_1)\phi_1(\tau_1)\phi_2(\tau_2)d\tau_1 d\tau_2
\end{aligned}
$$

$$= \quad -\int\int \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_1^*(\tau_1)\phi_2^*(\tau_2)\phi_1(\tau_2)\phi_2(\tau_1)d\tau_1 d\tau_2 \tag{2.24}$$

となっている。
変分問題を適用するので、(2.20) を極小にすることを考える。未定係数法を用いることにし、

$$I \equiv \int\int \Phi^* H \Phi d\tau_1 d\tau_2 - \sum_{i,j} \varepsilon(i,j) \int \phi_i^* \phi_j d\tau \tag{2.25}$$

を極小にする ($\varepsilon(i,j)$ が未定係数)。
まず、

$$\phi_1^* \longrightarrow \phi_1^* + \delta\phi_1^* \tag{2.26}$$

のように変わるとすると、極値となるためのの条件は $\delta I = 0$ なので、

$$\begin{aligned} \delta I &= \delta H_{11} + \delta H_{22} + \delta J_{12} - \delta K_{12} - \delta \sum_{i,j} \varepsilon(i,j) \int \phi_i^* \phi_j d\tau & (2.27)\\ &= 0 & (2.28) \end{aligned}$$

となる。$\delta H_{11}, \delta H_{22}, \delta J_{12}, -\delta K_{12}, -\delta \sum_{i,j} \varepsilon(i,j) \int \phi_i^* \phi_j d\tau$ はそれぞれ次のようになる。

$$\begin{aligned} \delta H_{11} &= \int \delta\phi_1^*(\tau_1) \left[-\frac{\hbar^2}{2m}\vec{\nabla}_1^2 - \frac{Ze^2}{4\pi\epsilon_0 r_1} \right] \phi_1(\tau_1) d\tau_1 \ , \\ \delta H_{22} &= 0 \ , \\ \delta J_{12} &= \int \delta\phi_1^*(\tau_1) \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_1(\tau_1) \left|\phi_2(\tau_2)\right|^2 d\tau_1 d\tau_2 \ , \\ -\delta K_{12} &= -\int \delta\phi_1^*(\tau_1) \left[\int \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_2^*(\tau_2)\phi_1(\tau_2) d\tau_2 \right] \phi_2(\tau_1) d\tau_1 \ , \\ -\delta \sum_{i,j} \varepsilon(i,j) \int \phi_i^* \phi_j d\tau &= -\sum_{1,j} \varepsilon(1,j) \int \delta\phi_1^* \phi_j d\tau \ . \end{aligned}$$

これらを (2.27) に代入すると、

$$\begin{aligned} = \int \delta\phi_1^*(\tau_1) \Bigg[&\left(-\frac{\hbar^2}{2m}\vec{\nabla}_1^2 - \frac{Ze^2}{4\pi\epsilon_0 r_1} \right) \phi_1(\tau_1) + \int \frac{e^2}{4\pi\epsilon_0 r_{12}} \left|\phi_2(\tau_2)\right|^2 d\tau_2 \phi_1(\tau_1) \\ &- \left(\int \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_2^*(\tau_2)\phi_1(\tau_2) d\tau_2 \right) \phi_2(\tau_1) - \sum_{1,j} \varepsilon(1,j) \int \phi_j \Bigg] d\tau_1 \end{aligned}$$

$$\begin{aligned} = \int \delta\phi_1^*(\tau_1) \Bigg[&\left\{ \left(-\frac{\hbar^2}{2m}\vec{\nabla}_1^2 - \frac{Ze^2}{4\pi\epsilon_0 r_1} \right) + \int \frac{e^2}{4\pi\epsilon_0 r_{12}} \left|\phi_2(\tau_2)\right|^2 d\tau_2 - \varepsilon(1,1) \right\} \phi_1(\tau_1) \\ &- \left\{ \int \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_2^*(\tau_2)\phi_1(\tau_2) d\tau_2 + \varepsilon(1,2) \right\} \phi_2(\tau_1) \Bigg] d\tau_1 = 0 \end{aligned} \tag{2.29}$$

となる。$\delta\phi_1^*(\tau_1)$ は任意なので取り除いて変形すると、

$$\left\{ \left(-\frac{\hbar^2}{2m}\vec{\nabla}_1^2 - \frac{Ze^2}{4\pi\epsilon_0 r_1} \right) + \int \frac{e^2}{4\pi\epsilon_0 r_{12}} \left|\phi_2(\tau_2)\right|^2 d\tau_2 \right\} \phi_1(\tau_1) - \left\{ \int \frac{e^2}{4\pi\epsilon_0 r_{12}} \phi_2^*(\tau_2)\phi_1(\tau_2) d\tau_2 \right\} \phi_2(\tau_1)$$

$$= \varepsilon(1,1)\phi_1(\tau_1) + \varepsilon(1,2)\phi_2(\tau_1) \tag{2.30}$$

が得られる。同様に、$\phi_2^* \longrightarrow \phi_2^* + \delta\phi_2^*$ とすると、

$$\left\{\left(-\frac{\hbar^2}{2m}\vec{\nabla}_2^2 - \frac{Ze^2}{4\pi\epsilon_0 r_2}\right) + \int \frac{e^2}{4\pi\epsilon_0 r_{12}} \left|\phi_1(\tau_1)\right|^2 d\tau_1\right\}\phi_2(\tau_2) - \left\{\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_1)\phi_2(\tau_1)d\tau_1\right\}\phi_1(\tau_2)$$

$$= \varepsilon(2,2)\phi_2(\tau_2) + \varepsilon(2,1)\phi_1(\tau_2) \tag{2.31}$$

が得られる。次に、ϕ_1, ϕ_2 に適当な一次変換を施して、

$$\begin{vmatrix} \varepsilon(1,1) & \varepsilon(1,2) \\ \varepsilon(2,1) & \varepsilon(2,2) \end{vmatrix} \tag{2.32}$$

を対角化して、

$$\begin{vmatrix} \varepsilon(1,1) & 0 \\ 0 & \varepsilon(2,2) \end{vmatrix} \tag{2.33}$$

とすると、(2.30) と (2.31) 式は、

$$\left\{\left(-\frac{\hbar^2}{2m}\vec{\nabla}_1^2 - \frac{Ze^2}{4\pi\epsilon_0 r_1}\right) + \int \frac{e^2}{4\pi\epsilon_0 r_{12}} \left|\phi_2(\tau_2)\right|^2 d\tau_2\right\}\phi_1(\tau_1) - \left\{\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_2^*(\tau_2)\phi_1(\tau_2)d\tau_2\right\}\phi_2(\tau_1)$$

$$= \varepsilon(1,1)\phi_1(\tau_1)\ . \tag{2.34}$$

この式と、この式の 1 と 2 を入れ替えた式

$$\left\{\left(-\frac{\hbar^2}{2m}\vec{\nabla}_2^2 - \frac{Ze^2}{4\pi\epsilon_0 r_2}\right) + \int \frac{e^2}{4\pi\epsilon_0 r_{12}} \left|\phi_1(\tau_1)\right|^2 d\tau_1\right\}\phi_2(\tau_2) - \left\{\int \frac{e^2}{4\pi\epsilon_0 r_{12}}\phi_1^*(\tau_1)\phi_2(\tau_1)d\tau_1\right\}\phi_1(\tau_2)$$

$$= \varepsilon(2,2)\phi_2(\tau_2) \tag{2.35}$$

が得られる。これが Hartree-Fock 法における方程式の正準形で、これらを連立させて解くことになる。

2.4.3 Hartree-Fock 法 (N 電子系の場合)

先ほどの He 様原子と同じことを N 電子系で行えば、結果として下記の方程式が得られる。

$$\left[-\frac{\hbar^2}{2m}\vec{\nabla}^2-\frac{Ze^2}{4\pi\epsilon_0 r}+\sum_{j=1}^{N}\int\frac{e^2}{4\pi\epsilon_0|\vec{r}-\vec{r'}|}|\phi_j(\tau')|^2d\tau'\right]\phi_i(\tau)$$

$$-\sum_{j=1}^{N}\left[\int\frac{e^2}{4\pi\epsilon_0|\vec{r}-\vec{r'}|}\phi_j^*(\tau')\phi_i(\tau')d\tau'\right]\phi_j(\tau)=\varepsilon_i\phi_i(\tau) \tag{2.36}$$

ここで $1\leq i\leq N$ であり、したがってこれは N 本の連立微積分方程式である。また、$\phi_i(r)$ は規格化された波動関数である。j についての和は、占有されているスピン軌道についての和である。
左辺第 1 項と第 2 項は一電子エネルギーの和、第 3 項と第 4 項は電子間のクーロンエネルギーの和である。第 3 項の形の積分をクーロン積分、第 4 項の形の積分を交換積分という。
一見するとクーロンエネルギーの項に $i=j$ として自分自身からのクーロン力が入ってしまうように見えるが、これは交換項のなかの $i=j$ の項と打ち消し合うので問題はない。

2.4.4 局所近似（Slater 近似）

(2.36) 式の左辺の交換項は、

$$U(\tau,\tau')=-\sum_{j=1}^{N}\frac{e^2\phi_j^*(\tau')\phi_j(\tau)}{4\pi\epsilon_0|\vec{r}-\vec{r'}|} \tag{2.37}$$

と定義される非局所ポテンシャル $U(\tau,\tau')$ を用いると、

$$\int U(\tau,\tau')\phi_i(\tau')d\tau' \tag{2.38}$$

と表すことができる。非局所ポテンシャルをそのまま扱おうとすると数値計算の量が非常に大きくなるので、本論文ではこれを下記のような局所ポテンシャルで近似することにする。

$$\int U(\tau,\tau')\phi_i(\tau')d\tau'\cong U_{\rm s}(\tau)\phi_i(\tau)\ , \tag{2.39}$$

$$U_{\rm s}(\tau)=-\frac{e^2}{4\pi\epsilon_0}\left(\frac{3}{\pi}\rho(\vec{r})\right)^{\frac{1}{3}}\phi_i(\tau)\ , \tag{2.40}$$

$$\rho(\vec{r})=\sum_{\sigma}\sum_{j=1}^{N}\phi_j^*(\vec{r},\sigma)\phi_j(\vec{r},\sigma)\ . \tag{2.41}$$

これをスレーター近似と呼ぶ。

2.4.5 全エネルギーの表式

交換ポテンシャルのスレーター近似は以下のようにして導かれたものである。
空間的に一様な系 (電子のフェルミガス) での交換項の期待値は単位体積あたり
$E_{\rm ex}=-\frac{3}{4}e^2\left(\frac{3}{\pi}\right)^{\frac{1}{3}}\rho^{\frac{4}{3}}$ であることが解析的に示されている。局所密度近似は、このエネルギーの表式が ρ が位置の関数として空間的に変動する場合も、成立すると考える近似法である。即ち、

$$E_{\rm ex}=-\frac{3}{4}e^2\left(\frac{3}{\pi}\right)^{\frac{1}{3}}\int(\rho(\vec{r}))^{\frac{4}{3}}\,d^3r \tag{2.42}$$

と近似するのである。
この近似を採用すると、N 電子系の系の全エネルギー E は下式で表される。

$$E = E_{\rm K} + E_{\rm N} + E_{\rm D} + E_{\rm ex} \ , \tag{2.43}$$

$$E_{\rm K} = \sum_{i=1}^{N} T_i \ , \ T_i = -\frac{\hbar^2}{2m_{\rm e}} < i|\vec{\nabla}^2|i >, \tag{2.44}$$

$$E_{\rm N} = V_i \ , \ V_i =< i|V(\vec{r})|i > \ , \ V(\vec{r}) = -\frac{Ze^2}{r}, \tag{2.45}$$

$$E_{\rm D} = \frac{e^2}{2} \int d^3r_1 d^3r_2 \frac{\rho(\vec{r_1})\rho(\vec{r_2})}{|\vec{r_1} - \vec{r_2}|} \ . \tag{2.46}$$

$E_{\rm K}$ は運動エネルギー、$E_{\rm N}$ は外場のポテンシャルエネルギー (原子では核の作るクーロンポテンシャル)、$E_{\rm D}$ は電子間に働くクーロン斥力の直接項部分である。
E を i 番目の電子の波動関数 ϕ_i で変分した結果が Hartree-Fock 法の一粒子ハミルトニアン h の定義を以下のように与える。

$$\frac{\delta E}{\delta \phi_i^*} = h\phi_i \ . \tag{2.47}$$

実際に左辺の変分を実行してみよう。

$$\begin{aligned}
\frac{\delta E_{\rm K}}{\delta \phi_i^*(\tau)} &= -\frac{\hbar^2}{2m_{\rm e}} \sum_{j=1}^{N} \frac{\delta}{\delta \phi_i^*(\tau)} \int \phi_j^*(\tau') \vec{\nabla_{\tau'}}^2 \phi_j(\tau') d\tau' && (2.48)\\
&= -\frac{\hbar^2}{2m_{\rm e}} \vec{\nabla}^2 \phi_i(\tau) \ , && (2.49)\\
\frac{\delta E_{\rm N}}{\delta \phi_i^*(\tau)} &= \sum_{j=1}^{N} \frac{\delta}{\delta \phi_i^*(\tau)} \int \phi_j^*(\tau') V(\vec{r'}) \phi_j(\tau') d\tau' && (2.50)\\
&= V(\vec{r})\phi_i(\tau) \ , && (2.51)\\
\frac{\delta E_{\rm D}}{\delta \phi_i^*(\tau)} &= \frac{e^2}{2} \int d\tau_1 d\tau_2 \frac{1}{|\vec{r_1} - \vec{r_2}} \left\{ \frac{\delta \rho(\vec{r_1})}{\delta \phi_i^*(\tau)} \rho(\vec{r_2}) + \rho(\vec{r_1}) \frac{\delta \rho(\vec{r_2})}{\delta \phi_i^*(\tau)} \right\} && (2.52)\\
&= \frac{e^2}{2} \left\{ \int d\tau_2 \frac{\rho(\vec{r_2})}{|\vec{r} - \vec{r_2}|} \phi_i(\tau) + \int d\tau_1 \frac{\rho(\vec{r_1})}{|\vec{r_1} - \vec{r}|} \phi_i(\tau) \right\} && (2.53)\\
&= e\varphi(\vec{r})\phi_i(\tau) \ . && (2.54)
\end{aligned}$$

ただし、ここで

$$\varphi(\vec{r}) = e \int d^3r_1 \frac{\rho(\vec{r_1})}{|\vec{r} - \vec{r_1}|} \tag{2.55}$$

は電位である。最後に、

$$\begin{aligned}
\frac{\delta E_{\rm ex}}{\delta \phi_i^*(\tau)} &= -\frac{3}{4} e^2 \left(\frac{3}{\pi} \right)^{\frac{1}{3}} \int d^3r_1 \frac{\delta \left(\rho(\vec{r_1}) \right)^{\frac{4}{3}}}{\delta \phi_i^*(\tau)} && (2.56)\\
&= -\frac{3}{4} e^2 \left(\frac{3}{\pi} \right)^{\frac{1}{3}} \frac{4}{3} \left(\rho(\vec{r}) \right)^{\frac{1}{3}} \phi_i(\tau) \ . && (2.57)
\end{aligned}$$

したがって

$$h = -\frac{\hbar^2}{2m} \vec{\nabla}^2 + V(\vec{r}) + e\psi(\vec{r}) - e^2 \left(\frac{3}{\pi} \rho(\vec{r}) \right)^{\frac{1}{3}} \tag{2.58}$$

であることが分かる。右辺の最後の項が交換ポテンシャルを局所近似したものである。

上記の局所近似した交換ポテンシャルの導出法は Dirac によるものであり、原子核物理ではこの方法によって陽子間のクーロン力の交換項を評価するのが一般的である。

一方、Slater は Dirac の結果の $\frac{3}{2}$ 倍の強さの交換ポテンシャルを使用た。これは局所密度近似をエネルギー密度に対してでなく、ポテンシャルエネルギーに対して適用したために生じた差異であると言われている。エネルギー密度との矛盾があるため本論文では Dirac の表式を採用した。経験的には Dirac の式と Slater の式の中間に最も良い結果を与えるポテンシャル強度があると言われている。これは電子間の斥力により、波動関数の反対称化の効果以上に電子間の斥力的相関が強くなっていることの反映であろう。

なお、Slater の式にせよ中間的な式にせよ Dirac の式と異なるポテンシャルを用いる場合は、エネルギー密度の表式もそれにあわせて定数倍すべきことだけは確かである。

さて、ϕ_i が h の固有状態であり、即ち

$$h\phi_i = E_i\phi_i \ (1 \leq i \leq N) \tag{2.59}$$

を満足する自己無撞着解であるときは E_i を利用することで E の計算を簡略化することができる。$E_{\rm K}$ と $E_{\rm N}$ が $\sum_{i=1}^{N} E_i$ の中に完全に取り込まれていることは明らかであるが $E_{\rm D}$ や $E_{\rm ex}$ は過剰に取り込まれているので、その分を差し引かねばならないことが以下のようにして示せる。

α を定数として、ρ^α の形のエネルギー密度を考えよう。$\alpha = \frac{4}{3}$ の場合が局所近似した交換項 $E_{\rm ex}$ に対応し、$\alpha = 2$ の場合は E_D と本質的に同じである。
(ρ^2 を $\int f(\vec{r_1},\vec{r_2})\rho(\vec{r_1})\rho(\vec{r_2})d^3r_1d^3r_2$ に変えても以下の計算はほとんど変わらない。)

$$E_{\rm V} = \int \rho^\alpha d^3r \tag{2.60}$$

とし、そこから導かれるポテンシャル V を

$$\frac{\delta E_{\rm V}}{\delta \phi_i^*} = \alpha\rho^{\alpha-1}\phi_i = V\phi_i \tag{2.61}$$

で定義する。各粒子の V の期待値は

$$V_i = \int \phi_i^* V \phi_i d^3\tau \tag{2.62}$$

である。したがって

$$\sum_{i=1}^{N} V_i = \int \alpha\rho^{\alpha-1} \sum_{i=1}^{N} |\phi_i|^2 d^3\tau \tag{2.63}$$

$$= \alpha \int \rho^\alpha d^3r \tag{2.64}$$

$$= \alpha E_{\rm V} \tag{2.65}$$

である。両辺から $(\alpha-1)E_{\rm V}$ を引くと、

$$E_{\rm V} = \sum_{i=1}^{N} V_i - (\alpha-1)E_{\rm V} \tag{2.66}$$

が得られる。$\alpha = 2$ および $\alpha = \frac{4}{3}$ を代入すれば下記の E の計算式が成り立つことが分かる。

$$E = \sum_{i=1}^{N} E_i - E_{\rm D} - \frac{1}{3}E_{\rm ex}\ . \tag{2.67}$$

2.4.6 Dirac-Fock 法

これまでは、説明を非相対論で進めてきたが、実際に行った計算は、相対論の効果を取り入れた Dirac-Fock 法で行った。Schrödinger 方程式を Dirac 方程式に置き換えることで、相対論の効果を取り入れたものであり、自己無撞着性に関しては Hartree-Fock 法と Dirac-Fock 法とで本質的な違いは何もない。

2.5 Dirac-Fock 法での電子配置

相対論の効果を取り入れると方位量子数 l が $l \neq 0$ のとき、スピン軌道相互作用による微細構造が生じるので n, l 以外にも、角運動量 $j = l \pm \frac{1}{2}$ によってエネルギー準位が 2 つに分かれる。したがって電子配置は表 2.1 のようではなく次の表 2.2 のようになる。

表 2.2: 元素の電子配置

エネルギー準位	1s 1/2	···	3s 1/2	3p 1/2	3p 3/2	3d 3/2	3d 5/2	4s 1/2	4p 1/2
18Ar	2								
19K	2		2	2	4			1	
20Ca	2		2	2	4			2	
21Sc	2		2	2	4	1		2	
22Ti	2		2	2	4	2		2	
23V	2		2	2	4	3		2	
24Cr	2		2	2	4	4	1	1	
25Mn	2		2	2	4	4	1	2	
26Fe	2		2	2	4	4	2	2	
27Co	2		2	2	4	4	3	2	
28Ni	2		2	2	4	4	4	2	
29Cu	2		2	2	4	4	6	1	
30Zn	2		2	2	4	4	6	2	
31Ga	2		2	2	4	4	6	2	1
32Ge	2		2	2	4	4	6	2	2
33As	2		2	2	4	4	6	2	3
34Se	2		2	2	4	4	6	2	4
35Br	2		2	2	4	4	6	2	5
36Kr	2		2	2	4	4	6	2	6

表 2.2 のように、n, l が同じ場合は、j の値が小さい方がエネルギー準位が低いので、そちらに電子が先に詰まっていく。実際に Dirac-Fock 法で計算するときは、まず電子配置を決めてその電子配置に対する自己無撞着解を求める。電子配置としては表 2.2 を参考にして、その配置に近いいくつかの配置に対して解を求め、それらの中で最も全エネルギーの低いものを基底状態解として採用することとした。

2.5.1 Dirac-Fock 法による計算結果

中性原子の電子配置の決定の実例を Cr と Cu について以下に示す。

(1) Cr(= 24)
実験的に知られた基底状態 $(3p)^6(3d)^5(4s)^1$ のときのエネルギーは、

$$E = -2.88805086 \times 10^{-2}[\mathrm{MeV}]$$

となる。
4s から 3d に電子が入れ替わらずに 4s に 2 つ入ったままの状態、$(3p)^6(3d)^4(4s)^2$ のときのエネルギーは、

$$E' = -2.88795417 \times 10^{-2}[\mathrm{MeV}]$$

となり二つのエネルギー差は、

$$\begin{aligned} E - E' &= -9.668 \times 10^{-7}[\mathrm{MeV}] \\ &= -0.97[\mathrm{eV}] \end{aligned}$$

となる。

(2) Cu($Z = 29$)
実験的に知られた基底状態 $(3d)^{10}(4s)^1$ のときのエネルギーは、

$$E = -4.5457566 \times 10^{-2}[\mathrm{MeV}]$$

となる。4s から 3d に電子が入れ替わらずに 4s に 2 つ入ったままの状態、$(3d)^9(4s)^2$ のときのエネルギーは、

$$E' = -4.5453985 \times 10^{-2}[\mathrm{MeV}]$$

となり、二つのエネルギー差は、

$$\begin{aligned} E - E' &= -3.580 \times 10^{-6}[\mathrm{MeV}] \\ &= -3.58[\mathrm{eV}] \end{aligned}$$

となる。

この他の原子についても調べてみたが、実験的に知られている配位の基底状態が最も低いエネルギーを持つ状態であるということを、確認することができた。この結果は Dirac-Fock 法という近似の精度の高さを実証するものである。また逆に言えば、現実の原子における電子配位を決める際のエネルギー尺度において電子間の相関はあまり重要な役割を果たしていないことを示唆しているとも考えることができる。

2.5.2 電子質量の変化に対する配位間のエネルギー差の感度

Cr(24) の基底状態 $(3p)^6(3d)^5(4s)^1$ のエネルギー E と、励起状態 $(3p)^6(3d)^4(4s)^2$ のエネルギー E' との差を、電子質量を変化させてプロットした。

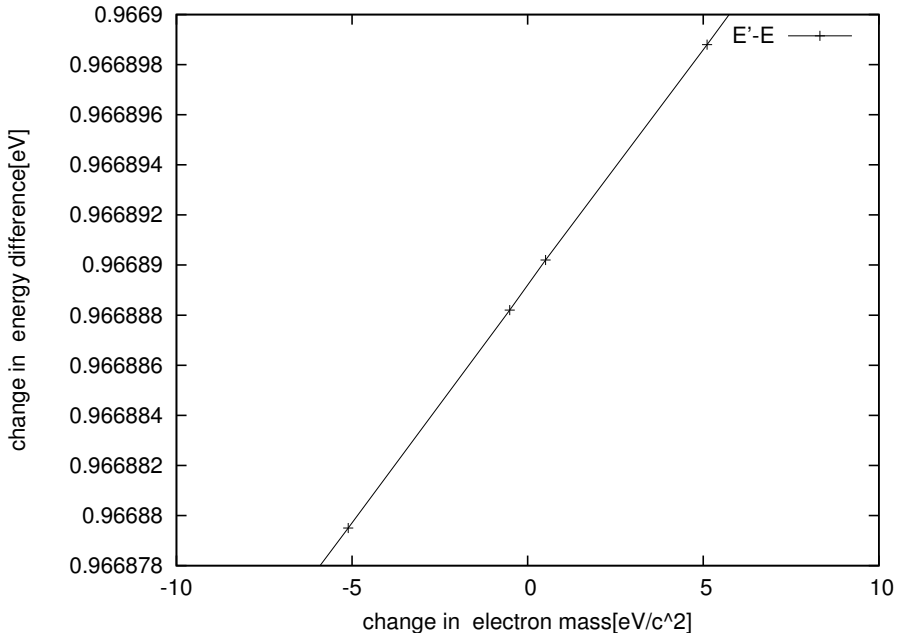


図 2.3: 電子質量を人為的に変化させることの、Cr の 2 つの配位間のエネルギー差への影響

図 2.3 は、横軸は電子質量の差（変化させた電子質量から元の電子質量を引いた）を、縦軸には基底状態と励起状態のエネルギー差をプロットした。

図から静止質量が 1[eV] ずれても基底状態と励起状態のエネルギー差が約 1.9×10^{-6}[eV] しかずれていないことが分かる。このことからエネルギーを静止質量に対して 6 桁の精度で求めるからといって、電子の静止質量も 6 桁の精度が必要というわけではないことがわかる。

2.5.3 計算で求めたイオン化エネルギー

計算で求めたイオン化エネルギーと実験値との比較を図 2.4 に示した。計算では、一価の正イオンの基底状態の全エネルギーから中性原子の基底状態の全エネルギーを引いてイオン化エネルギーを計算した。中性原子の場合と同様に、一価の正イオンの配位についても、いくつかの配位のエネルギーを比べて最もエネルギーの低い配位を採用した。

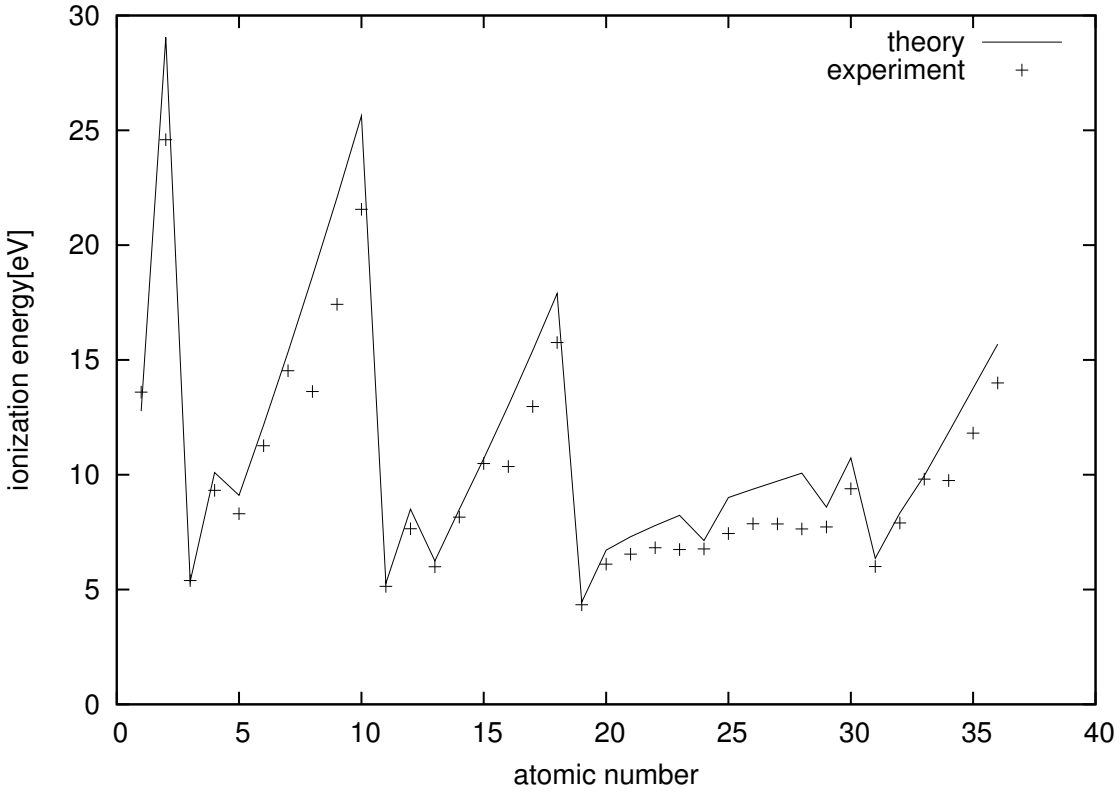


図 2.4: イオン化エネルギー

実線が計算による値、点が実験値である。計算結果は実験値と定量的によく一致しており原子に対する Dirac-Fock 法の信頼性の高さを示している。ただし系統的な相違点として殻の後半 ($8 \leq Z \leq 10, 16 \leq Z \leq 18, 33 \leq Z \leq 35$) においてイオン化エネルギーが一定値だけ下方にシフトする挙動が計算結果には全く現れていないことがあげられる。これは殻内の電子数の増加による電子相関の強度の増大が背景にあるのかもしれない。
また、$20 \leq Z \leq 23, 25 \leq Z \leq 28$ でも誤差が大きくなっているが、これらの原子は $4s$ 軌道に 2 個の電子の入った配位をとる原子とちょうど一致しており、軌道の特殊性を示唆しているように思われる。

第3章　結論

本論文ではまず、水素様原子のエネルギー準位の特徴を論じ、そこに現れる相対論的効果による微細構造について説明した。第 2 章で扱う多電子系では動径波動関数の数値解法によってエネルギーを求めるため、数値解法で問題となる $1/r$ 依存性に対処するため変数変換を導入した。最後に数値解法の精度を確認した。

第 2 章では Dirac-Fock 法を用いて、多電子原子の基底状態の電子配位とイオン化エネルギーを計算して求めた。基底状態の電子配位はすべて実験的な配位と一致した。イオン化エネルギーも実験値をおおむねよく再現したが、一部再現できない挙動も見られた。これは電子相関によると思われる。

関連図書

[1] 「物理学辞典」, 培風館 (1984),p.1439

[2] 高柳和夫:「原子分子物理学」, 朝倉書店 (2000)

[3] 有馬朗人:「原子と原子核」, 朝倉書店 (1982)

謝辞

本論文を作成するにあたり，田嶋直樹 先生には終始御厚いご指導をして頂いたことに誠に感謝し，お礼申し上げます。また 林明久 先生, 鈴木敏男 先生, にも本研究及び日常的なことにおいても，実に丁寧なご指導，お世話をして頂きました。計算に用いた Dirac-Fock 法プログラムは大学院博士前期課程在籍の三和之浩氏、山田昌平氏の開発してきた原子核の相対論的平均場模型のプログラムを修正して作られたものであり、プログラムを提供していただいたことに対し、両氏に深く感謝いたします。本研究に対してご意見を頂いた，多くの物理工学科先生方にもお礼申し上げ，謝辞の言葉とさせて頂きます。

付録　Program List

```
/*
-----------------------------------------------------------------------------------------------------
Dirac-Fock (reletivistic Hartree-Fock) calculation of atomic systems
  The Fock term is treated in the Slater approximation (in the manner of energy densiy functional)
  For a single type of leptons.

2008/01/26: debugged version
2008/01/24: completed, renamed as difo1.c
2008/01/18-23:created by utilizing atom1a.c and hydrogen2.c, named as atom2a.c
2008/01/08,14,15 : created by modifying rmf3a.c and named atom1a.c
-----------------------------------------------------------------------------------------------------
*/

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

// Arrays to store the energies and wavefunctions of the eigenstates
#define ELECTRON 0 // idxt=0 for electrons
#define IDXT 1    // idxt (index t) = 0..IDXT=2->1
                  //    idxt = ELECTRON (=0) : electron
#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 7    // idxj(index j)=0..IDXJ-1
                  //    J = idxj+1/2
#define NODE 7    // node=0..NODE-1
                  //    node=0 for the gournd state
#define IDXR  10001 // idxr(index r)=0..IDXR-1
#define IDXRD (2*IDXR-1) // for half grid size
#define NSP  50   // the maximum number of single-particle states to be stored

double rx[IDXRD];   // radial grid points (fm)
double drdx[IDXRD]; // derivarive of scale transformation function

                                      // "s" stands for "stored in memory"
int    ispx;
double Es[NSP];       // energy (MeV)
double Fs[NSP][IDXR]; // large component of the radial wavefunction
double Gs[NSP][IDXR]; // small component of the radial wavefunction
                                      // r = idxr*dr  : idxr = 0..IDXR-1
double Vs[IDXT][IDXRD];              // scalar potential (MeV)
                                               Vs[i] = Vs(r=dr*i/2)
double Vv[IDXT][IDXRD];              // vector potential (MeV)
                                              Vv[i] = Vv(r=dr*i/2)
double Dens[IDXT][IDXRD];            // Dens[0][idxrd] :electron |F|^2+|G|^2
                                      // r = idxrd*dr/2 : idxrd = 0..2*IDXR-2
```

```
double phi[IDXT][IDXRD];           // electric potential of ELECTRONS [MeV/e]
double source[IDXT][IDXRD];        // source term of electric fields

double Vcn[IDXRD];                 // Coulomb potential created
                                          by the atomic nucleus {MeV]

// physical constants
const double HbC = 197.326960;          // h-bar c [MeV fm^2]
const double fineStructureConstant = 1.0/137.03599976;
                                              // fine structure constant
const double Ebig  = 9.99e+30;
const double Ebigg = 9.999e+30;

// global variables
int Anum; // atomic number
int Mnum; // mass number
double Rnucl; // nuclear radius (fm)
int idxtGlobal;
int Ne[IDXT]; // the number of electrons
double Mass[IDXT] = {0.510998902}; // electron mass (MeV/c^2)
double Coupling[IDXT]; // 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 Rmatch; // (fm) The radius where the forward and backward solutions meet.
double RadialGridSpacing; // (fm) Copy to a local variable "dr" and use "dr".

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
} Idxr;   // NB) IDXR is macro, Idxr is structure, idxr is int

typedef struct{
  double e ; // energy {MeV)
  int t ; // idxt
  int p ; // idxp
  int j ; // idxj
  int n ; // idxn
  int o ; // occupation number
} orbital;

orbital orb[NSP];

// prototype declarations of functions
int setParameters();
int writeParameters(FILE *FO);
int labelOrbitals(int idxt);
int printOrbitals(int idxt);
int prog1();
int prog2(int idxt, orbital *orb, double *sumspe);
int writeWavefunction(FILE *FO, int idxp, int idxj, int idxn, int occu,double *Fs, double *Gs);
// int swap : defined before function sort, only by which it is called.
int sort(orbital *orb);
int NuclearCoulombPotential();
int initialPotential(int sw);
int idx2qn(int sw,int idxp,int idxj, double *J,int *w,int *L,int *kappa);
```

```
int setRadialGrid(int sw);
int solveDirac(int idxp,int idxj,int idxn, double mass, double *Esa, double *Fsa, double *Gsa);
// 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, double mass, int *node, double*maco
         ,double *Fsa, double *Gsa);
int PoissonRef(int imax,double h,double *rx,double *drdx,double *source,double *phi);
int Poisson(int imax,double dx,double *rx,double *drdx,double *source,double *phi);
int interpolate(double *y, int n);

//-------------------------------------------------------------------------------------
int main(){
  setParameters();
  writeParameters(stdout);
  labelOrbitals(ELECTRON);
  setConfiguration();
  prog1();
}

//-------------------------------------------------------------------------------------
int setParameters(){
  int idxt;
  for(idxt=0;idxt<IDXT;idxt++){
    Coupling[idxt]=sqrt(4*M_PI*HbC*fineStructureConstant);
  }
}

//-------------------------------------------------------------------------------------
int writeParameters(FILE *FO)
{ // writes the physical parameters of the model to a stream
  fprintf(FO,"hbar*c=%12.6f (MeV*fm)\n",HbC);
  fprintf(FO,"electron mass=%.9e (MeV/c^2)\n",Mass[ELECTRON]);
  fprintf(FO,"fine structure constant=1/%.9f\n",1.0/fineStructureConstant);
  fprintf(FO,"Coulomb Coupling Constant=%.6e\n",Coupling[ELECTRON]);
}

//-------------------------------------------------------------------------------------
int labelOrbitals(idxt){
  int idxp,idxj,idxn;
  char *oam = "spdfghijklmn";
  int n,l,j,js,isp,degen,ntot;
  const int debug=0;

  ntot=0;
  isp=0;
  for(n=1;n<8;n++){ // principal quantum number
    for(l=0;l<n;l++){ // orbital angular momentum
      js=l-1; if(js<0) js=0;
      for(idxj=js;idxj<=l;idxj++){ // angular momentum -1/2
        idxn=n-l-1; // the number of radial nodes
        idxp=l-idxj;
        degen=2*idxj+2;
        ntot+=degen;
        if(debug) printf("(%2d) %d%c %2d/2 (t,p,j,n)=(%d %d %d %d) %2d %2d\n"
                 ,isp,n,oam[l],2*idxj+1,idxt,idxp,idxj,idxn,degen,ntot);
        if(isp>NSP){
          fprintf(stderr,"labelOrbitals: error: isp>NSP %d %d\n",isp,NSP);
```

```
          exit(1);
        }
        orb[isp].e=0.0;
        orb[isp].t=idxt;
        orb[isp].p=idxp;
        orb[isp].j=idxj;
        orb[isp].n=idxn;
        orb[isp].o=0;
        isp++;
      }
    }
  }
  ispx=isp;
}

//-----------------------------------------------------------------------------------------
int printOrbitals(int idxt){
  int ntot,isp,l,n,degen,occu;
  char *oam = "spdfghijklmn"; // letters for orbital angular momentum
  orbital *ob;

  ntot=0;
  for(isp=0;isp<ispx;isp++){
    ob=&orb[isp];
    if(ob->t != idxt) continue;
    occu=ob->o;
    l=ob->p+ob->j;
    n=l+1+ob->n;
    degen=2*ob->j+2;
    ntot+=occu;
    printf("%2d %.12e   %2d %c %2d/2   %d %d %d %d %d :SPL\n"
   ,isp,ob->e-Mass[idxt], n, oam[l], 2*ob->j+1, ob->t, ob->p, ob->j, ob->n, occu);
    if(occu<0 || occu>degen) printf(" error in occupation number\n");
    if(occu>0 && ob->e >= Ebig) printf("error: undoube orbital is occupied\n");

  }
  printf("# total number of particles(t=%d)=%d\n",idxt,ntot);
}

//-----------------------------------------------------------------------------------------
int setConfiguration(){
  int idxt=ELECTRON,idxp,idxj,idxn,isp,n,l,degen,occu,nerr,ntot;
  char *oam = "spdfghijklmn"; // letters to represent orbital angular momenta

  Anum=8; // atomic number
  Mnum=16; // Anum*2; // mass number
  Ne[idxt]=Anum;// Anum-1
  Rnucl=1.2*pow((double) Mnum, 1.0/3.0); // nuclear radius (fm)
  printf("# Z=%d A=%d Ne=%d Rnucl=%.6e\n",Anum,Mnum,Ne[idxt],Rnucl);

  // Ne=8 system
  orb[ 0].o=2; // 1s  1/2 (p,j,n)=(0 0 0)  2=degeneracy
  orb[ 1].o=2; // 2s  1/2 (p,j,n)=(0 0 1)  2
  orb[ 2].o=2; // 2p  1/2 (p,j,n)=(1 0 0)  2
  orb[ 3].o=2; // 2p  3/2 (p,j,n)=(0 1 0)  4
  orb[ 4].o=0; // 3s  1/2 (p,j,n)=(0 0 2)  2
  orb[ 5].o=0; // 3p  1/2 (p,j,n)=(1 0 1)  2
```

```
orb[ 6].o=0; // 3p  3/2 (p,j,n)=(0 1 1)  4
orb[ 7].o=0; // 3d  3/2 (p,j,n)=(1 1 0)  4
orb[ 8].o=0; // 3d  5/2 (p,j,n)=(0 2 0)  6
orb[ 9].o=0; // 4s  1/2 (p,j,n)=(0 0 3)  2
orb[10].o=0; // 4p  1/2 (p,j,n)=(1 0 2)  2
orb[11].o=0; // 4p  3/2 (p,j,n)=(0 1 2)  4
orb[12].o=0; // 4d  3/2 (p,j,n)=(1 1 1)  4
orb[13].o=0; // 4d  5/2 (p,j,n)=(0 2 1)  6
orb[14].o=0; // 4f  5/2 (p,j,n)=(1 2 0)  6
orb[15].o=0; // 4f  7/2 (p,j,n)=(0 3 0)  8
orb[16].o=0; // 5s  1/2 (p,j,n)=(0 0 4)  2
orb[17].o=0; // 5p  1/2 (p,j,n)=(1 0 3)  2
orb[18].o=0; // 5p  3/2 (p,j,n)=(0 1 3)  4
orb[19].o=0; // 5d  3/2 (p,j,n)=(1 1 2)  4
orb[20].o=0; // 5d  5/2 (p,j,n)=(0 2 2)  6
orb[21].o=0; // 5f  5/2 (p,j,n)=(1 2 1)  6
orb[22].o=0; // 5f  7/2 (p,j,n)=(0 3 1)  8
orb[23].o=0; // 5g  7/2 (p,j,n)=(1 3 0)  8
orb[24].o=0; // 5g  9/2 (p,j,n)=(0 4 0) 10
orb[25].o=0; // 6s  1/2 (p,j,n)=(0 0 5)  2
orb[26].o=0; // 6p  1/2 (p,j,n)=(1 0 4)  2
orb[27].o=0; // 6p  3/2 (p,j,n)=(0 1 4)  4
orb[28].o=0; // 6d  3/2 (p,j,n)=(1 1 3)  4
orb[29].o=0; // 6d  5/2 (p,j,n)=(0 2 3)  6
orb[30].o=0; // 6f  5/2 (p,j,n)=(1 2 2)  6
orb[31].o=0; // 6f  7/2 (p,j,n)=(0 3 2)  8
orb[32].o=0; // 6g  7/2 (p,j,n)=(1 3 1)  8
orb[33].o=0; // 6g  9/2 (p,j,n)=(0 4 1) 10
orb[34].o=0; // 6h  9/2 (p,j,n)=(1 4 0) 10
orb[35].o=0; // 6h 11/2 (p,j,n)=(0 5 0) 12
orb[36].o=0; // 7s  1/2 (p,j,n)=(0 0 6)  2
orb[37].o=0; // 7p  1/2 (p,j,n)=(1 0 5)  2
orb[38].o=0; // 7p  3/2 (p,j,n)=(0 1 5)  4
orb[39].o=0; // 7d  3/2 (p,j,n)=(1 1 4)  4
orb[40].o=0; // 7d  5/2 (p,j,n)=(0 2 4)  6
orb[41].o=0; // 7f  5/2 (p,j,n)=(1 2 3)  6
orb[42].o=0; // 7f  7/2 (p,j,n)=(0 3 3)  8
orb[43].o=0; // 7g  7/2 (p,j,n)=(1 3 2)  8
orb[44].o=0; // 7g  9/2 (p,j,n)=(0 4 2) 10
orb[45].o=0; // 7h  9/2 (p,j,n)=(1 4 1) 10
orb[46].o=0; // 7h 11/2 (p,j,n)=(0 5 1) 12
orb[47].o=0; // 7i 11/2 (p,j,n)=(1 5 0) 12
orb[48].o=0; // 7i 13/2 (p,j,n)=(0 6 0) 14


// check and print

ntot=0;
nerr=0;
for(isp=0;isp<ispx;isp++){
  if(orb[isp].t == idxt && orb[isp].o > 0) {
    idxp=orb[isp].p;
    idxj=orb[isp].j;
    idxn=orb[isp].n;
    occu=orb[isp].o;
    l=idxp+idxj;
    n=l+1+idxn;
```

```
      degen=2*idxj+2;
      ntot+=occu;
      printf("(%2d) %d%c %2d/2 (t,p,j,n)=(%d %d %d %d) %2d %2d\n"
             ,isp,n,oam[l],2*idxj+1,orb[isp].t,idxp,idxj,idxn,degen,occu);
      if(degen<occu) {fprintf(stderr,"error : degen < occu\n"); nerr++;}
    }
  }
  printf("the number of electrons = %2d\n",ntot);
  if(ntot != Ne[idxt]) {fprintf(stderr,"error : ntot != Ne\n");nerr++;}
  if(nerr>0) exit(1);
}

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

  int i,idxt,idxp,idxj,idxn,idxr,idxrd,degeneracy,num,iter,isp;
  int imax=IDXP*IDXJ*NODE;
  int itermax=300;
  double E,sumspe[IDXT],edirect,eexch,etot;    // Energy [MeV], sum of single-particle energy
  double r,dr,drb,s; // radial coordinate (fm) and its grid spacing (drb=drb/2)
  int mprint;
  double diffVs[IDXT],diffVv[IDXT]; // maximum (for r) of the discrepancy in potentials
  double diff;
  double strVexch;
  FILE *FFD, *FBD, *FPO, *FWF, *FHI;
  double p, damp=0.25; // damping factor and its initial value
  const int debug=0;

  FFD=fopen("fdens.dat","w"); // Fermion (=lepton) densities
  FBD=fopen("bdens.dat","w"); // Boson densities
  FPO=fopen("pot.dat","w");   // Potential energies
  FWF=fopen("wf.dat","w");    // Nucleon wavefunctions
  FHI=fopen("hist.dat","w");  // Convergence history

  idxt=ELECTRON;

  strVexch = -fineStructureConstant*HbC*pow(3/M_PI,1.0/3.0);
  printf("# strength paremeter of the exchange potential =%.8e\n",strVexch);

  setRadialGrid(1);
  NuclearCoulombPotential();
  initialPotential(1);

  dr=RadialGridSpacing;
  drb=dr/2;
  if(debug) fprintf(FPO,"# iter %d (initial potential)\n",0);

  mprint=IDXRD/1000;
  if(mprint<1) mprint=1;

  if(debug){
    for(i=0;i<IDXRD;i+=mprint){
      fprintf(FPO,"%.8e %.7e %.7e %.7e %.7e\n",rx[i],Vv[idxt][i],Vcn[i],0.0,0.0);
    }
```

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

fprintf(FHI,"# iter Nerr etot sumspe edirect eexch damp potdiff\n");

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

  fprintf(FHI,"%d ",iter);

  for(idxt=0;idxt<IDXT;idxt++){
    prog2(idxt,orb,&sumspe[idxt]);
    if(debug) printf("iter=%d prog2 finished. sumspe(t=%d)=%e\n",iter,idxt,sumspe[idxt]);
  }

  printf("# J=idxj+0.5, L=idxj+idxp, varpi=2*idxp-1, kappa=varpi*(J+0.5)\n");
  for(isp=0;isp<ispx;isp++){
    if(orb[isp].o > 0 && orb[isp].e < Ebig){
 idxt=orb[isp].t;
       printf(" (%2d) (t,p,j,n,o)=%d %d %d %d %d E-m=%.12e\n"
              ,isp, idxt, orb[isp].p, orb[isp].j, orb[isp].n, orb[isp].o
              ,orb[isp].e-Mass[idxt]);
    }// end if
  }// end for(isp)

  if(debug){
  idxt=ELECTRON;
  for(idxrd=0;idxrd<IDXRD;idxrd+=mprint) fprintf(FFD,"%.8e %.7e\n",rx[idxrd],Dens[idxt][idxrd]);
  fprintf(FFD,"\n");
  } // end if

  // total number of particles

  idxt=ELECTRON;
  s=0;
  for(i=1;i<IDXRD;i++){
    r=rx[i];
    s+=r*r*Dens[idxt][i]*drdx[i];
  }
  s*=4*M_PI*drb;
  printf("# total number of particles(t=%d)=%.12e  error=%.5e\n",idxt,s,s-Ne[idxt]);

  fprintf(FHI," %.5e ",s-Ne[idxt]);

  // boson field

  idxt=ELECTRON;
  for(i=0;i<IDXRD;i++) source[idxt][i] = Dens[idxt][i]*Coupling[idxt];

  // PoissonRef(IDXRD,drb,rx,drdx,source[idxt],phi[idxt]);

  Poisson(IDXRD,drb,rx,drdx,source[idxt],phi[idxt]);


  if(debug){
    for(i=0;i<IDXRD;i+=mprint) fprintf(FBD,"%.8e %.7e\n",rx[i],phi[idxt][i]);
    fprintf(FBD,"\n");
  }// end if
```

```
  // total energy

  idxt=ELECTRON;
  edirect=0;
  eexch=0;
  for(i=1;i<IDXRD;i++){
    r=rx[i];
    edirect+=r*r*phi[idxt][i]*source[idxt][i]*drdx[i];
    eexch+=r*r*pow(Dens[idxt][i],4.0/3.0);
  }
  edirect*=4*M_PI*drb/2;
  eexch*=4*M_PI*drb*strVexch*3.0/4.0;
  etot=sumspe[idxt]-edirect-eexch/3;
  printf("# total energy(MeV)=%.12e SumSPE=%.12e ED=%.9e EX=%.9e\n"
    ,etot,sumspe[idxt],edirect,eexch);

  fprintf(FHI,"%.9e %.9e %.9e %.9e ",etot,sumspe[idxt],edirect,eexch);

  // new potential for the next iteration

  p=damp;
  if(iter > 30) p*=pow(0.98,iter-30);
  fprintf(FHI,"%.9e ",p);

  idxt=ELECTRON; diffVs[idxt]=0; diffVv[idxt]=0;
  if(debug) fprintf(FPO,"# iter %d\n",iter);
  for(i=0;i<IDXRD;i++){
    double Vcoul, Vexch, Vsnew[IDXT], Vvnew[IDXT];

    Vcoul = Coupling[idxt]*phi[idxt][i];
    Vexch = strVexch*pow(Dens[idxt][i],1.0/3.0);

    Vsnew[idxt]=0;
    Vvnew[idxt]=Vcoul+Vexch+Vcn[i];
    if(debug){
      if(i % mprint == 0) fprintf(FPO,"%.8e %.7e %.7e %.7e %.7e\n",rx[i],Vvnew[idxt],Vcn[i],Vcoul,Vexch);
    }
    diff=fabs(Vvnew[idxt]-Vv[idxt][i]); if(diffVv[idxt]<diff) diffVv[idxt]=diff;
    Vv[idxt][i]=(1-p)*Vv[idxt][i]+p*Vvnew[idxt];
  }
  if(debug) fprintf(FPO,"\n");

  diff  = diffVv[idxt] ;
  printf("# max potential inconsistency (MeV) %.5e iter=%d\n",diff,iter);

  fprintf(FHI,"%.5e\n",diff);

  if(diff<1.0e-12) { // =1.0e-6 eV
    printf("# selfconsistency fulfilled: potential difference<%.5e iter=%d\n",diff,iter);
    break;
  }

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

idxt=ELECTRON;
for(i=0;i<IDXRD;i+=mprint){double Vcoul, Vexch;
```

```
    Vcoul = Coupling[idxt]*phi[idxt][i];
    Vexch = strVexch*pow(Dens[idxt][i],1.0/3.0);
    fprintf(FFD,"%.8e %.7e\n",rx[i],Dens[idxt][i]);
    fprintf(FBD,"%.8e %.7e\n",rx[i],phi[idxt][i]);
    fprintf(FPO,"%.8e %.7e %.7e %.7e %.7e\n",rx[i],Vv[idxt][i],Vcn[i],Vcoul,Vexch);
  }

  // calculation of unoccupied orbitals

  for(isp=0;isp<ispx;isp++){int occu;
    occu=orb[isp].o;
    if(occu <= 0){
      idxp=orb[isp].p;
      idxj=orb[isp].j;
      idxn=orb[isp].n;
      idxtGlobal=orb[isp].t;
      solveDirac(idxp, idxj, idxn, Mass[idxt], &orb[isp].e, Fs[isp], Gs[isp]);
    } // end if
    if(orb[isp].e < Ebig)
      writeWavefunction(FWF, orb[isp].p, orb[isp].j, orb[isp].n, occu, Fs[isp], Gs[isp]);
  } // end for(isp)

  // sort and print information on the single-particle levels

  sort(orb);
  printOrbitals(ELECTRON);

  fclose(FFD); fclose(FBD); fclose(FPO); fclose(FWF); fclose(FHI);

  return 0;
}

//---------------------- calculation of single-particle eigensstates ----------------------
int prog2(int idxt, orbital *orb, double *sumspe){
  int idxp, idxj, idxn, occu, noccu, isp, idxr, idxrd;
  double r;
  const int debug=1;
  idxtGlobal=idxt;

  for(idxrd=0;idxrd<IDXRD;idxrd++) Dens[idxt][idxrd]=0.0;
  *sumspe=0; // sum of single-particle energies

  // calculation of the energies and wavefunctions of the eigenstates
  // for orbitals which are spedified to be occupied

  noccu=0;
  for(isp=0;isp<ispx;isp++){
    if(orb[isp].t == idxt && orb[isp].o > 0){
      idxp=orb[isp].p;
      idxj=orb[isp].j;
      idxn=orb[isp].n;
      occu=orb[isp].o;
      solveDirac(idxp, idxj, idxn, Mass[idxt], &orb[isp].e, Fs[isp], Gs[isp]);
      if(orb[isp].e>=Ebig){
fprintf(stderr,"solution unbound for (t,p,j,n)=%d %d %d %d\n",idxt,idxp,idxj,idxn);
        continue;
      }
```

```
      noccu+=occu;
      *sumspe+=occu*orb[isp].e;
      for(idxr=1;idxr<IDXR;idxr++){ double Ft,Gt;
        idxrd=2*idxr;
        r=rx[idxrd];
        Ft=Fs[isp][idxr];
        Gt=Gs[isp][idxr];
        Dens[idxt][idxrd]+=(Ft*Ft+Gt*Gt)*occu/(4.0*M_PI*r*r);
      }
    } // end if
  } // end for(isp)

  *sumspe-=noccu*Mass[idxt];

  // calculation of the densities
  double f6 =  4.0/3.0;
  double f7 = -1.0/3.0;
  Dens[idxt][0]=f6*Dens[idxt][2]+f7*Dens[idxt][4];
  interpolate(Dens[idxt],IDXRD);
  // fprintf(stderr,"Dens(r=0)=%e\n",Dens[idxt][0]);

}

//---------------------------------------------------------------------------
int writeWavefunction(FILE *FO, int idxp, int idxj, int idxn, int occu
     ,double *Fs, double *Gs)
{ // writes the wavefunction to a stream
  int i,ii;
  double r1, F, G;

  F=0;G=0;
  for(i=0;i<IDXR;i++){
    if(F<fabs(Fs[i]))F=fabs(Fs[i]);
    if(G<fabs(Gs[i]))G=fabs(Gs[i]);
  }
  fprintf(FO,"# (idxp,idxj,idxn,occu)=(%d %d %d %d)\n",idxp,idxj,idxn,occu);
  fprintf(FO,"# r(fm),F/%.12e,G/%.12e\n",F,G);
  F=1.0/F; G=1.0/G;
  ii=(IDXR-1)/100000;
  if(ii<1) ii=1;
  for(i=0;i<IDXR;i+=ii){
    fprintf(FO,"%.8e %.8e %.8e\n",rx[i*2],Fs[i]*F,Gs[i]*G);
  }
  fprintf(FO,"\n");
}

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

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
int sort(orbital *orb){
  int isp,isp2;
```

```
  for(isp=0;isp<ispx-1;isp++){
    for(isp2=isp+1;isp2<ispx;isp2++){
      if(orb[isp].t == orb[isp2].t && orb[isp].e > orb[isp2].e){
  swap(orb, isp, isp2);
      }
    }
  }
}

//-------------------------------------------------------------------------------------
int NuclearCoulombPotential() {
// (MeV)
  int i;
  double r,t,Vc0;
  // external variables : RadialGridSpacing, Rnucl, Anum, fineStructureConstant, HbC
  Vc0=-Anum*fineStructureConstant*HbC;
  for(i=0;i<IDXRD;i++){
    r=rx[i];
    if(r<=Rnucl){
      t=(1/Rnucl)*r;
      Vcn[i]=(0.5*Vc0/Rnucl)*(3-t*t);
    }
    else {
      Vcn[i]=Vc0/r;
    }
  }
  return 0;
}

//-------------------------------------------------------------------------------------
int initialPotential(int sw) {
// 0th compoment of vector potential (MeV) and scalar potential (MeV)
  int i,idxt;
  for(idxt=0;idxt<IDXT;idxt++){
    for(i=0;i<2*IDXR-1;i++){
      Vs[idxt][i]=0;
      Vv[idxt][i]=Vcn[i];
    }
  }
  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;
  }
}

double sfdrv(double x){
  //  return 1; // (0)
  return 1/sqrt(1 + x*x); // (1)
}

double sffor(double x){
  // return x; // (0)
  return log(x+ sqrt(1 + x*x)); // (1)
}

double sfbac(double x){
  double t;
  // return x; // (0)
  t=exp(x); return (t-1/t)*0.5; // (1)
}

//-------------------------------------------------------------------------------------------
int setRadialGrid(int sw)
{ // sets up the parameters of the radial grid to express the radial wavefunctions
  double dr, Rs,Rsinv;
  double x,xn,xout,dx,dxhf,rxn;
  int i;
  FILE *FGR;

  // used global variables : Rnucl

  if(0){
  for(i=0;i<30;i++){double x1,x1b,x2,r1,r2,t,tb;
    x1=i*0.1;       r1=sfbac(x1);  x1b=sffor(r1);
    x2=(i+1e-7)*0.1; r2=sfbac(x2);
    t=sfdrv((r1+r2)/2);
    tb=(x2-x1)/(r2-r1);
    printf("%e %e %e %e %e %e\n",r1,x1,x1b,x1b-x1,t,tb,tb-t);
  }
  }

  Idxr.o=IDXR-1; // For radial grid number >= Idxr.o, rungeBCo is used.
  Rout=30.0e+5;  // (fm) corresponding to Idxr.o
  Rs=0.25;  // the grid spacing begins to increase geometrically fore r>Rs

  Rsinv=1/Rs;
  xout=Rs*sffor(Rout*Rsinv);
  dx=xout/(IDXR-1);
  dxhf=dx*0.5;

  Idxr.i=Rs*sffor(2*Rs*Rsinv)/dx+0.5; // inner side connection at r=Rs*2=0.5fm
  if(Idxr.i<5) Idxr.i=5;
```

```
  rx[0]=0;
  drdx[0]=1;
  for(i=1;i<2*IDXR-1;i++){
    xn=(dxhf*Rsinv)*i;
    rxn=sfbac(xn);
    rx[i]=Rs*rxn;
    drdx[i]=1/sfdrv(rxn);
  }

  dr=dx;
  RadialGridSpacing=dr;
  Rin=rx[Idxr.i];

  //  Idxr.m=0.5e+5/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;

  FGR=fopen("grid.dat","w");
  for(i=0;i<2*IDXR-1;i++){
    fprintf(FGR,"%.5e %.5e %.5e\n",rx[i],i*dx,drdx[i]);
  }
  fclose(FGR);

  if(sw>0){
    printf("   R(in,out)=%.5e %.5e (fm)\n",Rin,Rout);
    printf("idxr(in,out)=%9d %9d d(xi)=%.5e (fm)\n",Idxr.i,Idxr.o,dr);
    printf(" r[2]-r[0]=%.5e r[2*IDXR-2]-r[2*IDXR-4]=%.5e (fm)\n"
    ,rx[2]-rx[0],rx[2*IDXR-2]-rx[2*IDXR-4]);
  }

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

}

//-------------------------------------------------------------------------------------
int solveDirac(int idxp,int idxj,int idxn, double mass, double *Esa, double *Fsa, double *Gsa)
{ // calculates the energy and the wavefunction of the eigenstate
  int i,idxt,node;
  double maco,maco1,maco2; // matching condition is "maco=0"
  double E,E0,E1,E2,dE, Eprec=1.0e-14; // energy (MeV)
  idxt = idxtGlobal;
  const int debug=0;

  E0=mass-mass*Eprec;
  runge(0,E0,idxp,idxj,idxn,mass,&node,&maco,Fsa,Gsa);

  if(debug) fprintf(stderr,"solveDirac: first call to 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=Ebigg;
```

```
    goto fin;
  }

  if(debug) fprintf(stderr,"For E-M=-0 eV, node=%d maco=%e\n",node,maco);

  dE=15.0e-6; E1=E0-dE;
  for(;;){
    runge(0,E1,idxp,idxj,idxn,mass,&node,&maco,Fsa,Gsa);

    if(debug) printf("solveDirac: #2# E1=%e dE=%e node=%d maco=%e\n"
                     ,E1-mass,dE,node,maco); // #2#

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

  E2=E1*0.8+E0*0.2;
  for(;;){
    runge(0,E2,idxp,idxj,idxn,mass,&node,&maco,Fsa,Gsa);
    if(node > idxn || node == idxn && maco > 0) break;
    E2=E2*0.8+E0*0.2;
    if(E2>E0) { // Actually,this condition was alread checked.
      fprintf(stderr,"no bound states(2): For E=mass, node=%d maco=%e\n",node,maco);
      E=Ebigg;
      goto fin;
    }
  }
  if(debug) fprintf(stderr,"upper bound (E,node,maco)=(%e %d %e)\n",E2,node,maco);

  // bisection method

  for(i=0;i<60;i++){
    if(E2-E1<=mass*Eprec) break;
    E=(E1+E2)*0.5;
    runge(0,E,idxp,idxj,idxn,mass,&node,&maco,Fsa,Gsa);
    if(debug) 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,mass,&node,&maco,Fsa,Gsa);

  *Esa=E;

  if(debug) fprintf(stderr,"iter=%d E=%f\n",i,*Esa);

 fin:
  *Esa=E;

  return 0;
}

//-------------------------------------------------------------------------------------
int rungeBCi(double r, double E, double mass, 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+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 mass, double *F, double *G){
// Boundary Condition for r -> infinity
  *F=exp(-r*sqrt(mass*mass-E*E)/HbC);
  *G=-sqrt((mass-E)/(mass+E))*exp(-r*sqrt(mass*mass-E*E)/HbC);
  return 0;
}

//-----------------------------------------------------------------------------------
#define RUNGE_OPTION 1

#if RUNGE_OPTION == 1
  // - - - - - - - - - - - - - - function version - - - - - - - - - - - - - -
  inline double rungeFun1(double rinv,double F,double G,double E,int kappa,double V,double mass,int i){
    return ((E-V+mass)*G*(1.0/HbC)-kappa*F*rinv)*drdx[i];
  }
  inline double rungeFun2(double rinv,double F,double G,double E,int kappa,double V,double mass,int i){
    return (-(E-V-mass)*F*(1.0/HbC)+kappa*G*rinv)*drdx[i];
  }
#elif RUNGE_OPTION == 2 // do not use
  // - - - - - - - - - - - - - - - - macro version - - - - - - - - - - - - - - -
  #define rungeFun1(rinv, F, G, E, kappa, V)\
    ((-(kappa))*(rinv)*(F)+(1.0/HbC)*((E)-(V)+mass)*(G))
  #define rungeFun2(rinv, F, G, E, kappa, V)\
    ((1.0/HbC)*(-((E)-(V)-mass))*(F)+(kappa)*(rinv)*(G))
#elif RUNGE_OPTION == 3 // do not use
  // - - - - - - - - - - - - - - - - inline version - - - - - - - - - - - - - - -
  //   Inline version uses neither functions nor macros.
#endif

int runge(int sw, double E, int idxp, int idxj, int idxn, double mass
        ,int *node, double*maco, double *Fsa, double *Gsa){
```

```
// 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
     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 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;
  const double debug=0;

  idxt=idxtGlobal;

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

  dr=RadialGridSpacing;

  if(sw & 0x01){
    for(i=0;i<=Idxr.i;i++){
      r1=rx[i*2];
      rungeBCi(r1,E,mass,idxp,L,&Fsa[i],&Gsa[i]);
    }
    for(i=IDXR-1;i>=Idxr.o;i--){
      r1=rx[i*2];
      rungeBCo(r1,E,mass,&Fsa[i],&Gsa[i]);
    }
  }

  *node=0;

  {double dE1,dE0=E-mass-Vv[idxt][Idxr.i*2]; double S,fct;
  for(i=Idxr.i*2+2;i<=Idxr.o*2;i+=2){
    dE1=E-mass-Vv[idxt][i];
    if(dE0 >0 && dE1 <0 ||dE0 <0 && dE1 >0 || dE1==0){
      Idxr.m=i/2;
      break;
    }
    dE0=dE1;
  }
  fct=sqrt(2*mass/(HbC*HbC));
  fct=log(1e16)/fct;
  S=0;
```

```
  Idxr.o=IDXR-1;
  for(;i<=Idxr.o*2;i+=2){
    dE1=mass+Vv[idxt][i]-E;
    if(dE1<0){
      S=0;
    }
    else{
      S+=sqrt(dE1)*(rx[i]-rx[i-2]);
    }
    if(S>fct){
      Idxr.o=i/2;
      break;
    }
  }
  if(Idxr.m <= Idxr.i) Idxr.m=Idxr.i+1;
  if(Idxr.m >= Idxr.o) Idxr.m=Idxr.o-1;
  }

  if(debug) fprintf(stderr,"Idxr.(m,o)=%d %d R.(m,o)=%e %e\n"
                           ,Idxr.m,Idxr.o,rx[Idxr.m*2],rx[Idxr.o*2]);

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

    if(forbac == 0){
      rungeBCi(rx[Idxr.i*2],E,mass,idxp,L,&F,&G);
      pmdr=dr; is=Idxr.i; ii=1;
    }
    else {
      rungeBCo(rx[Idxr.o*2],E,mass,&F,&G);
      pmdr=-dr; is=Idxr.o; ii=-1;
    }
    halfpmdr=pmdr*0.5;

    c1=pmdr*kappa;     // used only in inline version
    c2=pmdr*(1.0/HbC); // used only in inline version
    c3=E+mass;         // used only in inline version
    c4=E-mass;         // used only in inline version

    i2=is*2; r1=rx[i2]; vs=Vs[idxt][i2]; vv=Vv[idxt][i2]; r1inv=1.0/r1;
    c5=c1*r1inv; // used only in inline version
    for(i=is+ii ; ; i+=ii){
#if ( RUNGE_OPTION == 1 ) || ( RUNGE_OPTION == 2 )
  //---------------------- function and macro versions ----------------------
      k1=rungeFun1(r1inv,F,G,E,kappa,vv-vs,mass,i2)*pmdr;
      l1=rungeFun2(r1inv,F,G,E,kappa,vv+vs,mass,i2)*pmdr;
        i2=i*2;
        i3=i2-ii;
        r1=rx[i3]; r1inv=1.0/r1;
        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,mass,i3)*pmdr;
      l2=rungeFun2(r1inv,F2,G2,E,kappa,vv+vs,mass,i3)*pmdr;
        F3=F+0.5*k2; G3=G+0.5*l2;
      k3=rungeFun1(r1inv,F3,G3,E,kappa,vv-vs,mass,i3)*pmdr;
      l3=rungeFun2(r1inv,F3,G3,E,kappa,vv+vs,mass,i3)*pmdr;
        r1=rx[i2]; 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,mass,i2)*pmdr;
      l4=rungeFun2(r1inv,F4,G4,E,kappa,vv+vs,mass,i2)*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]);

  if(debug && *maco == 0.0){
    fprintf(stderr,"maco=0, %e %e %e %e\n",Fm[0],Gm[0],Fm[1],Gm[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++){
      i2=i*2;
      t1=Fsa[i];
      t2=Gsa[i];
```

```
      s1+=(t1*t1+t2*t2)*drdx[i2];
    }
    s2=0.0;
    for(i=Idxr.m;i<IDXR;i++){
      i2=i*2;
      t1=Fsa[i];
      t2=Gsa[i];
      s2+=(t1*t1+t2*t2)*drdx[i2];
    }
    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;i++){
      Fsa[i]*=nf2;
      Gsa[i]*=nf2;
    }
  }
  //---------------- normalization of the wavefunction : end ------------------

  return 0;

}

// block 2 : begin
//   This is an independently available part.
//  ================================================================================
//                     proper Poisson equation solver
//  ================================================================================

//------------------------- Poisson equation solver -------------------------
int PoissonRef(int imax,double h,double *rx,double *drdx,double *source,double *phi){
  double r0,r1;
  double I1,I2;
  double rhol;
  int i,j,k,l,al;
  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};
  int debug=1;

  if(debug==1) fprintf(stderr,"Poisson:begin\n");

  //r=0
  I2=0;
  for(j=0;j<imax-1;j++){
    for(k=0;k<6;k++){
      l=j+k-2;
      if(l>=0) r1=rx[l]; else r1=-rx[-l];
      al=abs(l);
      if(al<imax) I2+=intcoef[k]*source[al]*r1*drdx[al];
    }
```

```
  }
  phi[0]=h*I2;

  //r>0
  for(i=1;i<imax;i++){
    r0=rx[i];
    I1=0;
    for(j=0;j<i;j++){
      for(k=0;k<6;k++){
        l=j+k-2;
        if(l>=0) r1=rx[l]; else r1=-rx[-l];
        al=abs(l);
        if(al<imax) I1+=intcoef[k]*source[al]*r1*r1*drdx[al];
      }
    }
    I1/=r0;
    I2=0;
    for(k=0;k<6;k++){
      for(j=i;j<imax-1;j++){
        l=j+k-2;
        if(l>=0) r1=rx[l]; else r1=-rx[-l];
        al=abs(l);
        if(al<imax) I2+=intcoef[k]*source[al]*r1*drdx[al];
      }
    }
    phi[i]=h*(I1+I2);
  }
  if(debug==1) fprintf(stderr,"Poisson:end\n");
  return 0;
}

//------------------------- Poisson equation solver --------------------------
int Poisson(int imax,double dx, double *rx, double *drdx, double *source,double *phi){
  double r1;
  double I1,I2,I1b,I2b;
  double rhol;
  int i,j,k,l,al;
  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};
  double bdrycoef[5];
  int debug=0;

  if(debug==1) fprintf(stderr,"Poisson(ver.3):begin\n");

  bdrycoef[0]=intcoef[0];
  for(i=1;i<5;i++) bdrycoef[i]=bdrycoef[i-1]+intcoef[i];

  if(debug){
    for(i=0;i<5;i++) printf("bdrycoef[%d]=%f/1440\n",i,bdrycoef[i]*1440);
    if(fabs(bdrycoef[5]-1)<1.0e-13) fprintf(stderr,"error in integral coef %.15e\n",bdrycoef[5]);
  }

  i=0; phi[i]=0;
  for(i=1;i<6;i++){
    I1=0;
    for(j=0;j<i;j++){
      for(k=0;k<6;k++){
```

```
      l=j+k-2;
      if(l>=0) r1=rx[l]; else r1=-rx[-l];
      al=abs(l);
      if(al<imax) I1+=intcoef[k]*source[al]*r1*r1*drdx[al];
    }
  }
  phi[i]=dx*I1/rx[i];
}
i=6;
  I1b=bdrycoef[0]*rx[2]  *rx[2]  *source[2]*drdx[2]
     +bdrycoef[1]*rx[1]  *rx[1]  *source[1]*drdx[1]
     +bdrycoef[2]*rx[0]  *rx[0]  *source[0]*drdx[0]
     +bdrycoef[3]*rx[1]  *rx[1]  *source[1]*drdx[1]
     +bdrycoef[4]*rx[2]  *rx[2]  *source[2]*drdx[2]
     +            rx[3]  *rx[3]  *source[3]*drdx[3];
  I1=I1b
    +bdrycoef[4]*rx[i-2]*rx[i-2]*source[i-2]*drdx[i-2]
    +bdrycoef[3]*rx[i-1]*rx[i-1]*source[i-1]*drdx[i-1]
    +bdrycoef[2]*rx[i  ]*rx[i  ]*source[i  ]*drdx[i  ];
  if(i+1<imax) I1+=bdrycoef[1]*rx[i+1]*rx[i+1]*source[i+1]*drdx[i+1];
  if(i+2<imax) I1+=bdrycoef[0]*rx[i+2]*rx[i+2]*source[i+2]*drdx[i+2];
  phi[i]=dx*I1/rx[i];
for(i=7;i<imax-2;i++){
  I1b+=          rx[i-3]*rx[i-3]*source[i-3]*drdx[i-3];
  I1=I1b
    +bdrycoef[4]*rx[i-2]*rx[i-2]*source[i-2]*drdx[i-2]
    +bdrycoef[3]*rx[i-1]*rx[i-1]*source[i-1]*drdx[i-1]
    +bdrycoef[2]*rx[i  ]*rx[i  ]*source[i  ]*drdx[i  ]
    +bdrycoef[1]*rx[i+1]*rx[i+1]*source[i+1]*drdx[i+1]
    +bdrycoef[0]*rx[i+2]*rx[i+2]*source[i+2]*drdx[i+2];
  phi[i]=dx*I1/rx[i];
}
for(i=imax-2;i<imax;i++){
  I1b+=          rx[i-3]*rx[i-3]*source[i-3]*drdx[i-3];
  I1=I1b
    +bdrycoef[4]*rx[i-2]*rx[i-2]*source[i-2]*drdx[i-2]
    +bdrycoef[3]*rx[i-1]*rx[i-1]*source[i-1]*drdx[i-1]
    +bdrycoef[2]*rx[i  ]*rx[i  ]*source[i  ]*drdx[i  ];
  if(i+1<imax) I1+=bdrycoef[1]*rx[i+1]*rx[i+1]*source[i+1]*drdx[i+1];
  if(i+2<imax) I1+=bdrycoef[0]*rx[i+2]*rx[i+2]*source[i+2]*drdx[i+2];
  phi[i]=dx*I1/rx[i];
}

for(i=imax-1;i>imax-7;i--){
  I2=0;
  for(k=0;k<6;k++){
    for(j=i;j<imax-1;j++){
      l=j+k-2;
      if(l>=0) r1=rx[l]; else r1=-rx[-l];
      al=abs(l);
      if(al<imax) I2+=intcoef[k]*source[al]*r1*drdx[al];
    }
  }
  phi[i]+=dx*I2;
}
i=imax-7;
  I2b=bdrycoef[2]*rx[imax-1]*source[imax-1]*drdx[imax-1]
```

```
     +bdrycoef[3]*rx[imax-2]*source[imax-2]*drdx[imax-2]
     +bdrycoef[4]*rx[imax-3]*source[imax-3]*drdx[imax-3]
     +            rx[imax-4]*source[imax-4]*drdx[imax-4];
  I2=I2b
    +bdrycoef[4]*rx[i+2]*source[i+2]*drdx[i+2]
    +bdrycoef[3]*rx[i+1]*source[i+1]*drdx[i+1]
    +bdrycoef[2]*rx[i  ]*source[i  ]*drdx[i  ];

  j=i-1;
    if(j<0)
      I2+=bdrycoef[1]*(-rx[-j])*source[-j]*drdx[-j];
    else
      I2+=bdrycoef[1]*rx[j]*source[j]*drdx[j];

  j=i-2;
    if(j<0)
      I2+=bdrycoef[0]*(-rx[-j])*source[-j]*drdx[-j];
    else
      I2+=bdrycoef[0]*rx[j]*source[j]*drdx[j];

for(i=imax-8;i>1;i--){
  I2b+=rx[i+3]*source[i+3]*drdx[i+3];
  I2=I2b
    +bdrycoef[4]*rx[i+2]*source[i+2]*drdx[i+2]
    +bdrycoef[3]*rx[i+1]*source[i+1]*drdx[i+1]
    +bdrycoef[2]*rx[i  ]*source[i  ]*drdx[i  ];

  j=i-1;
    if(j<0)
      I2+=bdrycoef[1]*(-rx[-j])*source[-j]*drdx[-j];
    else
      I2+=bdrycoef[1]*rx[j]*source[j]*drdx[j];

  j=i-2;
    if(j<0)
      I2+=bdrycoef[0]*(-rx[-j])*source[-j]*drdx[-j];
    else
      I2+=bdrycoef[0]*rx[j]*source[j]*drdx[j];
  phi[i]+=dx*I2;
}

for(i=1;i>=0;i--){
  I2b+=rx[i+3]*source[i+3]*drdx[i+3];
  I2=I2b
    +bdrycoef[4]*rx[i+2]*source[i+2]*drdx[i+2]
    +bdrycoef[3]*rx[i+1]*source[i+1]*drdx[i+1]
    +bdrycoef[2]*rx[i  ]*source[i  ]*drdx[i  ];

  j=i-1;
    if(j<0)
      I2+=bdrycoef[1]*(-rx[-j])*source[-j]*drdx[-j];
    else
      I2+=bdrycoef[1]*rx[j]*source[j]*drdx[j];

  j=i-2;
      I2+=bdrycoef[0]*(-rx[-j])*source[-j]*drdx[-j];
```

```
   phi[i]+=dx*I2;
  }

  if(debug==1) fprintf(stderr,"Poisson(ver.3):end\n");
  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){
    fprintf(stderr,"interpolate: argument n =%d : abort\n",n);
    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
```