モンテカルロ積分による
二体相互作用行列要素の計算

2003年2月7日

福井大学 工学部 物理工学科
11年度入学 9番 稲垣 安章

目 次

第 1 章　核力と核子間の相関 **5**

1.1 核力 5
1.2 核力の周辺部分を表す湯川型ポテンシャル 5
1.3 核力ポテンシャルの斥力芯 7
1.4 配位混合と有効相互作用 8

第 2 章　原子核中の陽子・中性子の波動関数 **9**

2.1 Woods-Saxon ポテンシャル 9
2.2 角運動量の合成 9
2.3 $l \cdot s$ポテンシャル 10
2.4 アイソスピン 11
2.5 Schrödinger 方程式 11
2.6 1 粒子状態の波動関数 13

第 3 章　モンテカルロ法 **18**

3.1 モンテカルロ積分 18
3.2 加重サンプリング (Importance sampling) 20
3.3 メトロポリス法 21
3.4 モンテカルロ積分のテスト計算 22

第 4 章　二体相互作用行列要素の計算 **26**

4.1 演算子の行列要素 26
4.2 2 体相互作用行列要素 26
4.3 有効相互作用 Gogny 27
4.4 核子間 2 体相互作用行列要素の計算 28

結論 **32**

謝辞 **33**

付録 Program List **35**

序章

モンテカルロ法は乱数を利用した計算法の総称であり、コンピュータに適した算法として幅広い分野で利用されている。そのひとつにモンテカルロ積分がある。これは乱数を用いて積分の値を計算することをさす。

モンテカルロ積分の計算手続きはきわめて単純である。そこでは積分を「被積分関数の期待値」×「積分領域の体積」としてとらえなおす。そして期待値を求めるのに、積分領域内にランダムに分布する標本点を利用するのである。期待値と同時にその誤差の大きさも統計的手法により評価することができる。このように手続きが単純であるため、他の手法での計算結果の確認のための比較用計算が用途のひとつとして挙げられる。

本研究では、原子核中の２個の核子の相互作用行列要素の計算にモンテカルロ積分を適用する。２個の粒子が相互作用してそれぞれ別の状態に移行する量子力学的遷移振幅は、それぞれの粒子に位置座標が３個づつあるため、合計６次元空間での積分により表される。後述のとおりモンテカルロ積分は高次元領域での積分に適した算法であるため、その計算に適していると思われる。

なお、この種の行列要素の計算には、より高速でより精度が高い方法が存在するが、それらの方法は複雑なため定式化やプログラミングで誤りをおかしやすい。本研究の結果は将来、そのような複雑な方法による計算プログラムの検証に役立つであろう。

本論文の構成は以下の通りである。

第１章で核力の特徴について概観したあと、第２章では、原子核中の核子の波動関数を求めるため、球対称一体ポテンシャル中の固有状態がどのような量子数でラベルづけされ、その動径波動関数をどのようにして求めるかを説明する。付録１に

は、ひとつの原子核中の１陽子および１中性子の固有状態をもれなく見つけてそのエネルギー準位と動径波動関数をメモリー上に整理して記憶しておくために本研究で作成したのＣ言語のプログラムのソースリストを載せた。

第３章ではモンテカルロ積分の一般論を述べたあと、テスト計算として、解析的に計算できる６次元積分をモンテカルロ法で計算し、各種の工夫がどのように計算効率に影響するかを調べる。

第４章では、モンテカルロ積分をニ体相互作用行列要素の計算に適用する。計算に用いた相互作用の Gogny 力について説明したあと、それに第２章で求めた動径波動関数、球面調和関数、Clebsh-Gordan 係数などを掛けあわせて定義される被積分関数の形を正確に述べる。その被積分関数をモンテカルロ積分した結果の例を示し、被積分関数にできるだけ類似した確率密度分布を持つ標本点の選び方をすることで、少ない標本点数で高い精度が得られることを示す。

序章を終えるまえに、本文中では述べることのできなかったニ体相互作用行列要素を計算する意義について述べておこう。系を構成している粒子の運動を考えるとき、それが他の粒子の状態と無関係に記述できるならば、その粒子は他粒子と相関がないと言う。粒子間に相互作用が働いている系では粒子相関があるのが普通である。直接に相互作用が無くても、Pauli の原理のように互いに他の運動を制約する法則があるときには、やはり相関が現れる。相互作用のある系では、ごく一般的にいえば、すべての粒子は互いに相関を持っているから、系が N 個の粒子から成り立っていれば N 粒子相関まで考えなければならない。３体以上の相関には３体力が関与しうるが、原子核では多体力の効果はニ体相互作用の密度依存性として取り入れることが多い。このため、相関のもととなる残留相互作用の効果は、二体相互作用行列要素にほとんどすべて取り入れられていると考えるのが普通である。このニ体行列要素を用いてどのように核子間相関を導き出して行くかは次の段階の問題であり、例えば殻模型の配位混合空間においては、対角化計算から量子モンテカルロ計算にわたる幅広いスペクトルの各種アプローチがあり、あるいは時間依存 Hartree-Fock-Bogoliubov 理論のような全く異なる取り組み方がある。様々な手法の競合により今後の進展が期待される。

第1章　核力と核子間の相関

1.1　核力

二つの核子間で中間子をやりとりすることによって、それらの二つの核子間には核力という、複雑な力が働く。この機構は湯川によって理論的に提唱されたものであるが、今では、パイ中間子を始めとして多くの中間子が見つかっている。また核力は核内での二核子間の距離を、多少の重なりを考慮にいれ、2fm 程度とし、核子をその距離に保ちつつ束縛する。

またその特徴として、強い相互作用であることと、短距離力であることががあげられる。また基本的には核力は引力として考られ、その引力ポテンシャルの到達距離はパイ中間子のド・ブロイ波長程度であり、1fm(10^{-15}) くらいである。つまり 1fm を大きくこえたところには核力は及ばない。

また次に斥力芯が核力の特徴としてあげられ、2 個の核子が接近していっても、ある程度以上には近づけなくなる。この性質は、2 核子間の距離が非常に小さいときのポテンシャルを無限に高くすることによって表される。距離の小さい所で現われるこの強い斥力部分をさして、斥力芯 (core) と呼ぶ。

1.2　核力の周辺部分を表す湯川型ポテンシャル

パイ中間子 (質量 m_π) が一時的に生じて二つの核子の間に交換されるときに走れる距離は、

$$\frac{\hbar}{m_a c} \equiv \frac{1}{\mu_a} \tag{1.1}$$

である。これはちょうど、パイ中間子のコンプトン波長に相当する。この核力の到達距離は、不確定性関係から、つぎのように理解できる。今、r_1 にある核子1が中間子を放出し、r_2 にある核子2がそれを吸収する過程を考える。パイ中間子の放出に伴うエネルギーの不確定の程度は $m_\pi c^2$ で、対応する時間の程度は $\Delta t \sim \hbar/m_\pi c^2$ である。中間子の伝播の早さは光速 c 以下だから、パイ中間子が核子1から離れる距離は $c\Delta t \sim \hbar/m_\pi c$ である。その範囲に核子2があれば、中間子を吸収して、核力が生じる。こうしてパイ中間子の交換が有効に起こる距離の上限の程度はおよそ 1.4fm といえる。ここで原点に e なる点電荷があるときのクーロンポテンシャル $\phi(\boldsymbol{r})$ は、

$$\nabla^2\phi(\boldsymbol{r}) = -4\pi e\delta(\boldsymbol{r}) \tag{1.2}$$

を満たす。ここで中間子 m_π が生み出すポテンシャルは

$$\left\{\nabla^2 - \left(\frac{m_a c}{\hbar}\right)^2\right\}\phi(\boldsymbol{r}) = 4\pi f\delta(\boldsymbol{r}) \tag{1.3}$$

と書ける。ここで f は核子と中間子 m_a の結合の強さを表す定数である。この方程式 (1.3) の解は

$$\phi(r) = -f\frac{\exp(r/r_\pi)}{r} \tag{1.4}$$

になる。2陽子間のクーロンポテンシャルエネルギーが $e\phi(r)$ であるように、2核子間の中間子交換力は

$$V(r) = f\phi(r) = -f^2\frac{\exp(r/r_\pi)}{r} \tag{1.5}$$

と書ける。この e^{-x}/x の形を湯川型ポテンシャルいう [1]。図1を見ると核子間距離は r >1.4fm では核力は急激に弱くなる。また強さは r=0.7fm でクーロン力の少なくとも6倍強い。ところが、r=3.5fm ではクーロン力の3倍くらい、r=5.6fm では同じくらいとなる。このように短距離力であることと、強いことが核力の特徴である。それに反して、クーロン力は r が大きくなっても0に近付くだけである。実際の核力はもっと複雑で、2個のパイ中間子を同時に交換したり、パイ中間子より重い質量の中間子を交換して生じる。しかし、まだ $r \leq$1fm の核力は確定していない。

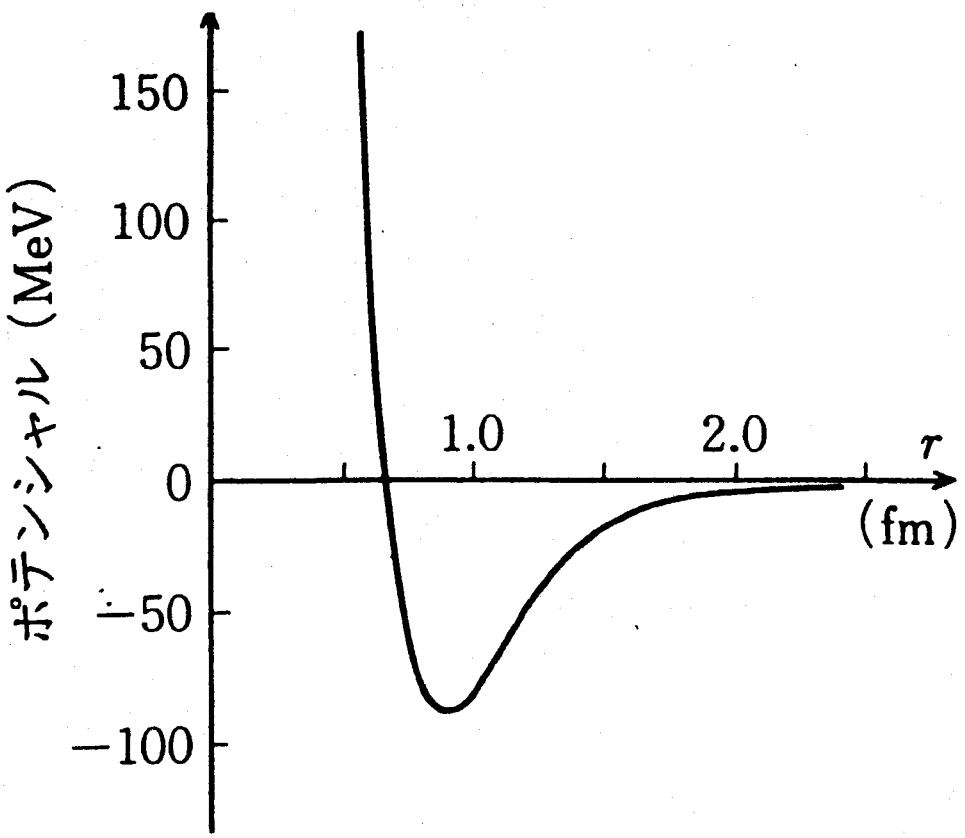

図 1. 核力の距離依存性 [2]

1.3 核力ポテンシャルの斥力芯

核子間に働く核力ポテンシャルは図 1 に示されるとおり、近距離 (r<1fm) では核子の内部構造とそれを支配する相互作用の問題と密接に関連している。とくに、r<0.5fm には非常に強い斥力ポテンシャルが存在する。これを斥力芯 (hard core) という。ここでは中間子交換による核力の記述はもはや有効性を持たないと考えられる。

1.4 配位混合と有効相互作用

原子核の殻模型では、どの軌道に何個の粒子が入っているかを指定して状態を決める。このような粒子の配置のしかたを配位という。通常は1粒子エネルギーの低い準位から順番を粒子に詰めることによって、低いエネルギーの状態が得られるので、エネルギーが低い軌道では閉殻が形成され、フェルミ準位付近の閉殻外の軌道への粒子配置を「配位」と呼ぶことが多い。そこで1粒子準位の間隔が大きい場合には最低のエネルギーのみ考えれば十分である。接近した1粒子準位を問題とする場合、それらの準位への粒子の詰めかたは一通りではなくなり、いくつかの配位が可能となるが、残留相互作用まで考慮したハミルトニアンの固有状態としていくつかの配位を含んだ状態を得る。その状態を配位混合という。配位混合を考えることによって、エネルギー準位などの原子核の性質をうまく説明できる例が多数ある。一方、ほかからかなり孤立した1粒子準位に粒子が入っているときなどのように、一意的に主な配位が指定できるときでも、配位混合の影響が大きい場合がある。それは1粒子エネルギーの差が大きく、残留相互作用によるほか配位の混合は小さくても、物理量のほかの配位を含む行列要素(電磁気モーメントなどの)が非常に大きいときは、この物理量を求めるときには配位混合を無視できないからである。

第2章　原子核中の陽子・中性子の波動関数

2.1　Woods-Saxon ポテンシャル

核力のポテンシャルの性質としてまず原子核の内部では密度の飽和性により核子の感じるポテンシャルは一定であることがあげられる。さらに r が表面に近付くにつれて核子の感じるポテンシャルは浅くなる。そして原子核の表面の外にでると、核力の到達距離を越えたところで 0 になるということがある。このポテンシャルは自分以外の他の核子からの効果をならしてとりいれているということで平均ポテンシャルという [3]。

このポテンシャルをモデル化したものとして井戸型ポテンシャルがあげられるがこれは核力が距離によって連続的に減少するという点が無視されているので、この点を改善した下式の表す Woods-Saxon ポテンシャルを本研究では用いることにする。

$$V(r) = -V_0 \frac{1}{1 + \exp\left\{-\left(r - R_0\right)/a\right\}} \tag{2.1}$$

ここで、V_0 はポテンシャルの深さを表し、R_0 はポテンシャルの半径パラメータ、a はポテンシャルのぼやけのパラメータと呼ばれる。

2.2　角運動量の合成

一般的に 2 つの角運動量 $\boldsymbol{l_1}$ と $\boldsymbol{l_2}$ を考える。それらの角運動量の和 $\boldsymbol{l}$ は

$$\boldsymbol{l} = \boldsymbol{l_1} + \boldsymbol{l_2} \tag{2.2}$$

である。古典力学であれば $\boldsymbol{l}$ は、l_1, l_2 から $|l_1 - l_2|$ までの実数値を自由にとれるのであるが、量子力学では様子が少し違う。合成した角運動量は

$$l = l_1 + l_2, l_1 + l_2 - 1, l_1 + l_2 - 2, \cdots, |l_1 - l_2| \tag{2.3}$$

というとびとびの値しかとれない。2つのスピン角運動量についても同じで、スピンの合成の結果は

$$S = \frac{1}{2} + \frac{1}{2}, \frac{1}{2} + \frac{1}{2} - 1, |\frac{1}{2} + \frac{1}{2}| \tag{2.4}$$

すなわち

$$S = 1, 0 \tag{2.5}$$

の2つの値をとる。

1個の核子の軌道角運動量lとスピン角運動量sの和はその核子の全角運動量であり記号j であらわす。

$$j = l + s \tag{2.6}$$

で与えられ、ふたたび (2.3) 式の場合のように

$$j = l + s, l + s - 1, \cdots, |l - s| \tag{2.7}$$

というとびとびの値をとる。$s = \frac{1}{2}$ であるから $l > 0$ なら

$$j = l + \frac{1}{2} \quad , l - \frac{1}{2} \tag{2.8}$$

の2つの値しかとれないことになる。$l = 0$ なら $j = \frac{1}{2}$ の1つの値しかとれない。

2.3 $l \cdot s$ポテンシャル

1つの核子の軌道角運動量lとスピン角運動量sの和はjは (2.8) 式のように1〜2個の値しかとれない、jの2乗を計算すると

$$\boldsymbol{j}^2 = \boldsymbol{l}^2 + \boldsymbol{s}^2 + 2\boldsymbol{l}\cdot\boldsymbol{s} \tag{2.9}$$

である。したがって

$$\boldsymbol{l}\cdot\boldsymbol{s} = \frac{1}{2}\left\{j(j+1) - l(l+1) - \frac{1}{2}\left(\frac{1}{2}+1\right)\right\} \qquad = \begin{cases} \frac{1}{2}l & (j = l + \frac{1}{2}) \text{のとき} \\ -\frac{1}{2}(l+1) & (j = l - \frac{1}{2}) \text{のとき} \end{cases} \tag{2.10}$$

である。

2.4 アイソスピン

中性子と陽子のそれぞれの質量 [4] は

$$\begin{aligned} M_p &= 1.67239 \times 10^{-27}(\mathrm{kg}) \\ M_n &= 1.67470 \times 10^{-27}(\mathrm{kg}) \end{aligned} \tag{2.11}$$

であり、その質量差はほとんどなく電荷$+e$をもっているかいないかの違いとなる。[4] そこで陽子と中性子とを核子という粒子を2つの違った状態とみなすと、核状態を分類したり、状態間の遷移を計算するときに便利な量子数となる。そのため核子の状態を表す波動関数を

$$\psi = \phi_{\text{空間}}(\boldsymbol{r})\phi^{\sigma}_{\text{スピン}}\phi^{\tau}_{\text{アイソスピン}} \tag{2.12}$$

と書き表す。

ここで$\phi^{\tau}_{\text{アイソスピン}}$は核子の2つの荷電状態に対応した波動関数を表し、スピン波動関数$\phi^{\sigma}_{\text{スピン}}$と同様に扱う。すなわち、スピン1/2の粒子では、そのz成分(+1/2)と(-1/2)とに対応した2つの状態があるが、これにならって新たにアイソスピンτを導入し、アイソスピンが上向き(+1/2)の状態は中性子、下向き(-1/2)の状態は陽子を表すものと定義する。[1] このように抽象的に導入された荷電空間でのスピンをアイソスピンとよぶ。

2.5 Schrödinger方程式

3次元空間中の1粒子のSchrödinger方程式は

$$\left\{-\frac{\hbar^2}{2m}\boldsymbol{\nabla}^2 + V(\boldsymbol{r},\sigma) - E\right\}\psi(\boldsymbol{r},\sigma) = 0 \tag{2.13}$$

[1] 原子核では、安定核の構成がN≳Zと中性子過剰側にあるので、中性子を+1/2とするのが便利である。ところが素粒子物理では+1/2を陽子、-1/2を中性子と定義するのが通例である。

である。またそのポテンシャルはWoods-Saxon型を用い

$$V(r) = V_{ws}F_{Rv,Ra}(r) + V_{LS}\ \boldsymbol{l}\cdot\boldsymbol{s}\ r_{LS}^2 \times a_{LS}^{-1}\frac{1}{r}\frac{d}{dr}F_{RLS,aLS}(r) + V_c(r) \tag{2.14}$$

と表し、またクーロンポテンシャル$V_c(r)$は

$$V_c(r) = \frac{(Z-1)\ e^2}{R_c}\ \frac{3-(\frac{r}{R_c})^2}{2}\ \Theta(R_c - r) + \frac{(Z-1)\ e^2}{r}\ \Theta(r - R_c) \tag{2.15}$$

となる。ここで$\Theta(R_c - r)$は階段関数で

$$\Theta(R_c - r) = \begin{cases} 1 & (R_c > r) \\ 0 & (R_c < r) \end{cases} \tag{2.16}$$

である。すなわち(2.15)の第1項は原子核中で働くポテンシャルを表し、第2項は原子核の外で働くポテンシャルを表す。

また本研究では以下のパラメーター[5]を用いた。

$$\cdot V_{ws} = -51 + 132t_z\frac{N-Z}{2A}(\mathrm{MeV}) \qquad t_z = \begin{cases} +\frac{1}{2} & : \text{中性子} \\ -\frac{1}{2} & : \text{陽子} \end{cases}$$

$$\cdot V_{LS} = -0.44\ V_{ws} \qquad \cdot r_v = r_{LS} = r_c = 1.27(\mathrm{fm})$$

$$\cdot a_v = a_{LS} = 0.67(\mathrm{fm}) \tag{2.17}$$

また計算は以下の定数を用いた。

$$\cdot e^2 = 1.439965 \quad (\mathrm{MeV}\cdot\mathrm{fm}) \qquad \cdot\frac{\hbar^2}{2m} = 20.7355 \quad (\mathrm{MeV}\cdot\mathrm{fm}^2) \tag{2.18}$$

2.6 1粒子状態の波動関数

1粒子状態の波動関数はスピン－軌道結合ポテンシャルのため$\boldsymbol{l}$のz成分m_lも$\boldsymbol{s}$のz成分m_sもよい量子数ではなくなるが、$\boldsymbol{j}=\boldsymbol{l}+\boldsymbol{s}$の大きさ$j$とそのz成分$m$はよい量子数である。したがって、この1粒子状態の波動関数は

$$\psi_{nljmq}(\boldsymbol{r},\sigma,\tau)=R_{nljq}(r)\,\mathcal{Y}_{ljm}(\theta,\varphi,\sigma)\,K(\tau) \tag{2.19}$$

と表される。

また動径波動関数を

$$R_{nljq}(r)=\frac{u_{nljq}(r)}{r} \tag{2.20}$$

とかくと$u(r)$の従う微分方程式は

$$\frac{d^2}{dr^2}u(r)=f(r)u(r)\,, \tag{2.21}$$

$$f(r)=\frac{2m}{\hbar^2}V(r)\quad+\quad\frac{l(l+1)}{r^2}-\varepsilon \tag{2.22}$$

である。また動径波動関数の規格化は

$$\begin{cases}\int_0^{R_{\max}}|u_{nljq}(r)|^2\,dr\;\;=1\\ \int_0^{R_{\max}}|R_{nljq}(r)|^2\;\;r^2\;dr\;\;=1\end{cases} \tag{2.23}$$

である。また角度部分は2つの球面調和関数で表され

$$\begin{aligned}\mathcal{Y}_{ljm}(\theta,\,\varphi,\,\sigma)\;\;&=\;\;<l,m-\frac{1}{2},\frac{1}{2},\frac{1}{2}|j,m>Y_l^{m-\frac{1}{2}}(\theta,\varphi)\;X_{\frac{1}{2}}(\sigma)\\ &+\;\;<l,m+\frac{1}{2},\frac{1}{2},-\frac{1}{2}|j,m>Y_l^{m+\frac{1}{2}}(\theta,\varphi)\;X_{-\frac{1}{2}}(\sigma)\end{aligned} \tag{2.24}$$

となる。

また Clebsch-Gordan 係数は表 2.1 に与えた。$j = l \pm \frac{1}{2}$ と $m_s = \pm\frac{1}{2}$ の 4 つの場合が必要である。

表 2.1: Clebsh-Gordan 係数 $\langle l, m - m_s, \frac{1}{2}, m_s | j, m \rangle$ [6]

	$m_s = \frac{1}{2}$	$m_s = -\frac{1}{2}$
$j = l + \frac{1}{2}$	$\sqrt{\frac{l+\frac{1}{2}+m}{2l+1}}$	$\sqrt{\frac{l+\frac{1}{2}-m}{2l+1}}$
$j = l - \frac{1}{2}$	$-\sqrt{\frac{l+\frac{1}{2}-m}{2l+1}}$	$\sqrt{\frac{l+\frac{1}{2}+m}{2l+1}}$

s の z 成分 m_s の波動関数部分は下記の通りである。

$$\begin{cases} X_{\frac{1}{2}}\left(\frac{1}{2}\right) = 1 \quad , & X_{-\frac{1}{2}}\left(\frac{1}{2}\right) = 0 \\ X_{\frac{1}{2}}\left(-\frac{1}{2}\right) = 0 \quad , & X_{-\frac{1}{2}}\left(-\frac{1}{2}\right) = 1 \end{cases} \tag{2.25}$$

次ページにの図2、次々ページの図3に、以上のパラメータを用いて付録1のプログラムで計算した中性子と陽子のポテンシャルとそれぞれの1粒子状態のエネルギー固有値を書き込んだものを示す。また本研究で原子核の種類は $^{100}_{50}Sn_{50}$ とした。

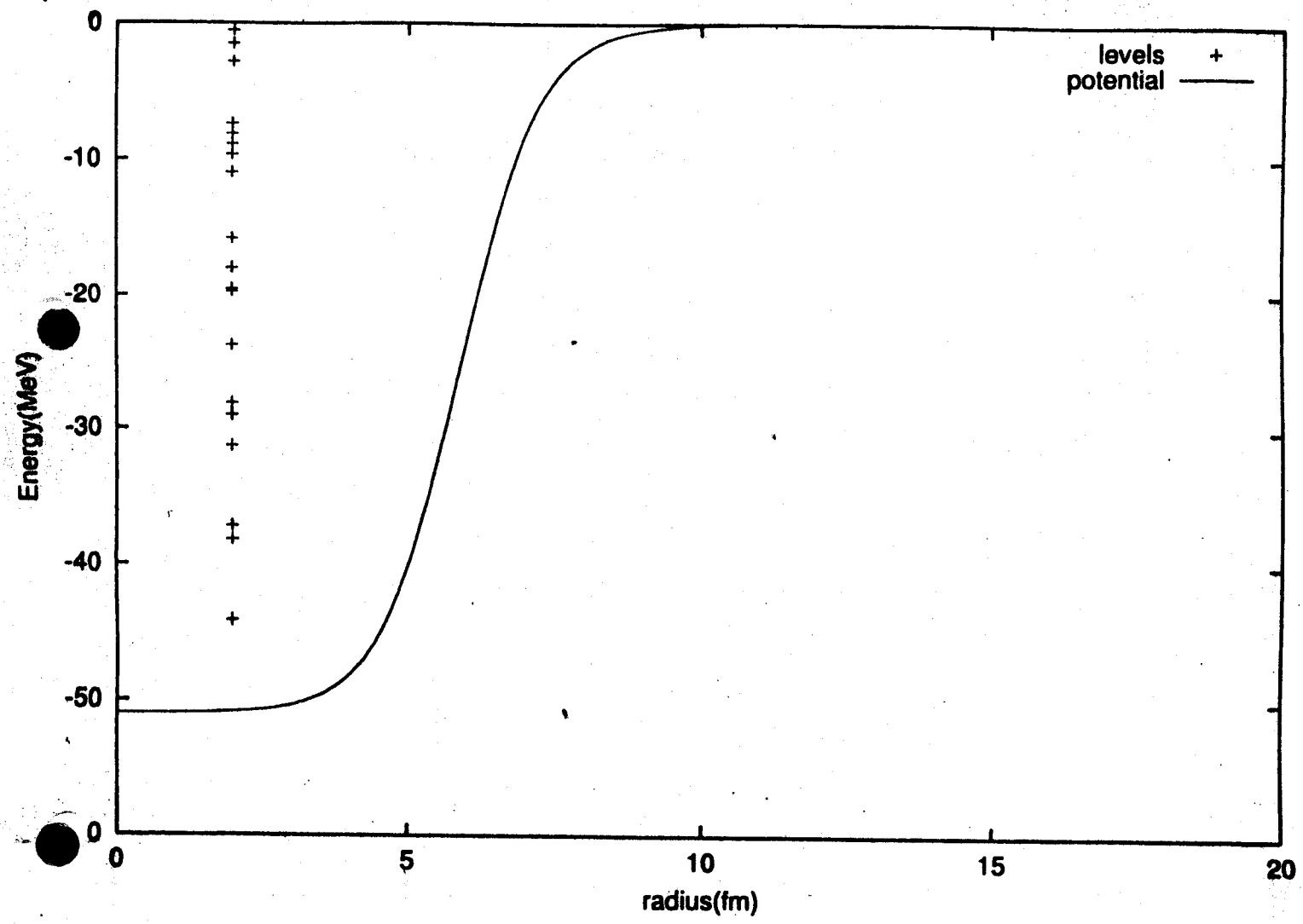

図2. 中性子のポテンシャルエネルギー ($l \cdot s$ポテンシャルを除く、実線の曲線) と中性子の一粒子準位 (+記号)。中性子のポテンシャルは負の値から 0 に向かって単調増加する曲線を描く。

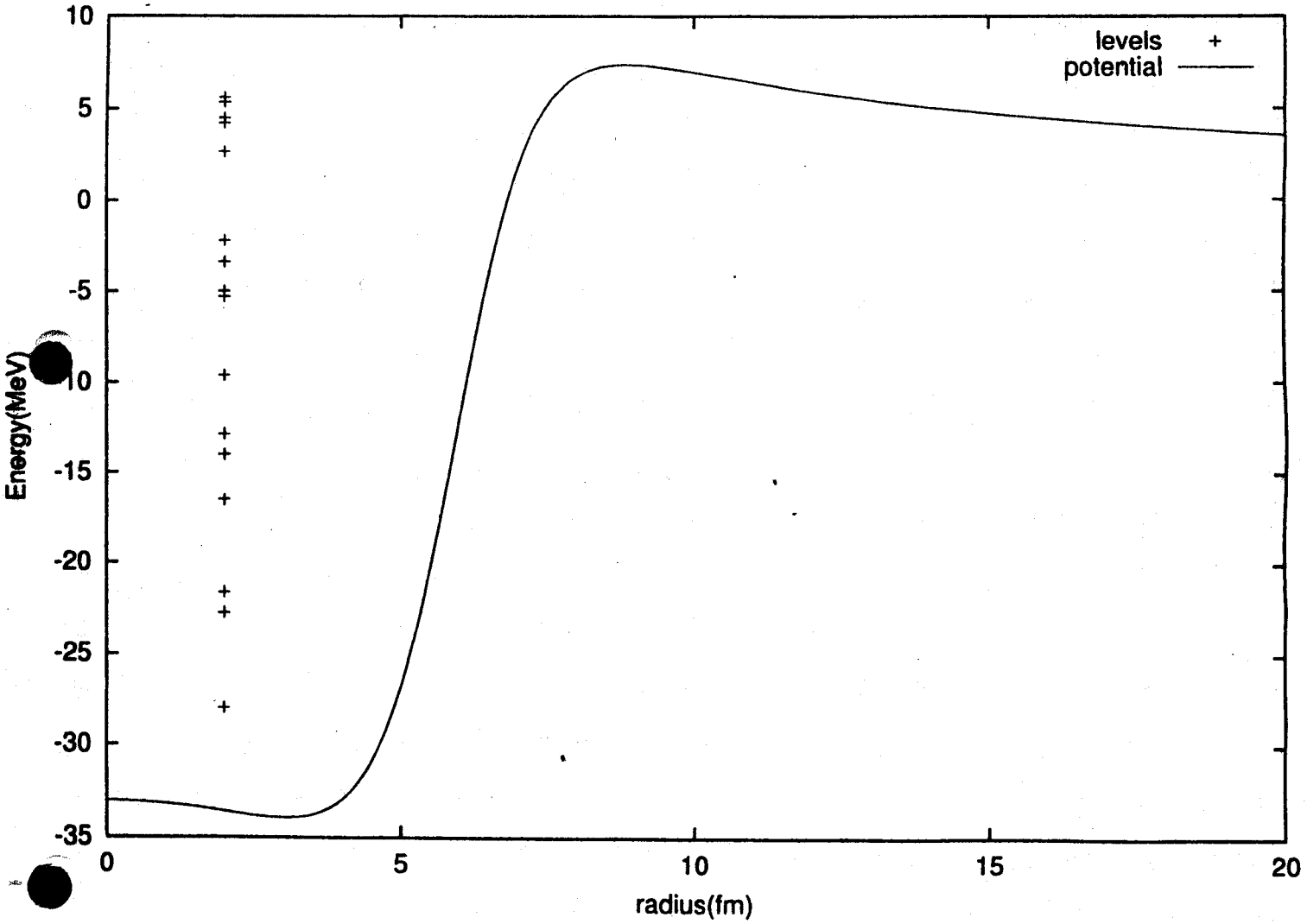

図3. 陽子のポテンシャルエネルギー ($l \cdot s$ポテンシャルを除く、実線の曲線) と陽子の一粒子準位 (+記号)。 正エネルギーの陽子のエネルギー固有値が離散的な値ををとるのはクーロンポテンシャルの外側をが 8MeV まで埋めてて計算したからである。陽子のポテンシャルは少し 0 をこえまた 0 に向かう形をとる。これは陽子にはクーロン力が働くからである。

第3章　モンテカルロ法

多自由度系の理論的究明は現在の科学の主要な課題である。しかし、自由度の壁に阻まれて十分な解析ができないことが多い。こういった問題に対して、モンテカルロ法は有用であり、多くの問題にたいしてほとんど唯一の手段となっている。モンテカルロ法のよさは、台形公式などの近似法では変数の個数である積分領域の次元数に本質的な制約を受けることになり、粒子が多くなると積分点の個数は次元数をべきとして増えるが、これに比べ、乱数を用いる方法はコンピュータのプログラムが簡単で次元数に無関係に同じ方法で積分値を求めることができ、誤差の評価も簡単である。さらに一度計算した後でも乱数を追加して精度を向上させることができるという便利な特徴をもっている。計算を任意の時点で打ち切っても全く無駄は生じず、それまでにかけた計算時間に応じた精度の結果を得ることができるのである。[7]

3.1　モンテカルロ積分

定積分

$$I = \int_a^b f(x)\,dx \tag{3.1}$$

について考える。$x_i (i = 1, 2, \cdots, N)$ は [a,b] で一様分布する乱数 とする。

定積分 I を中点公式で近似すると

$$I \doteq \frac{(b-a)}{N}\sum_{i=1}^{N} f(u_i), \quad u_i = \frac{b-a}{N}(i-\frac{1}{2}) \tag{3.2}$$

となるが、等間隔分点 u_i のかわりに一様に分布乱数点 x_i を用いて近似すると

$$\begin{aligned} I &\doteq \frac{b-a}{N}\sum_{i=1}^{N} f(x_i) \\ &= (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i) \end{aligned} \tag{3.3}$$

となる。x_i は乱数であるから $Y_i = f(x_i)$ も乱数である。すなわち (3.3) 式は、積分値は (区間の長さ) × (Y_i の期待値) とも表されることを示している。区間の長さは厳密にわかるので、積分の誤差は Y_i の期待値の誤差から生じるが、この誤差は中心極限定理により統計的に評価することができる。誤差を含めた結果を M 次元の場合について書くと積分値 I は

$$\begin{aligned} I &= \int_V f(\boldsymbol{x}) dx_1 dx_2 \cdots dx_M \\ &= V < f > \left\{ 1 \pm \frac{1}{\sqrt{N}} \sqrt{\frac{< f^2 > - < f >^2}{< f >^2}} \right\} \end{aligned} \tag{3.4}$$

となる。ここで $< f >$ と $< f^2 >$ は以下のように定義される。

$$< f >= \frac{1}{N}\sum_{i=1}^{N} f(x_i) \qquad < f^2 >= \frac{1}{N}\sum_{i=1}^{N} \{f(x_i)\}^2 \tag{3.5}$$

ここで分散を V、標準偏差を Δf と書くと

$$\begin{aligned} V &=< (f - < f >)^2 > \\ &= < f^2 - 2 < f > f + < f^2 > \\ &= < f^2 > - < f >^2 \end{aligned} \tag{3.6}$$

$$\Delta f = \sqrt{V} = \sqrt{< f^2 > - < f >^2} \tag{3.7}$$

であるので以上より積分値 I は以下のようにも表すことができる。

$$I = V < f > \left\{ 1 + \frac{\Delta f}{\sqrt{N} < f >} \right\} \tag{3.8}$$

モンテカルロ積分の誤差は (3.8) 式より $1/\sqrt{N}$ に比例するが M によらないので、次元数に関係なく積分値 I を求めることができるのである。

3.2 加重サンプリング (Importance sampling)

式 (3.8) より標本点 N を増やすこと以外に、$\Delta f / < f >$ を小さくすることでも精度が向上することがわかる。そのためには分布が一様でない乱数を用いればよい。[8] そこで $|f|$ に類似した確率密度 w をもつ乱数ベクトル ξ_i を用いると積分値 I は

$$\begin{aligned} I &= \int_V f(\boldsymbol{x}) dx_1 dx_2 \cdots dx_M \\ &= \int \frac{f(\boldsymbol{x})}{w(\boldsymbol{x})} w(\boldsymbol{x}) dx_1 dx_2 \cdots dx_M \\ &\doteq \frac{1}{N} \sum_{i=1}^{N} \frac{f(\boldsymbol{\xi_i})}{w(\boldsymbol{\xi_i})} \end{aligned} \tag{3.9}$$

となる。すなわち (3.8) 式と同様に中心極限定理で誤差を評価すると (3.9) 式の積分値 I は

$$I = < \frac{f}{w} > \left\{ 1 \pm \frac{1}{\sqrt{N}} \frac{\Delta\left(\frac{f}{w}\right)}{< \frac{f}{w} >} \right\} \tag{3.10}$$

となる。なお $w(\boldsymbol{x})$ は規格化

$$\int_V w(\boldsymbol{x}) dv = 1 \tag{3.11}$$

が解析的にできる関数であることが望ましい。

本研究では、ガウス分布をする乱数は、極座標法 (Polar 法、Box-Muller 法) [9] を用いて一様分布乱数を変換して発生させた。任意の分布関数をもつ乱数の発生も、次項で述べるメトロポリス法をはじめとするランダムウォークを用いる方法を使えば可能である。

なお、統計力学では物理量の熱力学的平均をモンテカルロ積分によって求めることが多い。この場合に用いる標本点のしたがう確率密度は $w = e^{-\beta E}$ であり、解析的に規格化できるような簡単な関数ではない。しかし計算すべき量が、

$$\langle \mathcal{O} \rangle = \frac{\int \mathcal{O} e^{-\beta E} d\Gamma}{\int e^{-\beta E} d\Gamma}. \tag{3.12}$$

のような二つの積分の比の形をしているため、分母のあらわす「確率分布関数の規格化定数」が、結局は不要となるのである。

一般には、２つの積分を同じ確率密度分布に従う乱数を用いてモンテカルロ積分で計算する場合、それぞれの積分値を求めるには確率密度分布の規格化定数が必要だが、２つの積分の比の計算には規格化定数は不要である。

3.3 メトロポリス法

任意の形の与えられた確率分布に従う乱数を発生させる1つの非常に一般的な方法はメトロポリス法として知られている。以下ではこの方法について説明する。説明の文章の作成にあたっては、おもに参考文献[10]のpp.213〜216を参考にした。

メトロポリス法は、積分変数の与えられた値にたいして重み関数が計算可能であることだけが必要なので、カノニカル・アンサンブルの重み関数が系の座標の非常に複雑な関数であるために、ほかの方法ではうまく乱数発生ができない統計力学の問題に幅広く利用されてきた。

メトロポリスのアルゴリズムは様々な方法で実行することができるが、本研究ではランダムウォークで乱数を発生させることにする。一般的には多次元の座標空間 X で確率分布 $w(X)$ にしたがって分布した点の集合を発生させてみる。メトロポリス法は、一連の点列 $X_0, X_1 \cdots$ を X 空間を動く酔歩者が次々と訪れる点として発生させる。酔歩が長ければ長いほどその結ぶ点は期待される分布より近付く。

酔歩が配位空間を動く規則はつぎのようである。酔歩者が一連のつながり中の X_n になる点にいるとする。X_{n+1} を生成するために、X_t に動くことを試みる。この新しい点は、例えば、X_n のまわりの小さな一辺 ξ なる多次元立方体中で一様かつランダムになど、任意の適当な方法で選ぶことができる。その後この試みの動きは、比

$$r = \frac{w(X_t)}{w(X_n)} \tag{3.13}$$

の値に応じて、「採用される」か、「棄却」される。もし r が1より大きければ、この動きを採用し、つまり、$X_{n+1} = X_t$ とし、1より小さければ確率 r で採用する。後者の操作は、r を区間[0,1]に一様分布した乱数 η と比較し、$\eta < r$ ならその動きを採用することによって実行することができる。もし試みの動きが採用さえなければ、それは棄却され、$X_{n+1} = X_n$ とする。このようにして X_{n+1} が生成され、同様に、X_{n+1} から試みの動きをすることによって X_{n+2} を生成する過程へとすすむ。酔歩の出発点としては任意の点 X_0 を用いる。

ある分布関数に見合う乱数 X を発生させる問題にメトロポリス法を適用すると、酔歩の次の段階を実行し、新しい X の値を発生させるプログラムは dx を一歩の最大幅、unirnd を [0,1] 区間に一様乱数とすると

$$X' = X * \boxed{\text{dx}} * (2 * \boxed{\text{unirnd}} - 1) \tag{3.14}$$

で決められ、その候補に移るか滞まるかは乱数によって決まる。ここでは酔歩の現時点での重み関数を保存することにより、試みの動きを採用するかどうかの決定に

あたって計算を繰り返す必要を避け効率をあげることができる。実際、w の演算は、メトロポリス法に基づいてモンテカルロ計算をする際にもっとも時間を費すところである。

ここで問題となるのは、もし試みの動きを X_n の近傍にとるべきだとしたら、どののように刻みの大きさを選ぶべきか という問題である。この問題はまず X_n がその最もありうる場所、つまり w の最大点にいるとすると、もし η が大きいと、多くの場合 $w(X_t)$ は (X_n) よりはるかに小さくなり、ほとんどの試みの動きは棄却され、w に従う乱数の抽出は非効率的になる。もし η が非常に小さいと、ほとんどの試みの動きは採用されるであろうが、酔歩者は決して、遠くまでいくことはなく、再び、効率の悪い乱数発生となる。そこで本研究ではよい目安として試みの動きの $\frac{1}{2}$ が採用されるように試みの動きの大きさを選ぶことにする。

またもう1つメトロポリス法を適用する際の問題は、酔歩をどこから始めるか、つまり初期値 X_0 として何をを用いるかという問題である。酔歩者はやがては「熱平衡」に達するので、原理的にはどのような場所にもとれるし、結果はその選択にはよらないはずである。実際には、適当な出発点の1つは、w が大きな値をとるもっともらしい点である。なお、本研究では6次元という比較的低次元のあまり複雑でない重み関数を用いるため初期値依存性はほとんど問題にならない。

最後に X_n からどれぐらいの周期で標本点をとると独立な点列が得られるかも大きな問題である。X_n は相関があるため、すべての X_n を積分点として使用しても (即ち周期1でサンプリングしても)、数十点に1点を積分点に使っても、結果として得られる積分値はほとんど同じ値になる。したがってサンプリング周期は、ランダムウォークの1ステップに要する計算時間と積分点を1点加えるのに要する計算時間のかねあいを考慮して決めるべきものである。また誤差評価は相関のため過小となることを理解しておくことが重要である。本研究ではサンプリング周期を 100 とした。

3.4 モンテカルロ積分のテスト計算

6重積分のテスト関数は以下の式を用いる。

$$
\begin{aligned}
I &= \frac{1}{I_0}\int_{-\infty}^{\infty}dx_1\int_{-\infty}^{\infty}dy_1\int_{-\infty}^{\infty}dz_1\int_{-\infty}^{\infty}dx_2\int_{-\infty}^{\infty}dy_2\int_{-\infty}^{\infty}dz_2 \\
&\quad \times e^{-ax_1^2}e^{-bx_2^2}e^{-c(x_1-x_2)^2}e^{-ay_1^2}e^{-by_2^2}e^{-c(y_1-y_2)^2}e^{-az_1^2}e^{-bz_2^2}e^{-c(z_1-z_2)^2} \qquad (3.15) \\
I_0 &= \left\{\frac{\pi}{\sqrt{ab+bc+ca}}\right\}^3 \qquad (3.16)
\end{aligned}
$$

I_0 は I=1 となるように選んである。積分公式

$$\int_{-\infty}^{\infty} du \int_{-\infty}^{\infty} dv \ e^{-au^2} e^{-bv^2} e^{-c(u-v)^2} = \frac{\pi}{\sqrt{ab+bc+ca}} \tag{3.17}$$

を使うと、(3.16) 式の値がえられる。ガウス関数のパラメータ a, b, c とその表す幅の関係は次の通りである。変数 ξ の確率密度が $e^{-\alpha\xi^2}$ のとき

$$\sqrt{<\xi^2>} = \frac{1}{\sqrt{2\alpha}} \quad . \tag{3.18}$$

パラメータの値は $a = 1.0, b = 2.0, c = 0.3$ にとった。

式を計算するモンテカルロ積分では次の 3 通りの乱数を使う。

(1) 一様乱数
その分布範囲は

$$\begin{aligned} &L_1 \le x_1 \le L_1, \ -L_2 \le x_2 \le L_2 \\ &L_1 \le y_1 \le L_1, \ -L_2 \le y_2 \le L_2 \\ &L_1 \le z_1 \le L_1, \ -L_2 \le z_2 \le L_2 \\ &L_1 = 6a, L_2 = 6b \end{aligned} \tag{3.19}$$

(2) ガウス分布乱数
確率密度 w は、被積分関数 (3.15) 式より 10 %大きな幅をもつガウス関数とした。

$$\begin{aligned} &w \propto e^{-a'\boldsymbol{r_1}^2} e^{-b'\boldsymbol{r_2}^2} \\ &a' = \frac{a}{1.1^2}, \quad b' = \frac{b}{1.1^2} \end{aligned} \tag{3.20}$$

c=0 の場合は $a = a'$,$b = b'$ ととると、モンテカルロ積分は標本数が 1 でも厳密に正しい値を与える。

(3) メトロポリス法を用いたランダムウォーク乱数
(3.14) 式により次の候補点 X_{i+1} を選び (X は$\boldsymbol{r_1}$,$\boldsymbol{r_2}$の組を表す)、その採否は確率密度 $w(X_i)$ と $w(X_{i+1})$ の比による。ランダムウォークから 100 点毎に 1 点を積分の標本点として採用した。確率密度は以下のとおりである。

$$w = \mathcal{N} \ e^{-a'\boldsymbol{r_1}^2} e^{-b'\boldsymbol{r_2}^2} e^{-c'(\boldsymbol{r_1}-\boldsymbol{r_2})^2} \tag{3.21}$$

$$\mathcal{N} = \left\{ \frac{\sqrt{a'b' + b'c' + c'a'}}{\pi} \right\}^3 \tag{3.22}$$

$$c' = \frac{c}{1.1^2} \tag{3.23}$$

である。エラーとエラーの評価値は

・エラー　$< \frac{f}{w} > -1$ (3.24)

・エラーの評価値　$\frac{\Delta(\frac{f}{w})}{\sqrt{N}}$, $\Delta(\frac{f}{w}) = \sqrt{< (\frac{f}{w})^2 > - < \frac{f}{w} >^2}$ (3.25)

である。次ページにエラーとエラーの評価値を標本点数Nに対して両対数スケールでプロットした図4をしめす。

図4を見て分かることは3通りの乱数のエラーはその評価値同様に $\frac{1}{\sqrt{N}}$ に比例して変化していること、また3つの乱数で $N = 1.0 \times 10^9$ で計算したところガウス乱数のほうが一様乱数より1/47倍、またメトロポリス法のほうがガウス分布より1/56倍小さくなっていることである。

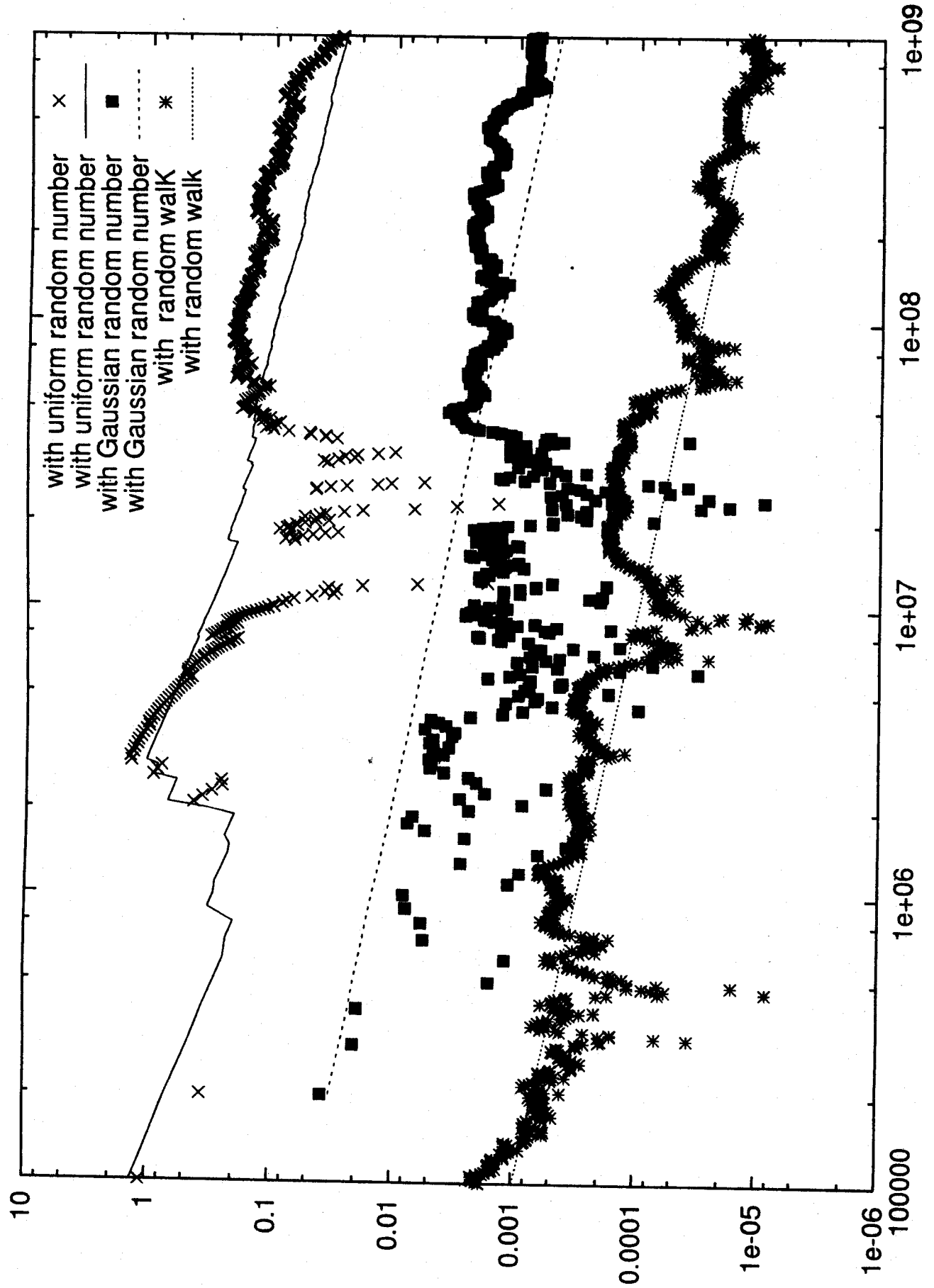

図 4. 3 通りの乱数の精度

第4章　二体相互作用行列要素の計算

4.1　演算子の行列要素

一般に、ある演算子$\mathcal{O}$を$\psi_m^*(x)$と$\psi_n(x)$とで挟んで積分をとったものを$< m|\mathcal{O}|n >$と書く。すなわち

$$< m|\mathcal{O}|n > \equiv \int_{\infty}^{\infty} \psi_m^*(x)\mathcal{O}\psi_n(x)dx \tag{4.1}$$

である。またm=nのときが$\mathcal{O}$のその状態における期待値である。系の正規完全直交完全系をなす波動関数$\psi_m(x)$の順番を適当に決めてやれば、$< m|\mathcal{O}|n >$は行列の(m,n)成分とみなせる。すなわち、演算子$\mathcal{O}$の行列と考えてよい。そこで、$< m|\mathcal{O}|n >$を行列要素とよぶ。

4.2　2体相互作用行列要素

状態ψ_3にある粒子1と状態ψ_4にある粒子2が相互作用してそれぞれ状態ψ_1と状態ψ_2に移る過程の2体の相互作用行列要素は6重積分

$$V_{1234} = \int\int d^3r_1 d^3r_2 \psi_1^*(\boldsymbol{r_1})\psi_2^*(\boldsymbol{r_2})V(\boldsymbol{r_2}-\boldsymbol{r_1})\psi_3(\boldsymbol{r_1})\psi_4(\boldsymbol{r_2}) \tag{4.2}$$

で表される。粒子が核子の場合は4.4節で述べるようにスピンσ、アイソスピンqについての和も必要となるので被積分関数が下式のように複雑になるがモンテカルロ積分の手続きには本質的な違いはない。

$$V_{1234} = \int\int d^3r_1 d^3r_2 f(\boldsymbol{r_1},\boldsymbol{r_2}), \tag{4.3}$$

$$f(\boldsymbol{r_1},\boldsymbol{r_2}) = \sum_{\sigma_1'q_1'\sigma_2'q_2'\sigma_1q_1\sigma_2q_2} \psi_1^*(\boldsymbol{r_1},\sigma_1^*,q_1^*)\psi_2^*(\boldsymbol{r_2},\sigma_2^*,q_2^*)$$
$$\times V(\boldsymbol{r_1}-\boldsymbol{r_2},\sigma_1',q_1',\sigma_2',q_2',\sigma_1,q_1,\sigma_2,q_2)\psi_3(\boldsymbol{r_1},\sigma_1,q_1)\psi_4(\boldsymbol{r_2},\sigma_2,q_2) \quad (4.4)$$

4.3 有効相互作用 Gogny

核力は相互作用している2核子の波動関数がどのようなスピン・アイソスピン状態にあるかに依存する。また、相対運動の軌道角運動にも強く依存する。
原子核という多体系での核力は、さまざまな多体効果のために、2核子系での核力と異なる。これを核内での有効相互作用という。Gogny力は現象論的に導かれた密度依存性をもつ有効相互作用であり下式で与えられる。

$$\begin{aligned} V(1,2) = &\sum_{i=1}^{2} e^{-\frac{(\boldsymbol{r_1}-\boldsymbol{r_2})^2}{\mu_i^2}}(W_i + B_iP^\sigma - H_iP^\tau - M_iP^\sigma P^\tau) \\ &+ iW_0(\boldsymbol{\sigma_1}+\boldsymbol{\sigma_2})\boldsymbol{k}\times\delta(\boldsymbol{r_1}-\boldsymbol{r_2})\boldsymbol{k} \\ &+ t_3(1+P^\sigma)\delta(\boldsymbol{r_1}-\boldsymbol{r_2})\rho^{\frac{1}{3}}\frac{(\boldsymbol{r_1}-\boldsymbol{r_2})}{2} \end{aligned} \quad (4.5)$$

表 4.1: Gogny力のパラメータ [11]

i	μ_i[fm]	W_i	B_i	H_i	M_i [MeV]	
1	0.7	−402.4	−100	−496.2	−23.56	$W_0 = +115$ [MeVfm^5]
2	1.2	−21.30	−11.77	37.27	−68.81	$t_3 = 1350$ [MeVfm^4]

また$\boldsymbol{r_1}$は粒子1の座標 、$\boldsymbol{r_2}$は粒子2の座標、 P^σ は粒子1,2のスピン座標の交換演算子、P^τ は粒子1,2のアイソスピン座標の交換演算子、$\boldsymbol{\sigma_{1,2}}$は粒子1,2のスピン演算子 、$\boldsymbol{k}$は粒子1,2の波数ベクトル演算子 、$\rho$ は粒子数密度を表す。

ここで(4.5)式の第1項のカッコの中の4項は4種類のスピン、アイソスピンの依存性を示し、第2項はスピン軌道を示し、また第3項は密度依存ゼロレンジ力を表す。第2項と第3項はゼロレンジ力なので、行列要素が3重積分で計算できるため、モンテカルロ積分を使わなくてよい。

今回の計算では第1項の有限レンジのスピン、アイソスピンに依存しない W_i 項だけを計算の対象とした。すなわちガウス分布乱数およびメトロポリス法で生成した乱数を用い以下の2体相互作用

$$v(\boldsymbol{r_1},\sigma_1 q_2,\boldsymbol{r_2},\sigma_2,q_2) = W_1 e^{-\frac{(\boldsymbol{r_1}-\boldsymbol{r_2})^2}{\mu_i^2}} + W_2 e^{-\frac{(\boldsymbol{r_1}-\boldsymbol{r_2})^2}{\mu_i^2}} \tag{4.6}$$

の行列要素を計算した。

4.4 核子間2体相互作用行列要素の計算

原子 $^{100}_{50}Sn_{50}$ の 行列要素

$$\langle\ \pi(0f_{\frac{7}{2}})\ \ \nu(0g_{\frac{7}{2}})|v|\pi(1p_{\frac{3}{2}})\ \ \nu(1d_{\frac{3}{2}})\rangle \tag{4.7}$$

の値の計算をする。ここで陽子の角運動量のz成分は m=1/2、中性子は m=-1/2 にとった。W_i 項の相互作用行列要素は

$$v_{abcd} = \sum_{\sigma_1}\sum_{\sigma_2}\int d^3r_1 \int d^3r_2 \psi_a^*(\boldsymbol{r_1},\sigma_1)\psi_b^*(\boldsymbol{r_2},\sigma_2)v(\boldsymbol{r_1}-\boldsymbol{r_2})\psi_c(\boldsymbol{r_1},\sigma_1)\psi_d(\boldsymbol{r_2},\sigma_2) \tag{4.8}$$

である。波動関数を軌道部分とスピン部分にわけると

$$\psi_{qnljm}(\boldsymbol{r},\sigma) = \psi_{qnlj,m_l=m-\frac{1}{2}}(\boldsymbol{r})X_{\frac{1}{2}}(\sigma) + \psi_{qnlj,m_l=m+\frac{1}{2}}(\boldsymbol{r})X_{-\frac{1}{2}}(\sigma) \tag{4.9}$$

となるが、以下では $m_s = 1/2$ を↑、$m_s = -1/2$ を↓と書き表して (4.9) 式を下記の (4.10)〜(4.13) 式のように略記する。

$$\psi_a^*(\boldsymbol{r_1},\sigma_1) = \psi_{a\downarrow}^*(\boldsymbol{r_1})X_\downarrow(\sigma_1) + \psi_{a\uparrow}^*(\boldsymbol{r_1})X_\uparrow(\sigma_1) \tag{4.10}$$

$$\psi_b^*(\boldsymbol{r_2},\sigma_2) = \psi_{b\downarrow}^*(\boldsymbol{r_2})X_\downarrow(\sigma_2) + \psi_{b\uparrow}^*(\boldsymbol{r_2})X_\uparrow(\sigma_2) \tag{4.11}$$

$$\psi_c(\boldsymbol{r_1},\sigma_1) = \psi_{c\downarrow}(\boldsymbol{r_1})X_\downarrow(\sigma_1) + \psi_{c\uparrow}(\boldsymbol{r_1})X_\uparrow(\sigma_1) \tag{4.12}$$

$$\psi_d(\boldsymbol{r_2},\sigma_2) = \psi_{d\downarrow}(\boldsymbol{r_2})X_\downarrow(\sigma_2) + \psi_{d\uparrow}(\boldsymbol{r_2})X_\uparrow(\sigma_2) \tag{4.13}$$

それぞれの波動関数が相互作用すると行列要素 v_{abcd} の成分が8項できるが、m_s に直交関係

$$\sum_{\sigma=\uparrow\downarrow} X_{\uparrow}(\sigma_1)X_{\downarrow}(\sigma_1) = 0 \tag{4.14}$$

が成り立つことを利用すると、相互作用行列要素の被積分関数には以下の4項

$$f(\boldsymbol{r_1},\boldsymbol{r_2}) = v(\boldsymbol{r_1}-\boldsymbol{r_2}) \times \{ \quad \psi^*_{1\downarrow}\psi^*_{2\downarrow}\psi_{3\downarrow}\psi_{4\downarrow} \tag{4.15}$$

$$+ \quad \psi^*_{1\downarrow}\psi^*_{2\uparrow}\psi_{3\downarrow}\psi_{4\uparrow}$$

$$+ \quad \psi^*_{1\uparrow}\psi^*_{2\uparrow}\psi_{3\uparrow}\psi_{4\uparrow}$$

$$+ \quad \psi^*_{1\uparrow}\psi^*_{2\downarrow}\psi_{3\uparrow}\psi_{4\downarrow}\} \tag{4.16}$$

が残る。式(4.16)の第1項を波動関数が各々虚数部分をもつことを考慮し計算すると

$$\begin{aligned}
&\psi^*_{1\downarrow}\psi^*_{2\downarrow}\psi_{3\downarrow}\psi_{4\downarrow} \\
&= \{(\phi_{1\downarrow}\phi_{2\downarrow}-\varphi_{1\downarrow}\varphi_{2\downarrow})(\phi_{3\downarrow}\phi_{4\downarrow}-\phi_{3\downarrow}\varphi_{4\downarrow})+(\phi_{1\downarrow}\varphi_{2\downarrow}+\varphi_{1\downarrow}\phi_{2\downarrow})(\phi_{3\downarrow}\varphi_{4\downarrow}+\varphi_{3\downarrow}\phi_{4\downarrow})\} \\
&+ \ i \ \{(\phi_{1\downarrow}\phi_{2\downarrow}-\varphi_{1\downarrow}\varphi_{2\downarrow})(\phi_{3\downarrow}\varphi_{4\downarrow}+\varphi_{3\downarrow}\phi_{4\downarrow})-(\phi_{1\downarrow}\varphi_{2\downarrow}-\varphi_{1\downarrow}\phi_{2\downarrow})(\phi_{3\downarrow}\phi_{4\downarrow}-\varphi_{3\downarrow}\varphi_{4\downarrow})\}
\end{aligned} \tag{4.17}$$

となる。他の3項は$\downarrow\uparrow$を入れ替えて項を計算した。すなわち求める行列要素の被積分関数の実数部分を$f_R(\boldsymbol{r_1},\boldsymbol{r_2})$虚数部分を$f_I(\boldsymbol{r_1},\boldsymbol{r_2})$とすると

$$\begin{aligned}
f_R(\boldsymbol{r_1},\boldsymbol{r_2}) = v(\boldsymbol{r_1}-\boldsymbol{r_2}) \times \Big[& \\
&\{(\phi_{1\downarrow}\phi_{2\downarrow}-\varphi_{1\downarrow}\varphi_{2\downarrow})(\phi_{3\downarrow}\phi_{4\downarrow}-\phi_{3\downarrow}\varphi_{4\downarrow})+(\phi_{1\downarrow}\varphi_{2\downarrow}+\varphi_{1\downarrow}\phi_{2\downarrow})(\phi_{3\downarrow}\varphi_{4\downarrow}+\varphi_{3\downarrow}\phi_{4\downarrow})\} \\
+ \ &\{(\phi_{1\downarrow}\phi_{2\uparrow}-\varphi_{1\downarrow}\varphi_{2\uparrow})(\phi_{3\downarrow}\phi_{4\uparrow}-\phi_{3\downarrow}\varphi_{4\uparrow})+(\phi_{1\downarrow}\varphi_{2\uparrow}+\varphi_{1\downarrow}\phi_{2\uparrow})(\phi_{3\downarrow}\varphi_{4\uparrow}+\varphi_{3\downarrow}\phi_{4\uparrow})\} \\
+ \ &\{(\phi_{1\uparrow}\phi_{2\uparrow}-\varphi_{1\uparrow}\varphi_{2\uparrow})(\phi_{3\uparrow}\phi_{4\uparrow}-\phi_{3\uparrow}\varphi_{4\uparrow})+(\phi_{1\uparrow}\varphi_{2\uparrow}+\varphi_{1\uparrow}\phi_{2\uparrow})(\phi_{3\uparrow}\varphi_{4\uparrow}+\varphi_{3\uparrow}\phi_{4\uparrow})\} \\
+ \ &\{(\phi_{1\uparrow}\phi_{2\downarrow}-\varphi_{1\uparrow}\varphi_{2\downarrow})(\phi_{3\uparrow}\phi_{4\downarrow}-\phi_{3\uparrow}\varphi_{4\downarrow})+(\phi_{1\uparrow}\varphi_{2\downarrow}+\varphi_{1\uparrow}\phi_{2\downarrow})(\phi_{3\uparrow}\varphi_{4\downarrow}+\varphi_{3\uparrow}\phi_{4\downarrow})\}\Big]
\end{aligned} \tag{4.18}$$

$$f_I(\boldsymbol{r_1},\boldsymbol{r_2}) = v(\boldsymbol{r_1}-\boldsymbol{r_2}) \times \Big[$$

$$
\begin{aligned}
& \{(\phi_{1\downarrow}\phi_{2\downarrow} - \varphi_{1\downarrow}\varphi_{2\downarrow})(\phi_{3\downarrow}\varphi_{4\downarrow} + \varphi_{3\downarrow}\phi_{4\downarrow}) - (\phi_{1\downarrow}\varphi_{2\downarrow} + \varphi_{1\downarrow}\phi_{2\downarrow})(\phi_{3\downarrow}\phi_{4\downarrow} - \varphi_{3\downarrow}\varphi_{4\downarrow})\} \\
+ \; & \{(\phi_{1\downarrow}\phi_{2\uparrow} - \varphi_{1\downarrow}\varphi_{2\uparrow})(\phi_{3\downarrow}\varphi_{4\uparrow} + \varphi_{3\downarrow}\phi_{4\uparrow}) - (\phi_{1\downarrow}\varphi_{2\uparrow} + \varphi_{1\downarrow}\phi_{2\uparrow})(\phi_{3\downarrow}\phi_{4\uparrow} - \varphi_{3\downarrow}\varphi_{4\uparrow})\} \\
+ \; & \{(\phi_{1\uparrow}\phi_{2\uparrow} - \varphi_{1\uparrow}\varphi_{2\uparrow})(\phi_{3\uparrow}\varphi_{4\uparrow} + \varphi_{3\uparrow}\phi_{4\uparrow}) - (\phi_{1\uparrow}\varphi_{2\uparrow} + \varphi_{1\uparrow}\phi_{2\uparrow})(\phi_{3\uparrow}\phi_{4\uparrow} - \varphi_{3\uparrow}\varphi_{4\uparrow})\} \\
+ \; & \{(\phi_{1\uparrow}\phi_{2\downarrow} - \varphi_{1\uparrow}\varphi_{2\downarrow})(\phi_{3\uparrow}\varphi_{4\downarrow} + \varphi_{3\uparrow}\phi_{4\downarrow}) - (\phi_{1\uparrow}\varphi_{2\downarrow} + \varphi_{1\uparrow}\phi_{2\downarrow})(\phi_{3\uparrow}\phi_{4\downarrow} - \varphi_{3\uparrow}\varphi_{4\downarrow})\}\Big]
\end{aligned}
\tag{4.19}
$$

となる。

図5より小さな値では大きな揺らぎがあるが N が大きくなると揺らぎが30keVくらいに減少することがわかる。また行列要素の収束値はおよそ-0.66MeVである。誤差棒で表した誤差の評価値 $\Delta(f/w)/\sqrt{N}$ は収束値からのずれの大きさと同程度になっていることが確認できる。図5にガウス分布乱数を用いたモンテカルロ積分の結果を示す。ランダムウォークによる行列要素の計算は時間が無くできなかった。

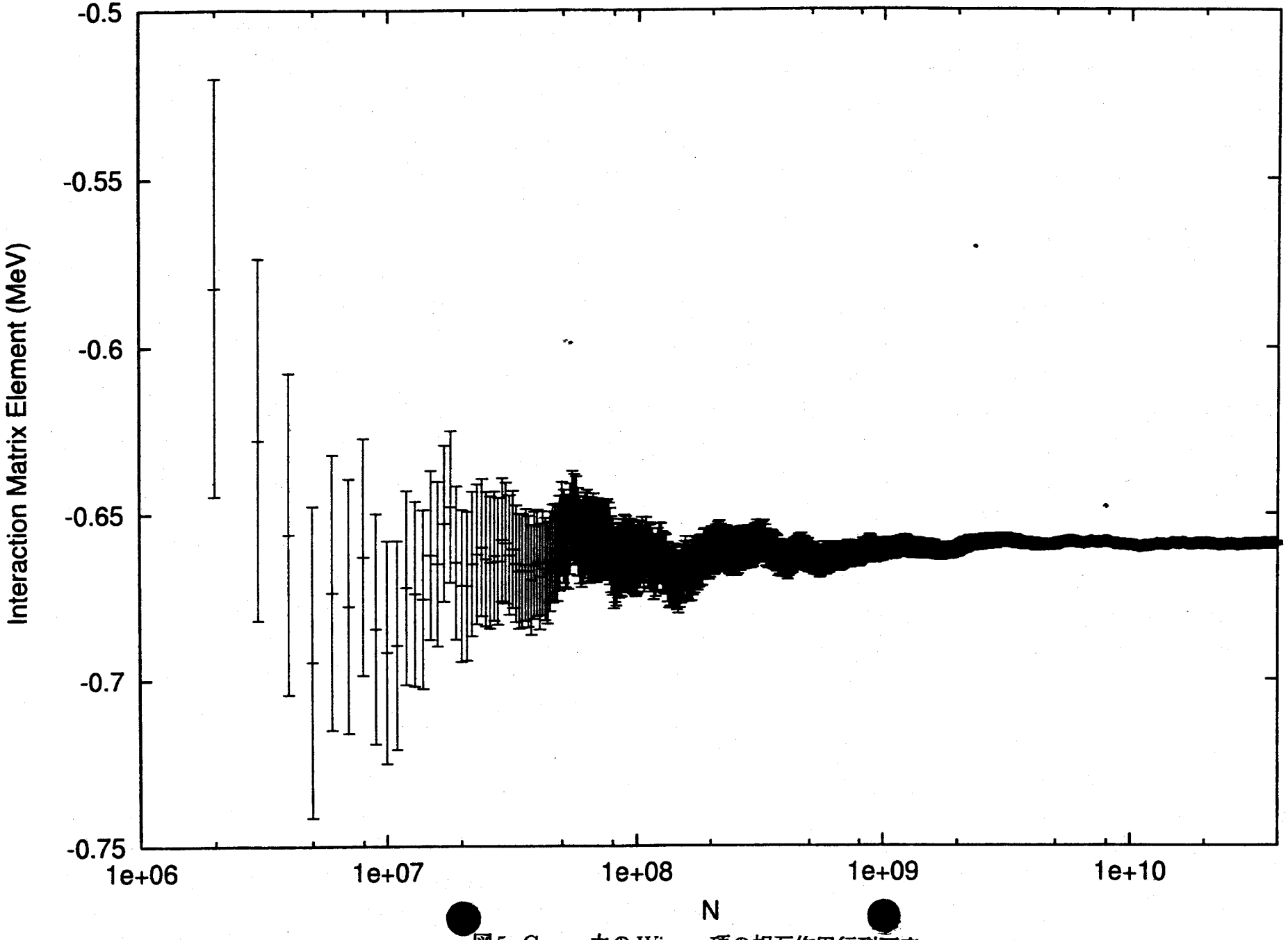

図5. Gogny 力の Wigner 項の相互作用行列要素

結論

本研究ではモンテカルロ積分を原子核中の２個の核子の相互作用行列要素を求めるための６重積分の計算に応用した。

相互作用としては、平均場計算でよく使われる Gogny 力を用いた。Gogny 力はレンジを違えた２個のガウス関数の重ね合わせで核力の斥力芯を除く有効部分を近似したものである。また核子の状態としては、^{100}Sn の陽子・中性子の状態をとった。

これらの状態と相互作用に対してモンテカルロ積分による行列要素の計算を実行し、積分の標本点の個数に応じて積分値が収束していく様子と、それぞれの時点での積分の誤差の評価値の妥当性の確認を行なった。

また、モンテカルロ積分の高効率化に効果が高いと言われる加重サンプリング法を試し、積分の標本点発生の確率密度として被積分関数に似た関数を用いることで、モンテカルロ積分の収束が速くなることを示した。

なお、行列要素の計算に先だって、解析的に値の求まる６重積分を用いて、モンテカルロ積分のテストを行い、使用した乱数が、一様分布乱数 < ガウス分布乱数 < メトロポリス法 の順に効率が高くなること、またどの手法も正確な値へと収束することを確認した。

本研究で計算した行列要素は Gogny 力の含む諸項のうち Wigner 項のみである。今後は、他の項も含めて完全な Gogny 力の行列要素を計算し、大学院進学後の研究に役立てたい。修士論文のテーマには異なる主殻にある陽子・中性子が対に組むような新しいタイプの核子の超伝導状態を探索する研究を想定している。

謝辞

本研究を行うにあたり、御指導を頂いた田嶋直樹先生、鈴木敏男先生、林明久先生に厚く御礼申し上げます。特に田嶋先生には最初から最後まで懇切丁寧な御指導を賜わり感謝の念に堪えません。
最後になりましたが、物理工学科の諸先生方に対しても、日頃の御指導に感謝し、謝辞の言葉とします。

2003年2月
稲垣安章

関連図書

[1] 有馬朗人 「原子と原子核」、朝倉書店 (1982)

[2] 市村宗武 他、「岩波講座 現代の物理学 9 原子核の理論」、岩波書店 (1993)

[3] 中村誠太郎 「大学院原子核物理」、講談社サイエンティフィク (1996)

[4] 杉本健三、「共立物理学講座 22 原子核物理学」、共立出版 (1988)

[5] A.Bore,B.R.Mottelson、 有馬朗人 他訳、 「原子核構造」 講談社、(1979)

[6] 河原林研、「岩波講座 現代の物理学 3 量子力学」、岩波書店 (1993 年)

[7] 宮武修 他、「乱数とモンテカルロ法」、森北出版 (1978)

[8] 津田孝夫 他、「数値処理プログラミング」、岩波書店、(1988)

[9] 奥村晴彦、「C 言語による最新アルゴリズム事典」、技術評論社 (1991 年)、p.133.

[10] S.E.Koonin、 阿部正典 他訳 「クーニン計算機物理学」、共立出版、(1992)

[11] P.Ring and P.Schuck,The nucler many-body program、(Spring,New York,1980)

付録　Program List

プログラム1

```
/*-----------------------------------------------------------------
原子核中の固有状態をメモリに記憶するプログラムのソースリスト
-----------------------------------------------------------------*/
int store_wavefunctions(){
  int j,i,q,l,cls,n ;
  double x,y;
  double *ptr;

  for(q=0;q<=1;q++){
    for(l=0;l<=LMAX;l++){
      for(cls=0;cls<=1;cls++){
        for(n=0;n<=NMAX;n++){
          spectrum[q][l][cls][n]=0.0;
          ptr_wf[q][l][cls][n]=NULL;
}
      }
    }
  }
  for(q=0;q<=1;q++){
    for(l=0;l<=LMAX;l++){
      if(l == 0) cls=1; else cls=0;
      for(;cls<=1;cls++){
        j=2*(l+cls)-1;
        set_potential(Nu,Z,A,l,j,q);
        for(n=0;n<=NMAX;n++){
          if(radwf(n,l,j,q)) break ;
          printf("Solution : E=%17.12f  for n=%d l=%d 2j=%d q=%d\n"
          ,energy,n,l,j,q);
          spectrum[q][l][cls][n]=energy;
          ptr=(double *)malloc(sizeof(double)*(IMAX+1));
          if(ptr==NULL){
            fputs("Failed to malloc. Abort.\n",stderr);
            exit(1);
  }
```

```
            for(i=0;i<=IMAX;i++) ptr[i]=wf[i];
            ptr_wf[q][l][cls][n]=ptr;
}
        }
      }
    }
  mesh_spacing=h;
  return 0;
}

/* ---------- Radial Wavefunctions ----------*/

int runge(double E,double L);
/************************************************/
int radwf(int N,int L,int J,int T){
/*
  radial wave function
  return value = 0 : solution found, -1:not found
  Results are set in global variables "wf[]" and "energy"
*/
  int node, node1,node2,node3,nodetemp,p,i;
  double x,E1,E2,Etemp,S,Sm,f;
  E1=-60;
  if(T==NEUTRON) E2=EMAXN; else E2=EMAXP;
  node1=runge(E1,L);
  node2=runge(E2,L);
  if(! (N >= node1 && N < node2)){
    /*
    printf(" No solutions for N=%d L=%d 2J=%d T=%d\n",N,L,J,T);
    printf("   # Node  = %d for Energy=%f\n", node1,E1);
    printf("   # Node  = %d for Energy=%f\n", node2,E2);
    */
    return -1;
  }

  for(;;){
    if(fabs((E1-E2)/E1)<=1e-10) break;
    Etemp=(E1+E2)/2;
    nodetemp=runge(Etemp,L);
    /* printf(" E=%16.8f node=%6d\n",Etemp,nodetemp); */
    if(nodetemp>N){
      E2=Etemp;
      node2=nodetemp;
    }
    else {
      E1=Etemp;
      node1=nodetemp;
```

```
    }
  }
  energy=E2;
  node3=runge(energy,L);
  if(node3 != N+1) printf(" error : node = %d != 1+N=1+%d\n",node3,N);

  for(i=IMAX-1;i>=0;i--){ if(wf[i]*wf[i+1]<=0) break; }
  p=i; for(i=p;i<=IMAX;i++) wf[i]=0;
  Sm=0; for(i=0;i<=IMAX;i++){ Sm=Sm+wf[i]*wf[i]; } Sm *= h;
  /*printf("Before Normalization: Sm=%f\n",Sm);*/
  f=1/sqrt(Sm); for(i=0;i<=IMAX;i++){ wf[i]*=f; }
  /*for(i=0;i<=IMAX;i++){x=h*i;printf("%f %f %f
:u\n",x,wf[i],potential(x));}*/
  return 0; /* solution is found */
}

/*************************************************************/
double frunge(double x, double L,double E) {
  return potential(x)/HBSO2M-E/HBSO2M+L*(L+1)/(x*x);
}
/*************************************************************/
int runge(double E,double L){
  double xs=0,xe=RMAX;
  double h2,h3,h6;
  double x0,x1,ya0,ya1,yb0,yb1,f0,fm,f1,ka1,kb1,kb2,ka2,ka3,kb3,ka4,kb4;
  int i;
  int node=0;
  h=(xe-xs)/IMAX;i=Rc/h;h=Rc/i;xe=IMAX*h;  h2=h/2.0; h3=h/3.0;h6=h/6.0;
  for(i=0;i<=5;i++){
    x1=i*h;
    ya1=pow(x1,L+1);
    wf[i]=ya1;
    if(L>0){
      yb1=(L+1)*pow(x0,L);
    }
    else{
      yb1=1;
    }
    /*printf("%12.6f %12.6f %12.6f\n",x1,ya1,yb1);*/
    x0=x1;ya0=ya1;yb0=yb1;
  }

  f0=frunge(x0,L,E);
  for(i=6;i<=IMAX;i++){
    x1=xs+i*h;
    fm=frunge(x0+h2,L,E);
    f1=frunge(x1,L,E);
```

```
    ka1=yb0;
    kb1=f0*ya0;
    ka2=yb0+h2*kb1;
    kb2=fm*(ya0+h2*ka1);
    ka3=yb0+h2*kb2;
    kb3=fm*(ya0+h2*ka2);
    ka4=yb0+h*kb3;
    kb4=f1*(ya0+h*ka3);
    ya1=ya0+h6*(ka1+ka4)+h3*(ka2+ka3);
    yb1=yb0+h6*(kb1+kb4)+h3*(kb2+kb3);
    /*printf("%12.6f %12.6f %12.6f\n",x1,ya1,yb1);*/
    /* xs< x0 < x1 <= xe で ya0 と ya1 の符号が違えば x0,x1 間に
       ノード 1個ありと数える */
    wf[i]=ya1;
    if(ya0*ya1 <= 0.0 ) node++;
    x0=x1;ya0=ya1;yb0=yb1;f0=f1;
  }
  return node;
}

/* ------------------ Potentials ---------------------*/
#define r0  1.27
#define Als 0.67
#define e2   1.439965
double p,v,b;   /* Rc is also set below */
int Proton_or_Neutron;
/*******************************************************/
double wood(double x){
     return 1/(1+exp((x-Rc)/Als));
}
/*************************************************************/
double potential(double x){
  double g,s;
  if(x == 0.0) x=1.0e-6;
  g=wood(x)*p+v*wood(x)*(wood(x)-1)/x;
  if(Proton_or_Neutron == PROTON){
    if(x<=Rc){
      s=b*(3-((x/Rc)*(x/Rc)))/(2*Rc);
    }
    else {
      s=b/x;
    }
    if(s < EMAXP) s=EMAXP;

    g+=s;
  }
  return g;
```

```
}
/*****************************************************************/
int set_potential(int Nu,int Z, int A,int L,int J,int T){
  if(J != 2*L-1 && J != 2*L+1){
    fprintf(stderr,"error: J=%d/2 for L=%d is not possible.\n",J,L);
    exit(1);
  }

  p=-51+132*(-0.5)*(Nu-Z)/(2*A);
  v=-(0.44)*p*r0*r0*(0.5*(J*(J+2)*0.25-L*(L+1)-0.5*(1.5)))/Als;
  b=(Z-1)*e2;
  Rc=r0*pow(A,1.0/3.0);
  Proton_or_Neutron=T;
  /*
  printf("Nu=%d  Z=%d  A=%d  L=%d  2J=%d\n",Nu,Z,A,L,J);
  printf(" p=%f  v=%f  b=%f  Rc=%f\n",p,v,b,Rc);
  */
}
```

プログラム2

```
/-------------------------------------------------------------/
ylm.c : 球面調和関数の計算
/-------------------------------------------------------------/
#include <stdio.h>
#include <math.h>
#define eps 1.0e-15

const double pi=3.1415926535897932;

double ylm(int l, int ma, double t);
int function1();
int function2();

main(){
  function2();
}

function1(){
  double t ,y;
  int i,j,k,l,n,m;
  double tdeg,deg;
  deg=pi/180;
  for(;;){
    printf("l,m,tdeg ?\n");
    scanf("%d %d %lf",&l,&m,&tdeg);
    if(l < 0) break;
    t=tdeg*deg;
    /*                  printf("[0] %f %f\n",tdeg,t); /*D*/
    y=ylm(l,m,t);
    printf("L=%3d M=%3d theta=%6.1f Y_lm=%20.16f\n",l,m,tdeg,y);
  }
}

function2(){
  double t,y,y0,deg,tdeg, errmax,err;
  int l,m,n;
  char str1[10];
  deg=pi/180;
  errmax=0;
  while((n=scanf("%d %d %lf %lf",&l,&m,&tdeg,&y0))!=EOF){
    /* printf("scanf read %d values.\n",n); */
    if(l<0) break;
    t=tdeg*deg;
    y=ylm(l,m,t);
```

```
    strncpy(str1," ",10);
    err=fabs(y-y0);
    if(err >errmax) errmax=err;
    if( fabs(y-y0)>1.0e-12) strncpy(str1,"*",10);
    if( fabs(y-y0)>1.0e-8) strncpy(str1,"**",10);
    if( fabs(y-y0)>1.0e-4) strncpy(str1,"***",10);
    if( fabs(y-y0)>1.0e-1) strncpy(str1,"*****",10);
    printf("L=%3d M=%3d T=%6.1f YLM=%12.8f %12.8f err=%12.4e
   %s\n",l,m,tdeg,y,y0,y-y0,str1);
  }
  printf("max error =%12.4e",errmax);
}


double ylm(int l, int ma, double t){
  double c,s,r,z;
  int i,j,k,m,n,u,v;
  double f1,f2,f3,f4,fct,imax,n3;
  int lm,lmm;
  /*                  printf("[1] %d %d %f\n",l,ma,t); /*D*/
  if(l <0 || fabs(ma)>l ){
    printf("ylm argument error %12d %12d %16.8e",l,ma,t);
    z=0;
    return z;
  }
  if(ma < 0){
    m=-ma;
    /* fct=pow(-1,m);  to be changed */
    if(m % 2) fct=-1; else fct=1;
    /*    printf("m=%d %f\n",m,fct); /*D*/
   }
  else{
    m=ma;
    fct=1;
    /*    printf("%d fct=%f\n",m,fct); /*D*/
  }
  lmm=l-m;

  c=cos(t);

  if(fabs(c) < eps){   /* when t = 90 degree */
    if(lmm%2==0){
      /*      printf("lmm=%d\n",lmm);/*D*/
      f1=1;
      for(i=l+m-1; i>=2; i-=2 )f1*=i;
f2=1;
/* printf("f1=%f f2=%f\n",f1,f2); /*D*/
```

```
      for(i=lmm; i>=2; i-=2)f2*=i;
f3=4*pi/(2*l+1);
/*       printf("f2=%f f3=%f\n",f2,f3);/*D*/

      for(i=lmm+1;i<=l+m;i++)f3*=i;
if((l+m)/2 % 2) u=-1; else u=1;
        z=f1*fct/(f2*sqrt(f3))*u;
/* printf("z=%f\n",z);/*D*/

    }
    else{
      z=0;
    }
    return z;
  }

  s=sin(t);
  if(m >0) fct*=pow(s/2,m);
  if(lmm > 0) fct*=pow(c,lmm);
  f1=(2*l+1)/(4*pi);
  for(i=lmm+1;i<=l+m;i++) f1*=i;
  if(m % 2) v=-1; else v=1;
  fct*=sqrt(f1)*v;
  r=-pow((s/(2*c)),2);
  imax=lmm/2;
  f1=1;
  f2=1;

  for(i=2;i<=m;i++) f2*=i;
  f3=1;f4=1;
  s=f3*f4/(f1*f2);
  n3=lmm;
  for(i=1;i<=imax;i++){
    f2*=m+i;
    f3*=n3*(n3-1);
    n3-=2;
    f4*=r;
    s+=f3*f4/(f1*f2);
    f1*=i+1;
  }
  z=fct*s;
  /*  printf("%f\n",z);/*D*/
  return z;
}
```

プログラム3

```
/* ------------------------------------------------
 一様乱数を用いたモンテカルロ積分のテスト計算
------------------------------------------------* /

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define ITER_IN  10000
#define ITER_OUT 10000000

#include "unirnd3.c"

#include "mci_integrand1.c"

int main ( int argc , char * * argv ){
  int i,l,n,last_l=0;
  double x1,x2,y1,y2,z1,z2,sn,s1,s2,s1b,s2b,f,avr,sgm,error,
         estimated_error,r1,r2;

  double x1a,x1b,x2a,x2b,y1a,y1b,y2a,y2b,z1a,z1b,z2a,z2b,volume,
         volume_x,volume_y,volume_z;

  unirnd_seed = 1;\

  x1a=-6*wd1; x1b=6*wd1; x2a=-6*wd2; x2b=6*wd2;
  y1a=-6*wd1; y1b=6*wd1; y2a=-6*wd2; y2b=6*wd2;
  z1a=-6*wd1; z1b=6*wd1; z2a=-6*wd2; z2b=6*wd2;
  volume_x=(x1b-x1a)*(x2b- x2a);
  volume_y=(y1b-y1a)*(y2b- y2a);
  volume_z=(z1b-z1a)*(z2b- z2a);
  volume=(volume_x)*(volume_y)*(volume_z);
  s1b=0;
  s2b=0;
  for(l=0;l<ITER_OUT;l++){
    s1=0;
    s2=0;
    for(i=0;i<ITER_IN;i++){
      x1=  x1a+(x1b-x1a)*unirnd();
      x2=  x2a+(x2b-x2a)*unirnd();
      y1=  y1a+(y1b-y1a)*unirnd();
      y2=  y2a+(y2b-y2a)*unirnd();
      z1=  z1a+(z1b-z1a)*unirnd();
      z2=  z2a+(z2b-z2a)*unirnd();
      r1=sqrt(x1*x1+y1*y1+z1*z1);
      r2=sqrt(x2*x2+y2*y2+z2*z2);
```

```
      f=integrand(x1,x2,y1,y2,z1,z2);
      s1+=f;
      s2+=f*f;
    }
    sn=ITER_IN; sn*=l+1; s1b+=s1; s2b+=s2;
    avr=s1b/sn; sgm=sqrt(s2b/sn-avr*avr);
    f=volume*avr;
    error = f - 1.0;  /* The exact value of the integral is 1.0 */
    estimated_error = f*sgm/(avr*sqrt(sn));

    if(last_l <=0 || l > floor(last_l*1.01) || l == ITER_OUT-1){
      printf("%16.8e %16.8e %16.8e %16.8e \n"
             ,sn,error,fabs(error),estimated_error);
      last_l=l;
    }
  }
  return 0;
}
```

プログラム4

```
/*-------------------------------------------------------------
unirnd3.c : 一様乱数発生プログラム
-------------------------------------------------------------*/

#define unirnd_M1 101
#define unirnd_M2  59
#define unirnd_M3  43

const char *unirnd_program_version = "03/1/27a";

int unirnd_seed = 1;

double unirnd(){
  const unsigned int a1 =   39894229u , b1 = 1367130551u;
  const unsigned int a2 =    1664525u , b2 = 2654435769u;
  const unsigned int a3 = 1566083941u , b3 =  613566757u;
  static unsigned int v1,v2,v3,loop=0;
  static unsigned int store1 [ unirnd_M1 ], store2 [ unirnd_M2 ],
                      store3 [ unirnd_M3 ] ;
  static int ptr1, ptr2, ptr3, init = -1;
  static double f1,f2,f3;
  unsigned int u1,u2,u3;
  double x;
  int i;

  if(init) {
    v1=unirnd_seed;
    v2=unirnd_seed+0x00000FFF;
    v3=unirnd_seed+0x00FFFFFF;
    f1= 0.5/(double) 0x80000000;
    f2=f1*2/(1+sqrt(5.0));
    f3=f1/104723;
    init=0;
    fprintf(stderr,"unirnd3(ver.%s): seed=%u\n",unirnd_program_version
          ,unirnd_seed);
    for(i=0;i<unirnd_M1;i++){v1*=a1;v1+=b1;store1[i]=v1;}ptr1=unirnd_M1-1;
    for(i=0;i<unirnd_M2;i++){v2*=a2;v2+=b2;store2[i]=v2;}ptr2=unirnd_M2-1;
    for(i=0;i<unirnd_M3;i++){v3*=a3;v3+=b3;store3[i]=v3;}ptr3=unirnd_M3-1;
  }
Redo:loop++;
  v1 *= a1; v1 += b1;
  v2 *= a2; v2 += b2;
  v3 *= a3; v3 += b3;
```

```
  if(loop == 0) {
    for(i=0;i< 997;i++){v2 *= a2; v2 += b2;}
    for(i=0;i<1663;i++){v3 *= a3; v3 += b3;}
  }
  u1 = store1 [ ptr1 ]; store1 [ ptr1 ] = v1; ptr1=(ptr1+v1+v2+v3)
       % unirnd_M1;
  u2 = store2 [ ptr2 ]; store2 [ ptr2 ] = v2; ptr2=(ptr2+v3) % unirnd_M2;
  u3 = store3 [ ptr3 ]; store3 [ ptr3 ] = v3; ptr3=(ptr3+v2) % unirnd_M3;
  x=u1*f1+u2*f2+u3*f3;
  if(x>=1.0) x-=1.0;
  if(x<=0.0) goto Redo;
  return x;
}
```

プログラム5

```
/*-------------------------------------------------------------
 mci_integrand.c ：テスト計算用の被積分関数
-------------------------------------------------------------*/

const double wd1=1.0, wd2=2.0, wd3=0.3;

double integrand(double x1, double x2, double y1,double y2, double z1,
                 double z2){
  static double a1,a2,a3,nrmfct;
  static int first_call = 1;
  double t,f,pi,f_exact,t_x,t_y,t_z;

  if(first_call){
    first_call=0;
    a1=0.5/(wd1*wd1); a2=0.5/(wd2*wd2); a3=0.5/(wd3*wd3);
    pi=4*atan(1.0); f_exact = pi/sqrt(a1*a2+a2*a3+a3*a1);
    f_exact=f_exact*f_exact*f_exact; nrmfct=1/f_exact;
    printf("wd1=%f wd2=%f wd3=%f f=%f\n",wd1,wd2,wd3,1.0);
  }

  t_x=x1-x2;   t_y=y1-y2;   t_z=z1-z2;
  f=exp(-(a1*(x1*x1+y1*y1+z1*z1)+a2*(x2*x2+y2*y2+z2*z2)+
        a3*(t_x*t_x+t_y*t_y+t_z*t_z)))*nrmfct;
  return f;
}
```

プログラム6

```
/* ------------------------------------------------------
 ガウス乱数を用いた二体相互作用行列要素の計算
------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define ITER_IN  10000
#define ITER_OUT 1000

#include "gaurnd3.c"
#include "wave_integrand3.c"


int main ( int argc , char * * argv ){
  int i,l,n,last_l=0;
  double x1,x2,y1,y2,z1,z2,sn,s2,s2b,w,sgm,ff, estimated_error,error1;
  double pwb1,pwb2,pwa1,pwa2,refitg,pi,I;
  complex fow,f,error,avr,s1b,s1;
  printf("\n");
  printf("\n");

  gaurnd_seed = 2;

  pwb1=wd1*1.1;  pwa1=-1/(2*pwb1*pwb1);
  pwb2=wd2*1.1;  pwa2=-1/(2*pwb2*pwb2);
  pi=4*atan(1.0);

  refitg=8*pi*pi*pi*pwb1*pwb1*pwb1*pwb2*pwb2*pwb2;

  fprintf(stderr,"Integral of the reference function = %f\n",refitg);

  fprintf(stderr,"b1=%f  b2=%f\n",pwb1,pwb2);

  s1b.r=0;
  s1b.i=0;
  s2b=0;
  for(l=0;l<ITER_OUT;l++){
    s1.r=0;
    s1.i=0;
    s2=0;
    for(i=0;i<ITER_IN;i++){
      x1=  pwb1*gaurnd();
      x2=  pwb2*gaurnd();
      y1=  pwb1*gaurnd();
```

```
      y2=  pwb2*gaurnd();
      z1=  pwb1*gaurnd();
      z2=  pwb2*gaurnd();

      f=integrand(x1,x2,y1,y2,z1,z2);
      w=exp(pwa1*x1*x1+pwa2*x2*x2+pwa1*y1*y1+pwa2*y2*y2+
            pwa1*z1*z1+pwa2*z2*z2);
      fow.r=f.r/w;
      fow.i=f.i/w;
      /*  fprintf(stderr,"%f %f %f %f %f A\n",x1,x2,f,w,fow);*/
      s1.r+=fow.r;
      s1.i+=fow.i;
      s2+=fow.r*fow.r+fow.i*fow.i;
    }
    sn=ITER_IN; sn*=l+1; s1b.r+=s1.r; s1b.i+=s1.i; s2b+=s2;
    avr.r=s1b.r/sn;avr.i=s1b.i/sn;
    sgm=sqrt(s2b/sn-(avr.r*avr.r+avr.i*avr.i));
    f.r=refitg*avr.r;
    f.i=refitg*avr.i;
    ff=sqrt(f.r*f.r+f.i*f.i);
    I=refitg*sgm/sqrt(sn);
    if(l > floor(last_l*1.01) || l == ITER_OUT-1){
      printf("%16.8e %16.8e %16.8e %16.8e %16.8e %16.8e
             \n",sn,f.r,f.i,sgm,ff,I);
      last_l=l;
    }
  }
  return 0;
}
```

プログラム7

```
/-----------------------------------------------------------------
 wave__integrand3.c  :Gogny力のWigner項の相互作用行列要素を
                      表す積分の被積分関数の値を返す関数
-----------------------------------------------------------------*/
#include "wavef.c"
#include "ylm1.c"
#include "complex1.c"
#include <math.h>
complex cmult(complex x,complex y);
const double wd1=6.0, wd2=6.0, wd3=0.85;
complex integrand(double x1, double x2, double y1,double y2, double z1,
                  double z2){
  static int first_call = 1;
  double phi1,theta1, phi2,theta2,r1,r2,r3,x3,y3,z3,rr3,v;
  complex w1d,w1u,w2d,w2u,w3d,w3u,w4d,w4u,z1a,z2a,z3a,z4,z5,z6,z7,z8,
          za,zb,zc,zd,f;
  if(first_call==1){
    first_call=0;
    store_wavefunctions();
  }

 r1=sqrt(x1*x1+y1*y1+z1*z1);
 r2=sqrt(x2*x2+y2*y2+z2*z2);
 if(r1 != 0.0) { theta1=acos(z1/r1); phi1=atan2(y1,x1); }
          else { r1=1.0e-6; theta1=0.0; phi1=0.0;}
 if(r2 != 0.0) { theta2=acos(z2/r2); phi2=atan2(y2,x2); }
          else { r2=1.0e-6; theta2=0.0; phi2=0.0;}
  w1d=wavefunction(1,0,3,1,1,0,r1,theta1,phi1);
  w1u=wavefunction(1,0,3,1,1,1,r1,theta1,phi1);
  w2d=wavefunction(0,0,4,0,-1,0,r2,theta2,phi2);
  w2u=wavefunction(0,0,4,0,-1,1,r2,theta2,phi2);

  w1d.i=-w1d.i; w1u.i=-w1u.i; w2d.i=-w2d.i; w2u.i=-w2u.i;

  w3d=wavefunction(1,1,1,1,1,0,r1,theta1,phi1);
  w3u=wavefunction(1,1,1,1,1,1,r1,theta1,phi1);
  w4d=wavefunction(0,1,2,0,-1,0,r2,theta2,phi2);
  w4u=wavefunction(0,1,2,0,-1,1,r2,theta2,phi2);

  z1a=cmult(w1d,w2d); z2a=cmult(w3d,w4d); za=cmult(z1a,z2a);
  z3a=cmult(w1d,w2u);  z4=cmult(w3d,w4u);  zb=cmult(z3a,z4);
  z5=cmult(w1u,w2u);  z6=cmult(w3u,w4u);  zc=cmult(z5,z6);
  z7=cmult(w1u,w2d);  z8=cmult(w3u,w4d);  zd=cmult(z7,z8);

  x3=x2-x1;y3=y2-y1;z3=z2-z1;
```

```
  r3=sqrt(x3*x3+y3*y3+z3*z3);
  rr3=r3*r3;
  v=-402.4*exp(-(1.0/0.7*0.7)*rr3)-21.30*exp(-(1.0/1.2*1.2)*rr3);
  f.r=(za.r+zb.r+zc.r+zd.r)*v;/*real part*/

  f.i=(za.i+zb.i+zc.i+zd.i)*v;/*imaginary part*/

  return f;
}
```

プログラム8

```
/*-------------------------------------------------------------
wavef.c：3次元空間の与えられた点での波動関数の複素数値を返す関数
-------------------------------------------------------------*/

#define IMAX 1000
#define RMAX 20.0
#define HBSO2M 20.7355
#define EMAXN 0.0
#define EMAXP 8.0
#define NEUTRON 0
#define PROTON 1
#define LMAX 9
#define NMAX 9
int radwf(int N,int L,int J,int T);
int set_potential (int Nu,int Z, int A,int L,int J,int T);
double potential(double x);
int Nu=50, Z=50,A=100,J,T,L,N;
double Rc,h;
double wf[IMAX+1],energy;
double spectrum[2][LMAX+1][2][NMAX+1];
double *ptr_wf[2][LMAX+1][2][NMAX+1];
double mesh_spacing;
/* ------------------------------------------------------------ */

typedef struct {double r,i;} complex;  /* r: real part, i:imaginary part */

complex wavefunction(int q,int n,int l,int cls, int m,int ms, double r,
                     double theta, double phi);
double radial_wavefunction(double r);
double ylm(int l, int ma, double t);
/*****************************************************************/
complex wavefunction(int q,int n,int l,int cls, int m,int ms, double r,
                     double theta, double phi){
  double y,clb,rr;
  int i,ml;
  complex w;
  if(ms==0){              /* sigma=-1/2 */
    ml=(m+1)/2;           /* jz = lz - 1/2 */
    if(cls==0){
          clb=sqrt((l+0.5+m)/(2*l+1));
    }
    else{
         clb=sqrt((l+0.5-m)/(2*l+1));
    }
  }
```

```
  else{                /* sigma = 1/2 */
    ml=(m-1)/2;
    if(cls==0){
       clb=-sqrt((l+0.5-m)/(2*l+1));
    }
      else{
        clb=sqrt((l+0.5+m)/(2*l+1));
      }
  }
  i=r/mesh_spacing;
  if(ptr_wf[q][l][cls][n] == NULL){
    printf("ptr is NULL for q,l,cls,n=%d %d %d %d\n",q,l,cls,n);
    exit(1);
  }
  if(i<0||i>IMAX){
    y=0;
  }
  else{
    y=ptr_wf[q][l][cls][n][i]+
    (ptr_wf[q][l][cls][n][i+1]-ptr_wf[q][l][cls][n][i])*(r/mesh_spacing-i)
  }
  if(r<mesh_spacing*0.1)
      rr=mesh_spacing*0.1;
    else
      rr=r;
  y=y*clb*ylm(l,ml,theta)/rr;
  w.r=cos(ml*phi)*y;
  w.i=sin(ml*phi)*y;
  return w;
}
\newpage
```

プログラム9

```
/*-------------------------------------------------------------
complex1.c:2個の複素数の積を計算する関数
-------------------------------------------------------------*/


complex cmult(complex x,complex y){
  complex z;
  z.r=x.r*y.r-x.i*y.i;
  z.i=x.r*y.i+x.i*y.r;

  return z;
}
```

プログラム10

```
/*-----------------------------------------------------
gaurnd.c:ガウス乱数発生プログラム
-----------------------------------------------------*/
#define gaurnd_M1 59
#define gaurnd_M2 43

int gaurnd_seed = 1;

double gaurnd(){
  const unsigned int a1 =  1664525ul , b1 =  5ul ;
  const unsigned int a2 = 39894229ul , b2 = 13ul ;
  static unsigned int v1,v2,loop=0;
  static unsigned int store1 [ gaurnd_M1 ], store2 [ gaurnd_M2 ] ;
  static int ptr1, ptr2, state = -1;
  static double f1,f2,x2;

  unsigned int u1,u2;
  double x1,s,w;
  int i;

  if(state == 2){
    state=1;
    return x2;
  }
  if(state < 0) {
    v1=gaurnd_seed;
    v2=gaurnd_seed+0x0000FFFF;
    f1= 1.0/(double) 0x80000000;
    f2=f1/2-1;
    state=1;
    fprintf(stderr,"gaurnd(ver.%s): seed=%u\n",gaurnd_program_version
          ,gaurnd_seed);
    for(i=0;i<gaurnd_M1;i++){v1*=a1;v1+=b1;store1[i]=v1;}ptr1=gaurnd_M1-1;
    for(i=0;i<gaurnd_M2;i++){v2*=a2;v2+=b2;store2[i]=v2;}ptr2=gaurnd_M2-1;
  }
  do {
    Redo:loop++;
    if(loop == 0) {
      for(i=0;i< 997;i++){v1 *= a1; v1 += b1;}
      for(i=0;i<1663;i++){v2 *= a2; v2 += b2;}
    }
    v1 *= a1; v1 += b1;
    u1 = store1 [ ptr1 ] ; store1 [ ptr1 ] = v1; ptr1 = v1 % gaurnd_M1 ;
    v2 *= a2; v2 += b2;
    u2 = store2 [ ptr2 ] ; store2 [ ptr2 ] = v2; ptr2 = v2 % gaurnd_M2 ;
```

```
    x1=u1*f1+f2;
    x2=u2*f1+f2;
    s=x1*x1+x2*x2;
  } while(s > 1.0);
  w=sqrt(-2*log(s)/s);
  x1=x1*w;
  x2=x2*w;
  state=2;
  return x1;
}
```

プログラム11

```
/-------------------------------------------------------
メトロポリス法を用いたモンテカルロ積分のテスト計算
------------------------------------------------------- */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define ITER_IN  10000
#define ITER_OUT 100000000

#include "unirnd3.c"

#include "gaurnd3.c"

#include "mci_integrand3.c"

int main ( int argc , char * * argv ){
  int i,l,n,next_l=1,move,stay,itmp,iii,needcalc;
  double x1,x2,x1b,x2b, y1,y2,y1b,y2b,z1,z2,z1b,z2b,tx,ty,tz,f,f_sp,
         w,wb,fow,fow2,
         avr,sgm,error,esterr;
  double sn_move,sn_stay,s0,s1,s2,t0,t1,t2,mobility,mobility_sp;
  double pwb1,pwb2,pwb3,pwa1,pwa2,pwa3,dx,dy,dz,refitg,pi;
  double num_samples = 1e9;
  int itvl = 100;

  gaurnd_seed = 1;
  unirnd_seed = 1;

  pwb1=wd1*1.1;  pwa1=-1/(2*pwb1*pwb1);
  pwb2=wd2*1.1;  pwa2=-1/(2*pwb2*pwb2);
  pwb3=wd3*1.1;  pwa3=-1/(2*pwb3*pwb3);
  dx=pwb3*1.0;  dy=pwb3*1.0;  dz=pwb3*1.0;
  /*
  if(argc >= 2){
    itmp=atoi(argv[1]);
    if(itmp>0 && itmp < 100000) dx=pwb3*(itmp*0.01);
  }

  if(argc >= 3){
    itmp=atoi(argv[2]);
    if(itmp>0 && itmp < 1000000) itvl=itmp;
  }

  if(argc >= 4){
```

```
  itmp = atoi(argv[3]);
  if(itmp > 0 ) {
    unirnd_seed = itmp;
    gaurnd_seed = itmp;
  }
}
*/
pi=4*atan(1.0);
refitg=pi*pi*pi/(sqrt(pwa1*pwa2+pwa2*pwa3+pwa3*pwa1)*
                (pwa1*pwa2+pwa2*pwa3+pwa3*pwa1));

fprintf(stderr,"# mci_wlk_1.c : ver.03/1/29a\n");
fprintf(stderr,"# Integral of the weight function = %f\n",refitg);
fprintf(stderr,"# b1=%f b2=%f b3=%f initial dx=%f sampling
        interval=%d\n"  ,pwb1,pwb2,pwb3,dx,itvl);

t0=0; t1=0; t2=0; sn_move=0; sn_stay=0;
x1=pwb1*(2*unirnd()-1); x2=pwb2*(2*unirnd()-1);
y1=pwb1*(2*unirnd()-1); y2=pwb2*(2*unirnd()-1);
z1=pwb1*(2*unirnd()-1); z2=pwb2*(2*unirnd()-1); w=-2; iii=0;

for(l=0;l<ITER_OUT;l++){
  s0=0; s1=0; s2=0; move=0; stay=0;
  for(i=0;i<ITER_IN;i++){
    x1b=x1; x2b=x2; y1b=y1; y2b=y2; z1b=z1; z2b=z2; wb=w;
 /* x1+=dx*gaurnd(); x2+=dx*gaurnd(); */
    x1+=dx*(2*unirnd()-1); x2+=dx*(2*unirnd()-1);
    y1+=dy*(2*unirnd()-1); y2+=dy*(2*unirnd()-1);
    z1+=dz*(2*unirnd()-1); z2+=dz*(2*unirnd()-1);
    tx=x2-x1; ty=y2-y1; tz=z2-z1;
    w=exp(pwa1*x1*x1+pwa2*x2*x2+pwa3*tx*tx+pwa1*y1*y1+
          pwa2*y2*y2+pwa3*ty*ty+pwa1*z1*z1+pwa2*z2*z2+pwa3*tz*tz);
    if(w >= wb || w > wb*unirnd()){  /* Metropolis method */
 /* if(w > (w+wb) * unirnd()){  */   /*  Heat-Bath method */
      move++; needcalc=1;
    }
    else {
      stay++; x1=x1b; x2=x2b; y1=y1b; y2=y2b; z1=z1b; z2=z2b; w=wb;
    }

    if((++iii)==itvl){
      iii=0;
      if(needcalc){
        f=integrand(x1,x2,y1,y2,z1,z2); fow=f/w; fow2=fow*fow;
        needcalc=0;
      }
      s0++ ; s1+=fow ; s2 +=fow2 ;
```

```
      }
    }
    f_sp=refitg*s1/s0;
    mobility_sp = (double)move/((double)move+(double)stay);

    t0+=s0; t1+=s1; t2+=s2; sn_move+=move; sn_stay+=stay;
    mobility = sn_move/(sn_move+sn_stay);
    avr=t1/t0; sgm=sqrt(fabs(t2/t0-avr*avr));
    f=refitg*avr;
    error = f - 1.0;  /* The exact value of the integral is 1.0 */
    esterr = f*sgm/(avr*sqrt(t0));
    if(l == 0 || l >= next_l || l == ITER_OUT-1){
      printf("%14.6e %14.6e %12.4e %12.4e %f %f %f\n"
             ,t0,error,fabs(error),esterr,mobility,f_sp,mobility_sp);
      next_l=ceil(l*1.01+0.5);
      if(t0 >= num_samples) {printf("# N>=%e\n",num_samples);break;}
    }
  }
  return 0;
}
```