数値計算で学ぶ量子力学:
プログラミング言語 Julia による数値計算入門

永井佑紀

2018 年 11 月 18 日

目次

第 1 章

はじめに

1.1　量子力学と数値計算

量子力学では当然シュレーディンガー方程式の解の性質を調べることが重要である。しかしながら、シュレーディンガー方程式を手で解けるケースというのは限られている。さらに、解けるケースでも面倒な手計算（できるようになることは重要ではある）が必要であり、量子力学の学習ではひたすら微分方程式を解いているような状態に陥ってしまうことがある。昨今はコンピュータの高速化や Python、Julia 等モダンなプログラム言語の登場によって、シュレーディンガー方程式の解を**数値的に**調べることは容易になっている。そこで、シュレーディンガー方程式を数値的に解くことによって、量子力学について親しんでみよう。差分化やフーリエ変換やベッセル関数など様々な方法でシュレーディンガー方程式を解くことにより、計算機の中で量子力学を解くという感覚が得られるかもしれない。

最新の理論研究の現場においても、ここで記した手法と類似あるいは拡張した手法が使われている。例えば、第一原理計算と呼ばれる手法（密度汎関数法）は固体の性質をシュレーディンガー方程式に類似した方程式（コーンシャム方程式）を解いて調べることができる。その際には、方程式を様々な基底関数を用いて展開して解く。その中には、実空間で差分化して解く手法もある。

1.2　このノートについて

このノートは現在未完成である。随時書き足すつもりである。また、間違っている部分などがある可能性がある。なお、このノートに書かれたすべてのコードを `http://park.itc.u-tokyo.ac.jp/kato-yusuke-lab/nagai/samplecode.jl` に置いたので、Julia を立ち上げて

```
include("samplecode.jl")
```

とするか、端末で

```
julia samplecode.jl
```

とすれば実行して確かめることができる。なお、図は全部 PDF として出力される。

1.3　プログラミング言語 Julia

1.3.1　Julia とは

Julia とは数値計算を高速にかつ簡便に実行することができるプログラミング言語である。大きな特徴としては、

- Fortran や C のようにコンパイルをする必要がなく、Python のように実行できる
- にもかかわらず C や Fortran にせまるほど高速である
- 記法が数式の記述によく似ているため式をコードにするのが容易
- 行列の対角化や数値積分などの数値計算で使う基本的なパッケージを用意されている
- 導入にお金がかからない（コンパイラ代やソフトウエア代が無い）

などである。Julia は `http://julialang.org` からバイナリをダウンロードすれば Windows でも Mac でも Linux でも実行できる。その際 root 権限は必要ない。

コードを見ればわかるが、Julia の場合、Fortran や C で必要であったたくさんの「おまじない」や Lapack のインストールや呼び出しなどの煩雑なことを一切する必要がない。Python でも似たような形でシンプルにコードが書けるが、Python は For ループが遅いという数値計算として使うには問題となる点（工夫すれば速くなるが Python ならではのこの工夫を習得するのに時間がかかる）がある。この問題点のせいで、教科書に書いてあるようなアルゴリズムを Python にそのまま移植するととんでもなく遅くなってしまうことがある。Python はあらかじめわかっているアルゴリズムを呼び出すことにかけてはそのライブラリの豊富さとコミュニティの広さで圧倒的であるが、新しいアルゴリズムを書いたりする場合には、最適ではないと考えている。その点、Julia はアルゴリズムをそのままコードにするだけで速い。これは、「物理以外の余計なことを考えずに物理の結果が知りたい」という物理をやる人間にとって重要な欲求を満たす可能性のある言語であるということであり、有望であると考えている。

1.3.2 Julia の実行

Julia のインストールと実行については様々な web 上の解説があるので、ここでは載せない。今後ここに書き記す可能性はある。Julia を試すのに現状 (2018 年 10 月現在) 一番簡単だと思われるのは、JuliaBox(`https://juliabox.com`) にアクセスして、google のアカウントなどでログインすることである。それによって、インストールなしで web 上で Julia を実行することができる。このノートに記しているコードはすべて JuliaBox 上で実行できるはずである。

1.3.3 Julia の書き方

未完成。今後ここに書き記すつもりであるが、他の web サイトを参考にされたい。たとえば、「Julia で数値計算　その 1：コードサンプル～基本的計算編～」(`https://qiita.com/cometscome_phys/items/31d0b811345a3e12fcef`)

以下に、簡単なコードを乗せ、Julia がどの程度シンプルで数値計算に向いているかを示す。ここでは、Julia 1.0 を用いている。

四則演算等

Julia は型を自動的に推論してくれるので、整数、実数、複素数を定義して、普通に四則演算ができる (ソースコード 1.1)。コメントアウトは#を使う。全体をコメントアウトしたい場合には、#= と =#を使う。虚数は im である。興味深い点としては、ギリシャ文字や日本語を変数として普通に使うことができる点である。これによって、物理で出てくるギリシャ語の変数をそのまま使うことができ、可読性が良くなる。以下にもう少し細かく記述する。

足し算と引き算

足し算は + でそのままできる。引き算は-である。

```
a = 1
b = 2
c = a+b
d = a-b
```

もちろん、実数をいれることもできる。

```
a = 1.2
b = 2.4
```

```
c = a+b
d = a-b
```

ここで a や b には複素数をいれても構わない。

```
a = 1 + im
b = 2 - im
c = a+b
d = a-b
```

掛け算

掛け算は*の記号でできる。

```
a = 2
b = 3*a
```

ここで、数字と変数をくっつけて表記することも可能である。

```
a = 2
b = 3a
```

そのため、虚数の部分を

```
a = 1 + 2im
b = 2 - 4im
c = a+b
d = a-b
```

と書いても構わない。複素数同士の掛け算も*をつかって普通に計算できて、

```
a = 1 + 2im
b = 2 - 4im
c = a*b
```

となる。

割り算

割り算は\でできる。

```
a = 1 + 2im
b = 2 - 4im
c = a/b
```

ポイントは、これは実数、これは複素数、と定義することなしに、演算ができるということである。このメリットは、行列の積にも同じように*が使える、ことをみれば理解できるかもしれない。

整数を割って整数を出したい場合には、div というものを使う。

```
a = 12
b = 4
c = div(a,b)
```

あまりを出したい場合には、

```
a = 12
b = 5
d = a %b
```

となる。(ソースコード 1.2 参照)。

なお、同じ変数に何かを足したり引いたりしたいときには、

```
a = 12
a += 3
a -= 10
a *= 100
a /= 25
```

とイコールにつけることで実行できる。

累乗

累乗は

```
a = 3
b = a^2
```

と^の記号でできる。

その他の演算

その他に、

```
a = 2.0
b1 = sin(a)
b2 = cos(a)
b3 = log(a)
```

```
b4 = tanh(a)
b5 = atan(a)
b6 = exp(a)
```

のような三角関数関連はそのまま使える。そして、複素数を入れても

```
a = 2.0+ 3im
b1 = sin(a)
b2 = cos(a)
b3 = log(a)
b4 = tanh(a)
b5 = atan(a)
b6 = exp(a)
```

計算することができる。また、円周率 π もそのまま使えるので、

```
a = 2.0
b = sin(a*π)
```

などもできる。

```
# 整数
a = 1
b = 2
c = a + b
println(c)
d = a*b
println(d)
#=
以下は
実数の場合
=#
a = 1.0
b = 2.2
c = a + b
println(c)
d = a*3b
println(d)
# 整数の複素数
a = 1+2im
b = 3+4im
c = a + b
println(c)
d = a*3b
println(d)
# 実数の複素数
a = 1.0+2.0im
b = 3.0+4.0im
c = a + b
println(c)
d = a*3b
println(d)
#= 円周率πも普通に使えるし、ギリシャ文字も使用可能。
cosやsinもそのまま使用可能
=#
α = π
β = 3π
γ = cos(π+β)
```

```
println(γ)
#= 日本語だって使える
=#
りんご = 3
ばなな = 4
総数 = りんご +ばなな
println("りんごの数 $りんご　ばなな の数 $ばなな　総数は$総数")
あたらしいりんご = 3
総数 += あたらしいりんご #+=で追加できる。
println("総数は",総数,"となった")
```

ソースコード 1.1　四則演算他

```
# 結果は実数になる
a = 12
b = 4
c = a/b
println(c)
# 結果は整数になる
a = 12
b = 4
c = div(a,b)
println(c)
# 割り切れない場合
a = 12
b = 5
c = div(a,b)
println(c)
# あまりの計算
a = 12
b = 5
d = a %b
println(c)
```

ソースコード 1.2　割り算

関数と多重ディスパッチ

sin や cos は Julia でただ呼び出せば実行できる。同様に、自前の関数を作ることができる。これが function である。function については、ソースコード 1.3 にまとめた。function と書いて end で閉じる方法と、一行で書く方法がある。面白い点としては、function の定義において、変数の型（実数だとか複素数だとか）を指定していないことである。つまり、変数に何がきても、そのまま計算ができる。これは、実数と複素数をわける必要がないことを意味しており、物理で使う数値計算では非常に役に立つ。一方、変数の引数が異なると異なる挙動をさせたい場合もあるだろう。たとえば、複素数のときだけ 2 π を足しておきたいとか。そういうときは、ソースコード 1.4 のようにすればよい。ここで、二つ目は型を明示している。ここで、function の名前が同じなことを着目してほしい。実際にこの関数を使うときに、変数の型を意識せずに、自動的に該当する関数 tasu を呼んでくれる。このような機能を「多重ディスパッチ」と呼ぶ。実は、四則演算の +、-、*や/も同じようになっており、型に応じて必要な関数を呼んでいるのである。

```
# 足し算
function tasu(a,b)
    c = a+b
    return c
end
```

```
# 足し算と掛け算
function tasukakeru(a,b)
    c = a+b
    d = a*3b
    return c,d
end
a = 1.0
b = 2.2
c = tasu(a,b)
println(c)
c,d = tasukakeru(a,b)
println("$c,$d")
a = 1.0 +3im
b = 2.2
c = tasu(a,b)
println(c)
c,d = tasukakeru(a,b)
println("$c,$d")
# functionは一行で書くこともできる。
tasu(a,b) = a+b
a = 1.0
b = 2.2
c = tasu(a,b)
println(c)
```

ソースコード 1.3　function の使い方

```
# 足し算
function tasu(a,b)
    c = a+b
    return c
end
# 足し算
function tasu(a::Complex,b)
    c = a+b+2π
    return c
end
a = 1.0
b = 2.9
c = tasu(a,b)
println(c)
a = 1.0+0.0im
b = 2.9
c = tasu(a,b)
println(c)
```

ソースコード 1.4　function の使い方

行列

物理系の数値計算で避けて通れないのは、行列である。Julia では、配列の添字は 1 から始まる。そして、配列は一番左側の添字が回ってからそのさらに左側の添字が回るような形でメモリに格納されているため、左側の添字が回るようにアクセスするとメモリアクセス的に効率的である。なお、このメモリ配置方法は Fortran と同じ順番であり、c とは逆になっている。行列の定義はソースコード 1.5 にあるように、いくつかのやり方ができる。ここで、typeof というのは、その変数の型を返す関数で、`Array{Int64,1}`だと整数の 1 次元配列、`Array{Int64,2}`だと整数の 2 次元配列である。

```
# 行列の定義
A = [1 2
3 4]
println(A)
println(typeof(A))
```

```
# 別の定義
A = [1  2;3  4]
println(A)
println(typeof(A))
# 要素がゼロの2x2行列
A = zeros(Int64,2,2)
A[1,1] = 1
A[1,2] = 2
A[2,1] = 3
A[2,2] = 4
println(A)
println(typeof(A))
# 要素が1の2x2行列
A = ones(Int64,2,2)
println(A)
println(typeof(A))
# ベクトルの定義
A = [1,2] #これはベクトル。つまり、1次元配列
println(A)
println("ベクトル ",typeof(A))
A = [1 2] #これは上記の
#=
A = [1 2
3 4]
の3,4を削ったかたちなので、1x2行列 =#
println(A)
println("2次元配列 ",typeof(A))
A = [1;2] #これは2x1行列
println(A)
println(typeof(A))
```

ソースコード 1.5　行列の定義

行列の定義の種類

ここで、

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \tag{1.1}$$

という行列を Julia で定義してみよう。一番素朴な書き方としては、

```
A = [
1 2
3 4
]
```

だろう。数字の間はスペースで区切ってある。ただ並べれば、2x2 の行列となる。これを使えば、縦ベクトルと横ベクトルは

```
a = [
5
6
]
b = [
7 8
```

```
]
```

と書ける。上で定義したのは片方の次元が 1 であるベクトルであるが、普通のベクトルとしては、

```
a = [5,6]
```

とする方法もある。

毎回直接書く他には、あらかじめサイズを宣言することもできる。その場合には、

```
A = zeros(3,3)
```

と書けば、要素がすべて 0 の 3×3 の行列を定義したことになる。これに値を入れたければ、

```
A[1,2] = 4
```

などとすればよい。ここで、zeros(n,m) と書くと $n \times m$ 行列で要素が倍精度実数の行列を定義したことになる。そのため、A[1,2] = 4 と右辺が整数になっていても、自動的に 4 は倍精度実数であるとしてくれる。もし、整数の行列が欲しい場合には、

```
A = zeros(Int64,3,3)
```

と型を先頭で指定すればよい。Int64 は整数を意味しており、Float64 は倍精度実数である。複素数が入りうる行列を定義したければ、ComplexF64 とすればよい。

また、要素が全て 1 の行列や、要素がすべて 0 から 1 までの乱数でできた行列などは、

```
A = ones(3,3)
B = rand(3,3)
```

で作ることができる。0 で初期化したくない場合、値を未定義とする行列を作るには、

```
A = Matrix{Float64}(undef,3,3)
```

とすれば作ることができる。

行列の演算

次は、行列の演算についてみてみる。行列同士の積は*を使うだけでよい：

```
A = [
1 2
3 4
]
```

```
B = [
10 20
30 40
]
C = A*B
```

これは数学的な意味での行列の積になっている。同じように、行列とベクトルの積は

```
A = [
1 2
3 4
]
b = [10,20]
C = A*b
```

と*を使って計算できる。これは、*を使う際に、Julia がそれぞれの型に合わせて適切な演算を呼び出しているからできることであり、これを「多重ディスパッチ」と呼ぶ。その結果、型ごとに異なる命令を呼び出す必要がなくなり、コードの見た目がシンプルになり、バグが入りにくくなる。

さらに、行列の指数関数

$$\exp A = \sum_{n=0}^{\infty} \frac{A^n}{n!} \tag{1.2}$$

なども

```
A = [
1 2
3 4
]
C = exp(A)
```

でできる。

行列と普通の数を足す場合はどうすればいいだろうか。通常、行列と普通の数を足す場合には、「それぞれの行列要素に数を足す」という演算を考えていると思われる。このような「要素のそれぞれに演算する」ときには、演算する関数に . をつければよい。たとえば、足し算ならば、

```
A = [
```

```
1 2
3 4
]
C = A .+ 1
```

とすればよい。同様に、行列の要素に sin 関数をかける、などは、

```
A = [
1 2
3 4
]
C = sin.(A)
```

となる。したがって、行列の各要素に exp をかけるのと行列の指数関数は異なっているので、

```
A = [
1 2
3 4
]
C = exp(A)
D = exp.(A)
```

は異なっている。

ドット. は自前の関数にも使えるので、

```
A = [1 2
3 4]
B = [10 20
30 40]
function tasu(a,b)
    c = a+b
    return c
end
C = tasu.(A,B)
```

とすれば、それぞれの要素に対した tasu を演算させたことになる。

行列のコピー

行列のコピーにもいくつか方法があり、

```
# コピーする
B = copy(A)
# 別の方法。配列次元がわかっているならこちらのがみやすいかもしれない。
B = A[:,:]
```

となる。なお、

```
B = A
```

としてしまうと、B と A が同一のメモリをさし示すことになり、A を変えると B も変わってしまう。例えば、

```
println("A: $A")
B = A
println("A: $A")
println("B: $B")
A[1,1] = 100
println("A: $A")
println("B: $B")
```

とすると、A も B も変更される。ここは注意。

一部取り出し

二次元配列として行列を持っているとき、その一部を取り出したいことがある。そんな時は、

```
A = [1 2
3 4]
b = A[:,1]
A = A*b
```

とすれば 2 次元配列から 1 次元配列を取り出すことができる。

ループ

対角化し固有値固有ベクトルを求める

連立方程式を解く

疎行列を使って問題を解く

第 2 章

時間に依存しない 1 次元シュレーディンガー方程式:有限系ポテンシャル問題

現実世界では空間は 3 次元であるが、簡単のため、1 次元シュレーディンガー方程式を考えることにする。ここでは、時間に依存しない 1 次元シュレーディンガー方程式を解いてみよう。数値計算的な考え方でシュレーディンガー方程式を眺めてみる。

2.1 1 次元シュレーディンガー方程式

一次元系のシュレーディンガー方程式は、

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+V(x)\right)\psi(x)=\epsilon\psi(x) \tag{2.1}$$

と書ける。この方程式は二階微分方程式なので、一般解には二つの未定定数が含まれ、それらの定数を決定するためには、この方程式の他に二つの方程式が必要となる。

2.2 標準的なやり方

2.2.1 未定定数

これをはっきりさせるため、一番簡単な場合 $(V(x)=0)$ を考えてみよう。この時のシュレーディンガー方程式は

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x)=\epsilon\psi(x) \tag{2.2}$$

となる。この方程式の解として指数関数：

$$\psi(x) = e^{ikx} \tag{2.3}$$

を仮定すると、

$$\frac{\hbar^2 k^2}{2m} e^{ikx} = \epsilon e^{ikx} \tag{2.4}$$

となるので、

$$\frac{\hbar^2 k^2}{2m} = \epsilon \tag{2.5}$$

すなわち、

$$k = \pm \frac{\sqrt{2m\epsilon}}{\hbar} \tag{2.6}$$

が得られる。以上から、二つの関数

$$\exp\left[i\frac{\sqrt{2m\epsilon}}{\hbar}x\right], \exp\left[-i\frac{\sqrt{2m\epsilon}}{\hbar}x\right] \tag{2.7}$$

が方程式の解であることがわかった。そして、容易に確認出来るように、二つの解を足した解も解である。つまり、

$$\psi(x) = C_1 \exp\left[i\frac{\sqrt{2m\epsilon}}{\hbar}x\right] + C_2 \exp\left[-i\frac{\sqrt{2m\epsilon}}{\hbar}x\right] \tag{2.8}$$

も解である。よって、解は C_1 と C_2 という未定定数を二つ持つことがわかった。これは、上述したように、方程式が二階微分方程式であるからである。また、エネルギー ϵ には、現時点では何も条件が課されていないため、任意の実数を取ることができる。そして、もし $\epsilon < 0$ であれば、k は純虚数となる。

2.2.2　境界条件

上述の解の係数を決めたい。まず、波動関数の絶対値の二乗は粒子を見出す確率であるので、全空間で粒子を見出す確率は 1 にならなければならない。つまり、

$$\int dx |\psi(x)|^2 = 1 \tag{2.9}$$

である。これを規格化と呼ぶ。この条件により、未定定数のうち一つが求まることになる。ここでの積分範囲は、考えている系全体である。

A. 片側に壁が存在する場合

$x = 0$ に壁が存在する場合 $(\psi(x = 0) = 0)$ を考えよう。この場合、

$$C_1 = -C_2 \tag{2.10}$$

である必要があり、

$$\psi(x) = C_1 \left[\exp\left[i\frac{\sqrt{2m\epsilon}}{\hbar}x \right] - \exp\left[-i\frac{\sqrt{2m\epsilon}}{\hbar}x \right] \right] = 2iC_1 \sin\frac{\sqrt{2m\epsilon}}{\hbar}x \tag{2.11}$$

となる。ここで、エネルギー ϵ が負の時、sin 関数は sinh 関数となり、$x \to \infty$ で値が発散してしまう。従って、

$$\epsilon \geq 0 \tag{2.12}$$

という条件が存在する。そして、エネルギーは連続の値を取ることができる。

B. 両側に壁が存在する場合

$x = 0$ と $x = L$ に壁が存在する場合 $(\psi(x = 0) = \psi(x = L) = 0)$ を考えよう。片側の壁の条件は全く同じなので、

$$C_1 = -C_2 \tag{2.13}$$

であり、解は

$$\psi(x) = C_1 \left[\exp\left[i\frac{\sqrt{2m\epsilon}}{\hbar}x \right] - \exp\left[-i\frac{\sqrt{2m\epsilon}}{\hbar}x \right] \right] = 2iC_1 \sin\frac{\sqrt{2m\epsilon}}{\hbar}x \tag{2.14}$$

である。しかし、この解は $\psi(x = L) = 0$ を常に満たすわけではない。常に満たすためには、

$$\frac{\sqrt{2m\epsilon}}{\hbar}L = n\pi \tag{2.15}$$

である必要がある。ここで、n は 1 以上の整数である。もし、$n = 0$ なら解は常に 0 になってしまう。よって、

$$\epsilon = n^2\frac{\hbar^2\pi^2}{2mL^2} \tag{2.16}$$

となる。

補足：エネルギーが実数である意味

実は、上述のような解き方では ϵ は複素数であっても構わない。これは、時間に依存しないシュレーディンガー方程式のみを考えているからである。もともとの時間依存するシュレーディンガー方程式は

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t)=\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}+V(x)\right)\Psi(x,t) \tag{2.17}$$

であった。ここで、ポテンシャル $V(x)$ が時間に依存しないとすれば、この偏微分方程式は変数分離することができて、解は

$$\Psi(x,t)=f(t)\psi(x) \tag{2.18}$$

と書け、微分方程式は

$$\frac{1}{f(t)}i\hbar\frac{\partial}{\partial t}f(t)=\frac{1}{\psi(x)}\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}+V(x)\right)\psi(x)=const. \tag{2.19}$$

となり、右辺の定数を ϵ とおけば、

$$f(t)=\exp\left(-i\frac{\epsilon}{\hbar}t\right) \tag{2.20}$$

が得られる。もし、ϵ が複素数であり、虚部が含まれれば、$f(t)$ は時間につれて発散あるいは減少する指数関数となり、解は安定しない。つまり、定常状態の解にはならない。よって、ϵ は実数である必要がある。

2.3　行列とベクトルを用いた解き方

上述したやり方だと、解の形を指数関数に仮定した。より複雑な問題に対応するためには、どのようにやるのが良いだろうか。まず、シュレーディンガー方程式を

$$H\psi(x)=\epsilon\psi(x) \tag{2.21}$$

と書く。ここで、

$$H=\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+V(x)\right) \tag{2.22}$$

である。このような形で書くと、シュレーディンガー方程式は、ハミルトニアン H に対する固有値問題であることがわかる。そして、エネルギーは固有値であり、対応する解は固有関数である。異なる固有値に属する固有関数は直交し、波動関数の二乗は確率を表す

ため、規格化されている。つまり、ある固有値 ϵ_n に属する解を $\psi_n(x)$ とすると、

$$\int dx\psi_n(x)^*\psi_m(x) = \delta_{nm} \tag{2.23}$$

となる。これは、直交規格化条件である。ここで、δ_{nm} はクロネッカーのデルタと呼ばれ、$n = m$ の時 1、$n \neq m$ の時 0 となる関数である。

2.3.1 基底の変換とベクトル表示

さて、シュレーディンガー方程式の解をフーリエ変換で求めてみよう。ある関数 $\psi(x)$ のフーリエ変換は

$$\psi(x) = \int dke^{ikx}c_k \tag{2.24}$$

と書ける。これを $V(x) = 0$ のシュレーディンガー方程式に代入すると、

$$\int dk\frac{\hbar^2k^2}{2m}e^{ikx}c_k = \epsilon\int dke^{ikx}c_k \tag{2.25}$$

となり、

$$\int dxe^{-ik'x}\int dk\frac{\hbar^2k^2}{2m}e^{ikx}c_k = \epsilon\int dxe^{-ik'x}\int dke^{ikx}c_k \tag{2.26}$$

$$\frac{\hbar^2k^2}{2m}c_k = \epsilon c_k \tag{2.27}$$

となるので、

$$\epsilon = \frac{\hbar^2k^2}{2m} \tag{2.28}$$

であれば、解となることがわかる。

次に、フーリエ変換の積分を和とみなすと、幅 dk、高さ $e^{ikx}c_k$ の長方形の面積の和となり、

$$\psi(x) = \sum_k c_k e^{ikx} \tag{2.29}$$

となる。ここで、dk は c_k を再定義することで押し付けた。さて、この形を眺めるとあることに気がつくかもしれない。線形代数において、ある行列 A とベクトル $\mathbf{v}$ の積でできたベクトル $\mathbf{g} = A\mathbf{v}$ の成分表示は

$$g_i = [A\mathbf{v}]_i = \sum_j A_{ij}v_j \tag{2.30}$$

と書ける。つまり、$\psi(x)$ を g_i、e^{ikx} を A_{ij}、c_k を v_j と見なせば、

$$\psi = U\mathbf{c} \tag{2.31}$$

と行列とベクトルを使った解を書くことができる。そして、異なる固有値に属する固有関数は

$$\psi_n = U\mathbf{c}_n \tag{2.32}$$

となる。これは、線形代数において基底を変換する演算そのものである。なお、この U はユニタリー行列である。

ここで、線形代数では添字 i はある離散的な値であったが、座標 x や k は連続的な値であることに注意する必要がある。ここでは詳しくは述べない。

2.3.2　シュレーディンガー方程式の行列表示

さて、座標 x を離散化し無数の格子点 x_i に分割したとする。この時、格子点 x_i での固有値 ϵ_n に対する解の値は $\psi_n(x_i)$ である。この $\psi_n(x_i)$ は、ベクトル表記ではどのように表せられるだろうか。実は、ベクトル $\mathbf{x_i}$:

$$\mathbf{x}_i^T = (0, 0, 0, 0, 0, \cdots, 0, 1, 0, 0, \cdots,) \tag{2.33}$$

を用意すれば良い。このベクトルの成分表示は

$$[\mathbf{x}_i]_\alpha = \delta_{i\alpha} \tag{2.34}$$

である。このベクトルを使うと、

$$\psi_n(x_i) = \mathbf{x} \cdot \psi_n \tag{2.35}$$

となる。また、ハミルトニアン $\hat{H}$ を格子点上で考える場合には、微分演算子を差分に変更する必要がある。ある点 x_i 近傍での関数 $\psi(x_i \pm a)$ は、テイラー展開より

$$\psi(x_i + a) = \psi(x_i) + \frac{d}{dx}\psi|_{x=x_i} a + \frac{1}{2}\frac{d^2}{dx^2}\psi|_{x=x_i} a^2 + \cdots \tag{2.36}$$

$$\psi(x_i - a) = \psi(x_i) - \frac{d}{dx}\psi|_{x=x_i} a + \frac{1}{2}\frac{d^2}{dx^2}\psi|_{x=x_i} a^2 + \cdots \tag{2.37}$$

となるので、

$$\frac{d^2}{dx^2}\psi|_{x=x_i} \sim \frac{\psi(x_i + a) - 2\psi(x_i) + \psi(x_i - a)}{a^2} \tag{2.38}$$

と微分を差分に直すことができる。一般的には、ある場所 x_i における微分演算子は、

$$\frac{d^2}{dx^2}\psi|_{x=x_i} \to \sum_j d_j \psi(x_j) \tag{2.39}$$

と書けるので、シュレーディンガー方程式は、

$$\sum_j \left(-\frac{\hbar^2}{2m} d_j + V(x_j)\delta_{ij} \right) \psi(x_j) = \epsilon \psi(x_i) \tag{2.40}$$

と書くことができる。この方程式は、先ほどと同様に行列とベクトルで表現することができて、

$$\hat{H}\psi = \epsilon\psi \tag{2.41}$$

となる。これがシュレーディンガー方程式の行列表示である。ここで、$\psi_n = U\mathbf{c}_n$ を使えば、

$$\hat{H}U\mathbf{c}_n = \epsilon U\mathbf{c}_n \tag{2.42}$$

$$U^+\hat{H}U\mathbf{c}_n = \epsilon U^+U\mathbf{c}_n \tag{2.43}$$

$$\hat{H}'\mathbf{c}_n = \epsilon\mathbf{c}_n \tag{2.44}$$

となる。

つまり、x で表現しようが k で表現しようが、演算子 $\hat{H}$ に対する固有値問題がシュレーディンガー方程式なのである。そこで、特定の ψ_n や $\mathbf{c}_n$ を使わずに、ベクトルとしてケットベクトル $|n\rangle$ を用いることで、

$$\hat{H}|n\rangle = \epsilon|n\rangle \tag{2.45}$$

とシュレーディンガー方程式を書くことができる。

2.3.3 座標表示との関係

上記の方程式と元の方程式との関係を見る。まず、単位行列 $\hat{I}$ をベクトル $\mathbf{x_j}$ で表現すると、

$$\hat{I} = \sum_j \mathbf{x_j}\mathbf{x_j}^T \tag{2.46}$$

と書ける。行列表示の方程式にこの単位行列を挟み、左から $\mathbf{x_i}^T$ をかけると、

$$\sum_j \mathbf{x_i}^T \hat{H} \mathbf{x_j} \mathbf{x_j}^T \psi = \epsilon \mathbf{x_i}^T \psi \tag{2.47}$$

$$\sum_j \left(-\frac{\hbar^2}{2m} d_j + V(x_j)\delta_{ij} \right) \psi(x_j) = \epsilon \psi(x_i) \tag{2.48}$$

となり、元の方程式が再現される。また、基底 ψ_n、$\mathbf{c}_n$ は、互いにユニタリー変換で結びつけられているため、ベクトル $\mathbf{x_j}$ も様々な基底で表現することができる。よって、これをケットベクトル $|x_i\rangle$ と書く。この時、ある基底で書かれた単位行列は

$$\hat{I} = \sum_j |x_i\rangle\langle x_i| \tag{2.49}$$

と書ける。ここで、$\langle x_i|$ はブラベクトルであり、$\langle\alpha|\beta\rangle$ はベクトルの内積を表す。以上から、

$$\sum_j \langle x_i|\hat{H}|x_j\rangle\langle x_j|n\rangle = \epsilon\langle x_i|n\rangle \tag{2.50}$$

が得られる。

さて、ここで離散化をやめて元の連続座標 x について考えると、波動関数 $\psi_n(x)$ は

$$\psi_n(x) = \langle x|n\rangle \tag{2.51}$$

であり、ハミルトニアンは

$$H\psi_n(x) = \int dx' \langle x|\hat{H}|x'\rangle\langle x'|n\rangle \tag{2.52}$$

となる。

つまり、**ベクトル $|n\rangle$ に対する座標表示の固有値方程式が、よく見る形のシュレーディンガー方程式だったのである。**そして、**うまく解けるように適当に基底を選んでシュレディンガー方程式を解くことができる。**それはフーリエ級数展開でも良いし、ベッセル関数でも良いし、チェビシェフ多項式でも良い。

2.4　1 次元シュレーディンガー方程式の数値的解法

2.4.1　無次元化

一番簡単なケースとして、1 次元シュレーディンガー方程式を差分化して解いてみよう。その前に、シュレーディンガー方程式を無次元化しておく。つまり、

$$\left(-\frac{1}{\sqrt{(2m/\hbar^2)}^2}\frac{d^2}{dx^2}+V(x)\right)\psi(x)=\epsilon\psi(x) \tag{2.53}$$

$$\left(-\frac{d^2}{dx'^2}+V(x')\right)\psi(x')=\epsilon\psi(x') \tag{2.54}$$

$$x'=x\sqrt{\frac{2m}{\hbar^2}} \tag{2.55}$$

としておく。これにより、計算が容易になる。以後、x' は x と置き直す。

x 軸を間隔 a の微小な座標点の集合に書き換えるとする。そして、二階微分を

$$\frac{d^2}{dx^2}\psi|_{x=x_i}\sim\frac{\psi(x_i+a)-2\psi(x_i)+\psi(x_i-a)}{a^2} \tag{2.56}$$

と差分化する。この時、シュレーディンガー方程式は

$$-\frac{1}{a^2}\left(\psi(x_{i+1})+\psi(x_{i-1})\right)+\left(\frac{2}{a^2}+V(x_i)\right)\psi(x_i)=\epsilon\psi(x_i) \tag{2.57}$$

となる。

座標点の数を N とする。境界条件としては、

$$\psi(x_{-1})=0 \tag{2.58}$$

$$\psi(x_{N+1})=0 \tag{2.59}$$

とする。つまり、両側が壁に囲まれている。この時、固有値はすでに導出しており、

$$\epsilon=n^2\frac{\pi^2}{L^2} \tag{2.60}$$

である。$L=(N+1)a$ である。それぞれの固有値における固有関数は

$$\psi_n(x)=C_1\sin\left(n\frac{x}{L}\right) \tag{2.61}$$

である。

2.4.2　数値計算と解析計算の比較

この時のハミルトニアンを作成する Julia のコードはソースコード 2.1 のようになる。

```
function make_H1d(N,a) #Nは離散点の数。aは空間の刻み幅
    mat_H = zeros(Float64,N,N) # ハミルトニアンの初期化
    for i in 1:N
        for dx in -1:1
            j = i + dx
            v = 0.0
            if dx == 0
                v = 2/a^2  # 中央
            elseif dx == 1
                v = -1/a^2 #+1方向
            elseif dx == -1
                v = -1/a^2 #-1方向
            end
            if 1 <= j <= N # 両側に壁があるためこの範囲にのみ制限
                mat_H[i,j] = v
            end
        end
    end
    return mat_H
end
```

ソースコード 2.1　ポテンシャルのない 1 次元シュレーディンガー方程式を差分化してえられる行列を作成する function

作成したハミルトニアンの固有値を計算すれば、シュレーディンガー方程式が解けたことになる。Julia では作った行列 A に対して eigen(A) とするだけで固有値固有ベクトルが得られる。固有値をプロットするコードはソースコード 2.2 となる。我々は解析解が式 (2.60) となることを知っているので、解析解と重ねて比較してみた。

```
using Plots # プロットするための準備
using LinearAlgebra # 対角化のルーチンeigenを呼ぶ準備
N = 1000 # 差分化した点の総数
a = 0.01 # 空間の刻み幅
mat_H = make_H1d(N,a) # ハミルトニアンの作成
ε, ϕ = eigen(mat_H) # 対角化し固有値と固有ベクトルを求める
sn = 200 # プロットする固有値の数を設定

# 解析解と比較するため解析解を用意
εa = zeros(Float64,N)
for n in 1:N
    εa[n] = n^2*π^2/(a*(N+1))^2 # 解析解を設定
end

# 固有値の最初の200個をプロット
plot([ε[1:sn],εa[1:sn]],label=["Numerical result","Analytical result"],marker=:
  circle)
```

ソースコード 2.2　行列を作り、対角化し、解析解を用意し、プロットして比較する。

得られた図を 2.1 に示す。

波動関数もプロットしてみよう。プロットするためのコードは 2.3 である。ここで、解析解とも比較している。

```
aϕ = zeros(Float64,N) # 解析解の用意
for i in 1:N
    xi = i*a
```

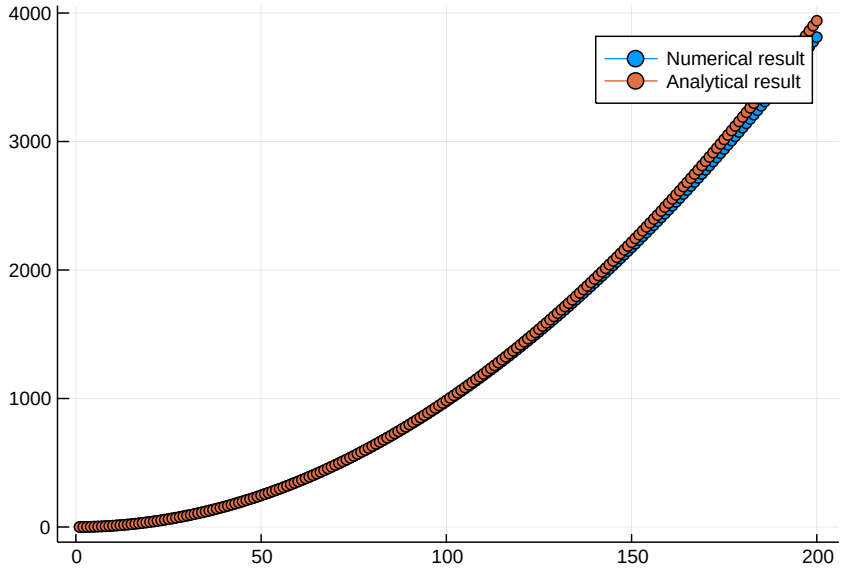

図 2.1　ポテンシャルのない 1 次元シュレーディンガー方程式を差分化して得られた解と解析解との比較

```
    aϕ[i] = sin(xi*π/((N+1)*a)) # 解析解
end
plot([ϕ[1:N,1],aϕ[1:N]],label=["Numerical result","Analytical result"])
```

ソースコード 2.3　波動関数のプロット

得られた図を 2.2 に示す。あれ、値がずれてしまった。これは、シュレーディンガー方程式の解は定数倍も解であるためである。 数値的に得られた解はソースコード 2.4 を計算すればわかるように、内積をとると 1 に規格化されている。解析解も同じように 1 に規格化してから比べてみよう。コードは 2.5 である。

```
sum(dot(ϕ[1:N,is],ϕ[1:N,is]))
```

ソースコード 2.4　規格化を調べる

```
aϕ = zeros(Float64,N)
for i in 1:N
    xi = i*a
    aϕ[i] = sin(xi*π/((N+1)*a))
end
C = sum(dot(aϕ[1:N],aϕ[1:N]))
aϕ = aϕ/sqrt(C)
plot([ϕ[1:N,1],aϕ[1:N]],label=["Numerical result","Analytical result"])
```

ソースコード 2.5　波動関数のプロット。解析解を規格化した

得られた結果を図 2.3 に示す。二つは完全に一致していることがわかる。

ポテンシャルがない場合を複数描くにはコード 2.6 とすればよくて、得られた結果は、図 2.4 となる。どれも綺麗な sin 関数となっている。

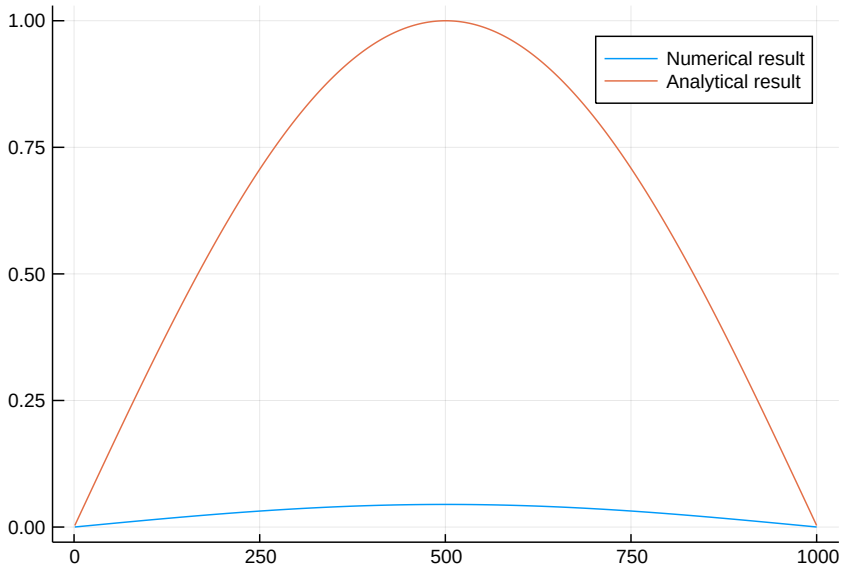

図 2.2　ポテンシャルのない 1 次元シュレーディンガー方程式の固有関数

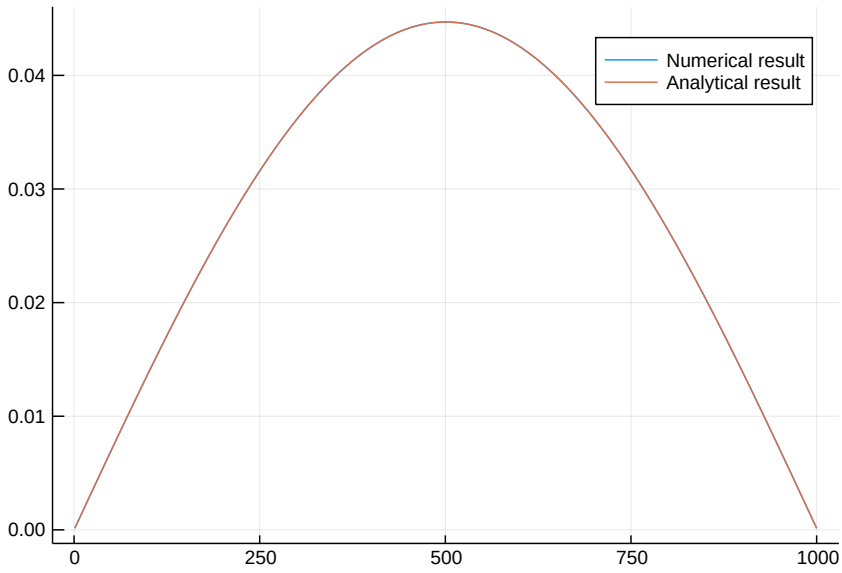

図 2.3　ポテンシャルのない 1 次元シュレーディンガー方程式の固有関数。規格化を揃えた。

```
1 plot(ϕ[1:N,1:6],label=["1st","2nd","3rd","4th","5th","6th"])
```

ソースコード 2.6　波動関数のプロット。下から 6 つのエネルギーに対応する波動関数

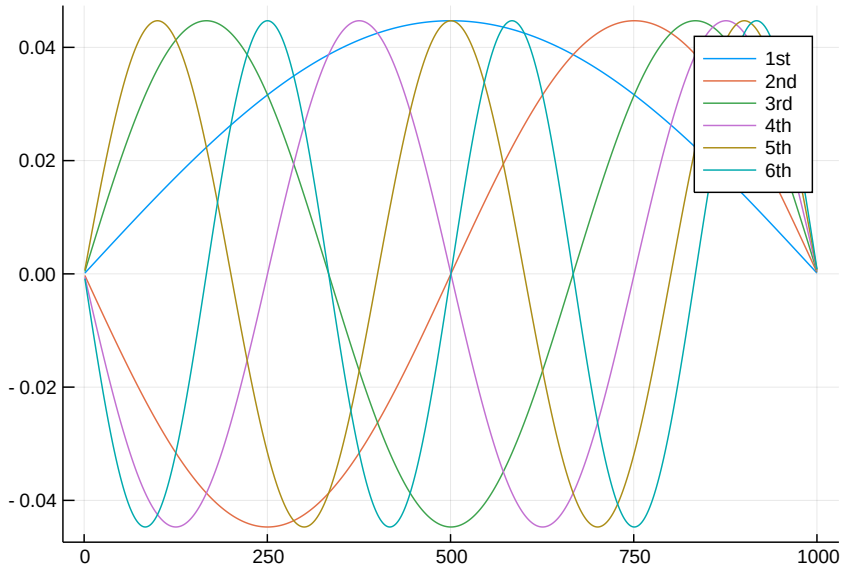

図 2.4　ポテンシャルのない 1 次元シュレーディンガー方程式の波動関数。下から 6 つのエネルギー固有値に対応する波動関数のプロット

2.4.3　高エネルギー領域でのずれとその原因

固有値をプロットした図 2.1 では固有値を低い順に 200 個表示している。この領域では、解析解と数値計算による解はよく一致しているように見える。さらに高いエネルギー領域まで比較してみよう。コードの sn を sn=200 から sn=1000 にすれば全固有値を表示することができる。図 2.5 を見ればわかるように、高エネルギー領域では盛大にずれている。なぜだろうか。

高エネルギーでずれる原因を考える。高エネルギーでの波動関数は、n が大きいことを意味しているので、sin 関数が非常に細かく振動していることを意味している。今、空間を差分化しているので、その間隔 a よりも短いスケールで振動している場合には、その関数を表現できない。これは、**差分化による解法では間隔 a より小さい長さスケールのものは記述できない**ことを意味している。つまり、波数 $k_{\mathrm{MAX}} = 1/a$ より大きな波数の物理は記述できない。この時のエネルギーは、

$$\epsilon = k_{\mathrm{MAX}}^2 = 1/a^2 \tag{2.62}$$

s である。従って、$a = 0.01$ の時は、$\epsilon > 10000$ の領域がずれることになる。もし、a = 0.1 とすると、さらに低いエネルギー領域からずれはじめる。これはコードにおける a を変更すれば確かめることができる。

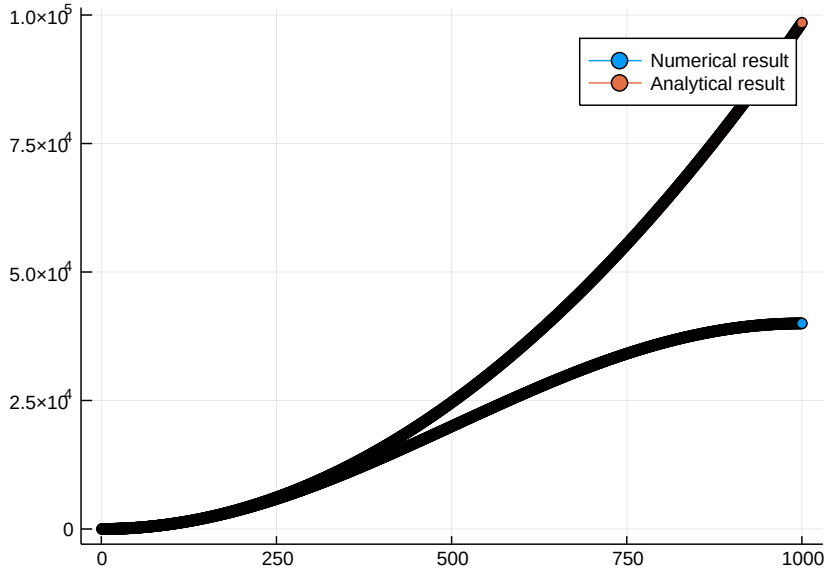

図 2.5　ポテンシャルのない 1 次元シュレーディンガー方程式を差分化して得られた解と解析解との比較。全固有値の比較。

2.5　ポテンシャルがある場合の 1 次元シュレーディンガー方程式

2.5.1　差分化による数値計算

差分化を使う場合には、ポテンシャルがない場合とコードはほとんど同じであり、ポテンシャルを考慮すればよいだけとなる。まず、ポテンシャルとして、矩形のポテンシャルを考えてみよう。コードはソースコード 2.7 となる。今考えているポテンシャルを図 2.6 に示す。

```
function calc_V(N,V0) # ポテンシャルを計算するfunction
    vec_V = zeros(Float64,N)
    dx = N/6
    for i in 1:N
        if N/2 - dx <= i <= N/2 + dx
            vec_V[i] = V0
        end
    end
    return vec_V
end

V0=1.0
vec_V = calc_V(N,V0) # ポテンシャルの計算
plot(vec_V[1:N],label="Potential")   # ポテンシャルをプロット
```

ソースコード 2.7　矩形ポテンシャルの定義とそのプロット

ハミルトニアンを作るには、最初に作った function にポテンシャルを加えればよくて、

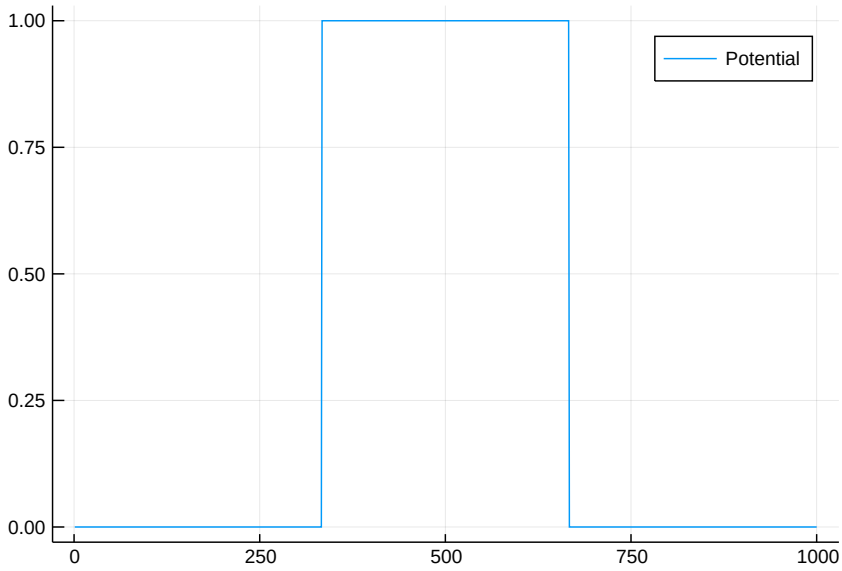

図 2.6　矩形ポテンシャルの例

ソースコード 2.8 となる。これを解くのはポテンシャルがない場合と完全に同じで、ただ eigen を呼べばよい。その時のコードはソースコード 2.9 となる。

```
function make_H1dv(N,a,V0)
    mat_H = zeros(Float64,N,N)
    vec_V = calc_V(N,V0) # ポテンシャルを計算する
    for i in 1:N
        for dx in -1:1
            j = i + dx
            v = 0.0
            if dx == 0
                v = (2/a^2 + vec_V[i])
            elseif dx == 1
                v = -1/a^2
            elseif dx == -1
                v = -1/a^2
            end
            if 1 <= j <= N
                mat_H[i,j] = v
            end
        end
    end
    return mat_H
end
```

ソースコード 2.8　ポテンシャルを引数とした 1 次元シュレーディンガー方程式のハミルトニアンの作成

```
N = 1000 # 差分の点の数
a = 0.01 # 刻み幅
V0 = 1.0 # ポテンシャル強度
mat_H = make_H1dv(N,a,V0) # ハミルトニアンの作成
ε,ϕ = eigen(mat_H) # 対角化し固有値と固有ベクトルを求める
plot(ϕ[1:N,1],label="Eigenfunction")
```

ソースコード 2.9　ポテンシャルを引数とした 1 次元シュレーディンガー方程式のハミルトニアンの作成

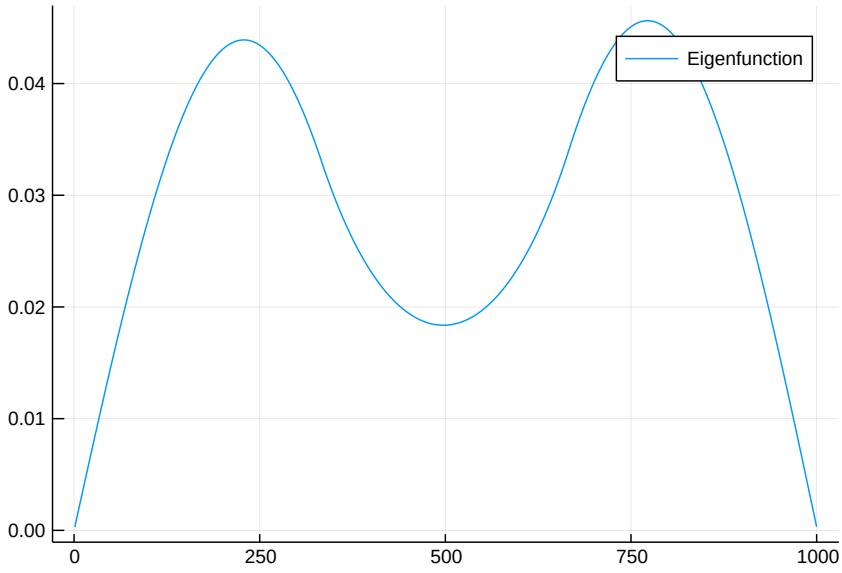

図 2.7　矩形ポテンシャルがあるときの最小エネルギーの波動関数

得られた結果は図 2.7 となる。ポテンシャルによって、波動関数が凹んでいることがわかる。当然、その二乗である存在確率も低くなっている。

ポテンシャル強度依存性と対称性

次に、ポテンシャルの強度を変えてみよう。コードはソースコード 2.10 となる。

```
N = 1000
a = 0.01
function gs1(N,a) # 強度依存性を調べるfunction
    groundstates = []
    labels = []
    for v in 1:10
        V0 = v*0.5
        mat_H = make_H1dv(N,a,V0)
        ε, ϕ = eigen(mat_H)
        println("Potential = ",V0," Minimum eigenvalue = ",ε[1]) # 最小エネルギーの表示
        push!(groundstates,ϕ[:,1]) # 最小エネルギーの波動関数をgroundstatesに格納
        push!(labels,string(V0)) # プロットラベルを作成
    end
    return groundstates,labels
end
groundstates,labels = gs1(N,a) # 強度依存性を調べる
plot(groundstates,label=labels)  # 結果をプロット
```

ソースコード 2.10　ポテンシャル強度依存性を調べる

得られた結果を図 2.8 に示す。本来、左右対称に出るべきもののような気がするが、なぜ非対称になってしまったのだろうか？ ポテンシャルを少し変え、ポテンシャルの中心を厳密に中央にしてみよう。つまり、ソースコード 2.7 の 5 行目の if 文の N/2 を (N+1)/2 とすればよい。その結果、波動関数は図 2.9 となる。ほんの少しの変更で結果が大きく変化した。これはなぜだろうか？その謎は、a の大きさにある。ポテンシャルをもとの定義

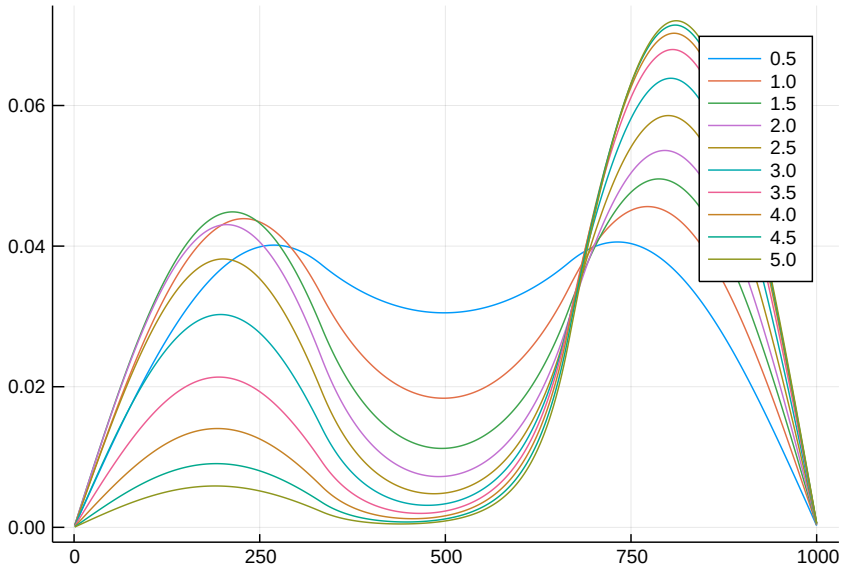

図 2.8　矩形ポテンシャルがあるときの最小エネルギーの波動関数のポテンシャル強度依存性

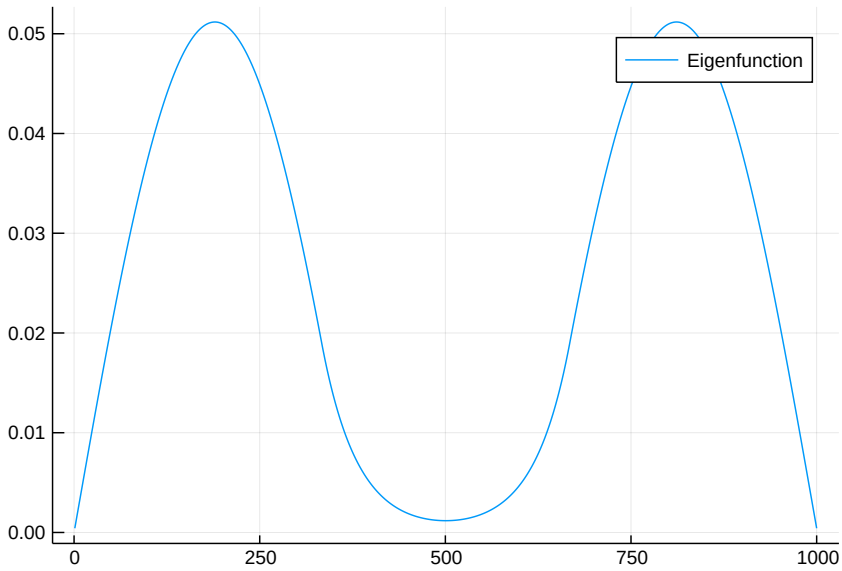

図 2.9　矩形ポテンシャルがあるときの最小エネルギーの波動関数。ポテンシャルの中心を系の中心に変更

にして、a の大きさを半分にしてみよう。系のサイズを揃えるために、a を半分にしたときには差分の点の数 N を倍にする必要がある。やってみるとわかるが、a が小さければ小さいほど差分化のエラーが小さくなり、その結果結果が改善されていく。つまり、非対称性は差分化したことによる数値計算のエラーが起源であることがわかる。

なぜ差分化でエラーが起きてしまったのだろうか？それは、ポテンシャルが矩形であり

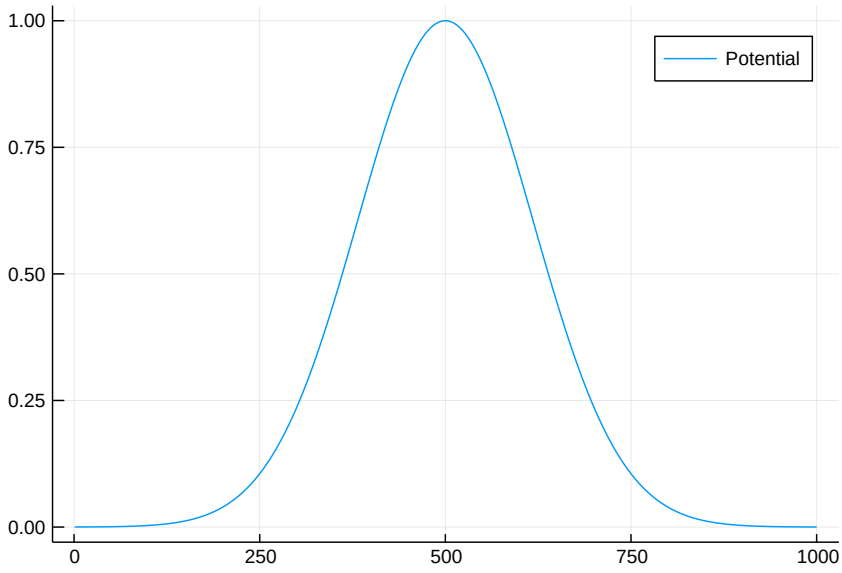

図 2.10　ガウス型ポテンシャルの例

非常に急峻に変化する関数だからである。差分化を行うと最小の長さスケールが a になり、それよりも小さなスケールで変化する関数を正しく記述することができない。高エネルギー領域で振動が大きい固有関数の固有値を再現できない理由と同じである。これを確認するために、ポテンシャルの形状をなだらかにしてみよう。ポテンシャルとしてガウス型関数を考える。ソースコードは 2.11 であり、その形は図 2.10 である。

```
function calc_V(N,V0)
    vec_V = zeros(Float64,N)
    dx = N/6
    center = (N)/2
    for i in 1:N
        vec_V[i] = V0*exp(-(i-center)^2/(dx^2))
    end
    return vec_V
end

N = 1000
V0=1.0
vec_V = calc_V(N,V0)
plot(vec_V[1:N],label="Potential")
```

ソースコード 2.11　ガウス型ポテンシャル

最小固有値の波動関数の強度依存性のソースコードは

```
N = 1000
a = 0.01
function gs2(N,a)
    groundstates = []
    labels = []
    for v in 1:10
        V0 = v*0.5
        mat_H = make_H1dv(N,a,V0)
        ε, ϕ = eigen(mat_H)
        println("Potential = ",V0," Minimum eigenvalue = ", ε[1])
        push!(groundstates, ϕ[:,1])
        push!(labels,string(V0))
```

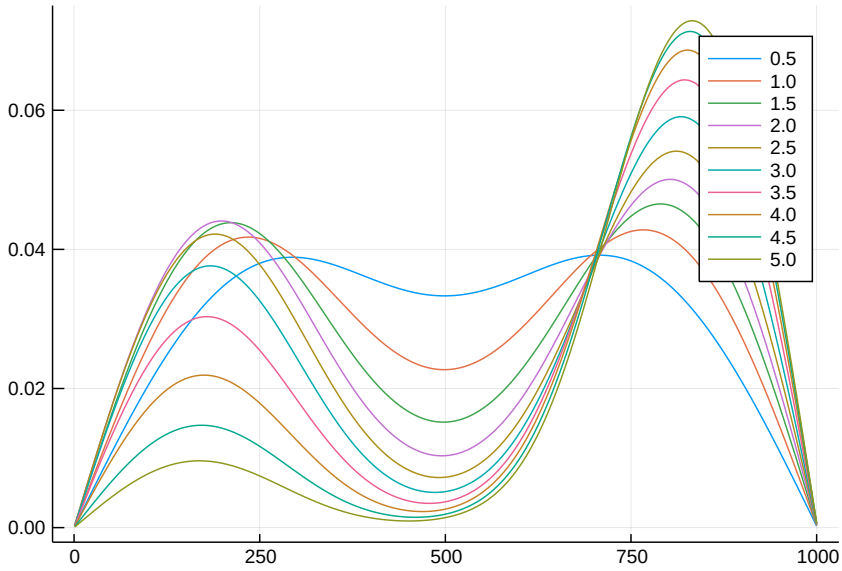

図 2.11　ガウス型ポテンシャルがあるときの最小エネルギーの波動関数のポテンシャル強度依存性

```
13      end
14      return groundstates,labels
15  end
16  groundstates,labels = gs2(N,a)
17  plot(groundstates,label=labels)
```

ソースコード 2.12　ガウス型ポテンシャル

であり、その図は図 2.11 である。矩形ポテンシャルのときよりも改善されたが、まだ強度が大きい時に非対称性が現れている。これは、ポテンシャルの急峻さが大きくなりすぎて現在使っている差分化の間隔では表すことができなくなったことを意味する。

最後に、ガウス型ポテンシャルの中心を (N+1)/2 に変更してみよう。その結果、図は図 2.12 となる。これで計算結果が対称となった。

以上より、数値計算をする場合、**系が持つ対称性を保つように数値計算を行う**（ポテンシャルの中心を系の中心に合わせるなど）ことが非常に重要であることがわかった。言い換えれば、すでにわかっている対称性があるならば、使った方が数値計算の精度がよくなることを意味している。

2.6　波数表示でのシュレーディンガー方程式

2.6.1　フーリエ変換

シュレーディンガー方程式を別の表示で書いてみる。ベクトルの言葉で言えば、$\psi(x)$ をベクトル ψ の x 成分である（離散化していれば x_i は i 番目の要素）とみなせば、任意

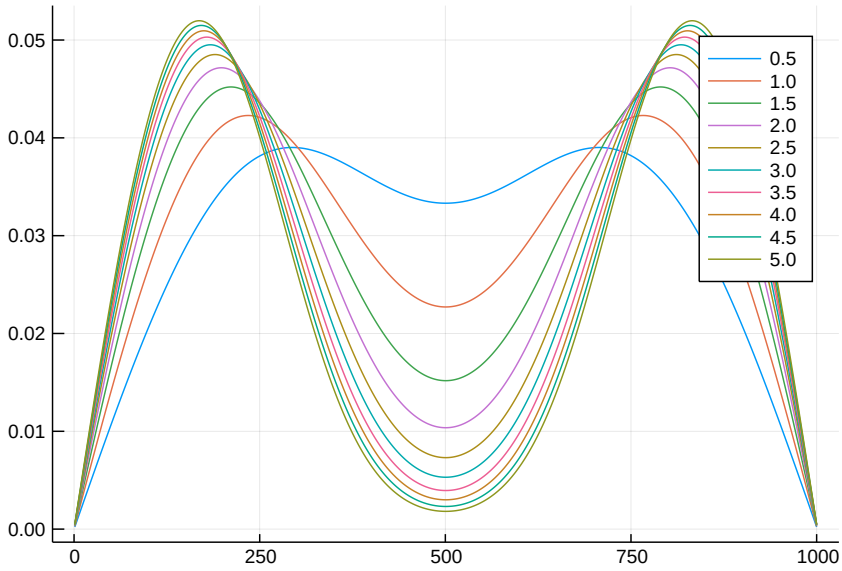

図 2.12　ガウス型ポテンシャルがあるときの最小エネルギーの波動関数のポテンシャル強度依存性。ポテンシャルの中心を厳密に系の中心とした場合。

の基底のベクトル $\mathbf{c}$ はユニタリー行列 U を用いて

$$\psi = U\mathbf{c} \tag{2.63}$$

と書ける。成分表示だと

$$\psi_i = \sum_k U_{ik} c_k \tag{2.64}$$

となり、U_{ik} を

$$U_{ik} = \exp(ikx_i) \tag{2.65}$$

とおけば、フーリエ変換となっていることがわかる。離散化しない場合には、

$$\psi(x) = \frac{1}{2\pi}\int dk \exp(ikx) c_k \tag{2.66}$$

である。このとき、

$$\frac{1}{2\pi}\int dk \left(\frac{\hbar^2}{2m}k^2 + V(x)\right)\exp(ikx)c_k = \epsilon\frac{1}{2\pi}\int dk \exp(ikx)c_k \tag{2.67}$$

となる。ここで、ポテンシャル $V(x)$ をフーリエ変換すると

$$V(x) = \frac{1}{2\pi}\int dq \exp(iqx) V_q \tag{2.68}$$

となるので、$\exp(-ik'x)$ をかけて x で積分すると

$$\int dx \frac{1}{2\pi} \int dk \left(\frac{\hbar^2}{2m} k^2 + \left(\frac{1}{2\pi} \int dq \exp(iqx) V_q \right) \right) \exp(i(k-k')x) c_k = \epsilon \int dx \frac{1}{2\pi} \int dk \exp(i(k-k')x) c_k \tag{2.69}$$

$$\frac{\hbar^2}{2m} k'^2 c_{k'} + \int dx \frac{1}{2\pi} \int dq \frac{1}{2\pi} \int dk V_q \exp(i(q+k-k')x) c_k = \epsilon c_{k'} \tag{2.70}$$

$$\frac{\hbar^2}{2m} k'^2 c_{k'} + \frac{1}{2\pi} \int dk V_{k'-k} c_k = \epsilon c_{k'} \tag{2.71}$$

となる。これが波数表示のシュレーディンガー方程式である。ここで、

$$\int dx \exp(i(k-k')x) = 2\pi \delta_{k,k'} \tag{2.72}$$

を用いた。これは、直感的には、位相の異なる波は全部足しあわせて消えてしまうけれど、位相の同じ波 $k = k'$ の時だけ残る、ということを表している。

2.6.2 両側に壁がある場合

これまで考えてきた、両側に壁がある場合を波数表示で解いてみよう。境界条件は、

$$\psi(x=0) = 0 \tag{2.73}$$

$$\psi(x=L) = 0 \tag{2.74}$$

である。解を

$$\psi(x) = \frac{1}{2\pi} \int dk \exp(ikx) c_k \tag{2.75}$$

と置く。このとき、

$$\frac{1}{2\pi} \int dk c_k = 0 \tag{2.76}$$

$$\frac{1}{2\pi} \int dk \exp(ikL) c_k = 0 \tag{2.77}$$

が境界条件となる。しかし、これらの境界条件をみたすように解を決めるのは難しい。なぜなら、それぞれの固有値に対応する固有関数ごとに、これらの境界条件を満たさなけれ

ばならないからである。もし、あらかじめ一般解が求められている場合、一般解を得たあとに境界条件によって係数 c_k を決めることができる。例えば、ポテンシャルがゼロの場合、一般解は

$$\psi_n(x) = C_1 e^{ikx} + C_2 e^{-ikx} \tag{2.78}$$

であり、ある固有値 ϵ があるとき、波数 k が定まる（k はいい量子数である、とも言う。）。しかし、ポテンシャルがある場合には、波数表示のシュレーディンガー方程式をみればわかるように、ある固有値に対する固有関数は複数の波数を持つ。どのような複数の波数をもつかは、解いてみないとわからないので、境界条件を満たすためにどのようにすればよいかがわからない。これを解決するために、解を少し書き換えてみよう。

$$\psi(x) = \frac{1}{2\pi}\int_0^\infty dk \exp(ikx)c_k + \frac{1}{2\pi}\int_{-\infty}^0 dk \exp(ikx)c_k \tag{2.79}$$

$$= \frac{1}{2\pi}\int_0^\infty dk \exp(ikx)c_k + \frac{1}{2\pi}\int_0^\infty dk \exp(-ikx)c_{-k} \tag{2.80}$$

$$= \frac{1}{2\pi}\int_0^\infty dk \left[\exp(ikx)c_k + \exp(-ikx)c_{-k}\right] \tag{2.81}$$

とする。さらに、

$$\psi(x) = \frac{1}{2\pi}\int_0^\infty dk \left[(\cos(kx) + i\sin(kx))c_k + (\cos(kx) - i\sin(kx))c_{-k}\right] \tag{2.82}$$

$$= \frac{1}{2\pi}\int_0^\infty dk \left[(c_k + c_{-k})\cos(kx) + i(c_k - c_{-k})\sin(kx))\right] \tag{2.83}$$

と書き換え、$a_k = c_k + c_{-k}$, $b_k = i(c_k - c_{-k})$ と新しい定数を定義すると、

$$\psi(x) = \frac{1}{2\pi}\int_0^\infty dk a_k \cos(kx) + \frac{1}{2\pi}\int_0^\infty dk b_k \sin(kx) \tag{2.84}$$

となる。この形で見ると、一つ目の境界条件

$$\psi(x = 0) = 0 \tag{2.85}$$

は

$$a_k = 0 \tag{2.86}$$

に、もう一つの境界条件

$$\psi(x = L) = 0 \tag{2.87}$$

は

$$k = n\frac{\pi}{L} \quad (n = 1, 2, 3, \cdots) \tag{2.88}$$

となる。そして、$k > 0$ として、k に関するシュレーディンガー方程式

$$\frac{\hbar^2}{2m}k^2 c_k + \frac{1}{2\pi}\int_{-\infty}^{\infty} dk' V_{k-k'} c_{k'} = \epsilon c_k \tag{2.89}$$

から $-k$ に関するシュレーディンガー方程式

$$\frac{\hbar^2}{2m}k^2 c_{-k} + \frac{1}{2\pi}\int_{-\infty}^{\infty} dk' V_{-k-k'} c_{k'} = \epsilon c_{-k} \tag{2.90}$$

を差し引くと、

$$\frac{\hbar^2}{2m}k^2 (c_k - c_{-k}) + \frac{1}{2\pi}\int_{-\infty}^{\infty} dk' (V_{k-k'} - V_{-k-k'}) c_{k'} = \epsilon (c_k - c_{-k}) \tag{2.91}$$

$$\frac{\hbar^2}{2m}k^2 (c_k - c_{-k}) + \frac{1}{2\pi}\int_{0}^{\infty} dk' (V_{k-k'} - V_{-k-k'}) c_{k'} + \frac{1}{2\pi}\int_{0}^{\infty} dk' (V_{k+k'} - V_{-k+k'}) c_{-k'} = \epsilon (c_k - c_{-k}) \tag{2.92}$$

$$\frac{\hbar^2}{2m}k^2 (c_k - c_{-k}) + \frac{1}{2\pi}\int_{0}^{\infty} dk' \left[(V_{k-k'} - V_{-k-k'}) c_{k'} - (V_{k+k'} - V_{-k+k'})(-c_{-k'}) \right] = \epsilon (c_k - c_{-k}) \tag{2.93}$$

$$\frac{\hbar^2}{2m}k^2 b_k + \frac{1}{2\pi}\int_{0}^{\infty} dk' \left[V_{k-k'} - V_{-k-k'} - V_{k+k'} + V_{-k+k'} \right] b_{k'} = \epsilon b_k \tag{2.94}$$

となる。

2.6.3 数値的に解く

さて、以上で問題を整理できたので、実際に数値的に解いてみよう。ポテンシャルは差分化して解いた時に用いたガウス関数:

$$V(x) = V_0 \exp\left[-\frac{(x - x_0)^2}{\xi^2} \right] \tag{2.95}$$

を用いる。ガウス関数のフーリエ変換

$$\int_{-\infty}^{\infty} dx e^{-iqx} e^{-ax^2} = \sqrt{\frac{\pi}{a}} \exp(-\frac{q^2}{4a}) \tag{2.96}$$

より

$$V(q) = \int_{-\infty}^{\infty} dx e^{-iqx} V(x) = V_0 \int_{-\infty}^{\infty} dx e^{-iq(x+x_0)} \exp\left[-\frac{1}{\xi^2}x^2\right] \tag{2.97}$$

$$= V_0 e^{-iqx_0} v(q) \tag{2.98}$$

$$v(q) = \sqrt{\pi\xi^2}\exp(-\frac{q^2\xi^2}{4}) \tag{2.99}$$

となる。よって、$V_{k-k'} - V_{-k-k'} - V_{k+k'} + V_{-k+k'}$ は

$$V_{k-k'} - V_{-k-k'} - V_{k+k'} + V_{-k+k'} = V_0 \Big(e^{-i(k-k')x_0} v(k-k') - e^{-i(-k-k')x_0} v(-k-k') - e^{-i(k+k')x_0} v(k+k') + e^{-i(-k+k')x_0} v(-k+k')\Big) \tag{2.100}$$

$$= V_0 \Big(e^{-i(k-k')x_0} v(k-k') + e^{i(k-k')x_0} v(k-k') - e^{-i(k+k')x_0} v(k+k') - e^{i(k+k')x_0} v(k+k')\Big) \tag{2.101}$$

$$= 2V_0 \left(\cos((k-k')x_0) v(k-k') - \cos((k+k')x_0) v(k+k')\right) \tag{2.102}$$

となる。

差分化の時とパラメータを合わせる。その結果、ポテンシャルの作成コードはソースコード 2.13 となる。となる。k' に関する積分は、k' が離散的にしかとれないため、

$$dk' \sim \frac{\pi}{L} \tag{2.103}$$

と近似し、積分を和に置き換える。そして、ハミルトニアンは、ソースコード 2.14 と書くことができる。

```
1  N=1000
2  a = 0.01
3  dx = N/6
4  ξ  = dx*a
5  center = (N+1)/2
6  x0 = center*a
7  function calc_vq(q,ξ,V0) # vq
8      vq = sqrt(π*ξ^2)*exp(-q^2*ξ^2/4)
9      return vq
10 end
11 function calc_Vkkp(k,kp,ξ,x0,V0) # $V_{k-k'}-V_{-k-k'} -V_{k+k'}+V_{-k+k'}$
12     q1 = k - kp
13     vq1 = calc_vq(q1,ξ,V0)
14     q2 = k + kp
15     vq2 = calc_vq(q2,ξ,V0)
16     Vkkp = 2*V0*(cos(q1*x0)*vq1 - cos(q2*x0)*vq2)
17     return Vkkp
18 end
```

ソースコード 2.13　ポテンシャルの作成

```
function make_Hk(N,a,V0)
    mat_Hk = zeros(Float64,N,N)
    dx = N/6
    ξ  = dx*a
    center = (N+1)/2
    x0 = center*a
    L = (N+1)*a
    for n in 1:N
        k = n*π/L
        for np in 1:N
            v = 0.0
            if n == np
                v = k^2
            end
            kp = np*π/L
            Vkkp = calc_Vkkp(k,kp,ξ,x0,V0)
            v += Vkkp*(1/2L)
            mat_Hk[n,np]= v
        end
    end
    return mat_Hk
end
```

ソースコード 2.14　波数表示でのハミルトニアン

ポテンシャルがない場合

ポテンシャルがない場合の最低固有値の計算は、差分化の時に作成したソースコード 2.9 の 4 行目の右辺を今用意したソースコード 2.14 に置き換えればよい。解いてみると、厳密に解いた解とほぼ一致した最小固有値が出ることがわかる。

2.6.4　ポテンシャルがある場合

ガウス型ポテンシャルがある場合は、差分化で解いた結果と比較すればよい。波動関数を比べてみよう。波数表示で数値的に解いているシュレーディンガー方程式は式 (2.94) である。よって、固有ベクトルとして得られるのは sin 関数の係数である b_k である。実空間でプロットするためには、得られた係数と sin 関数をかけて足し合わせればよい。そのコードをソースコード 2.15 に示す。

```
V0 = 1.0
N = 1000
a = 0.01
mat_H = make_Hk(N,a,V0)
ep,psi = eigen(mat_H)
println(ep[1])
a = 0.01
rp = zeros(Float64,N)
L = a*N
for i in 1:N
    xi = a*i
    for ik in  1:N
        k = π*ik/L
        rp[i] += psi[ik,1]*sin(k*xi)
    end
end
C = sum(dot(rp[1:N],rp[1:N])) # 規格化
rp = rp/sqrt(C)
plot(rp)
```

ソースコード 2.15　実空間でのプロット

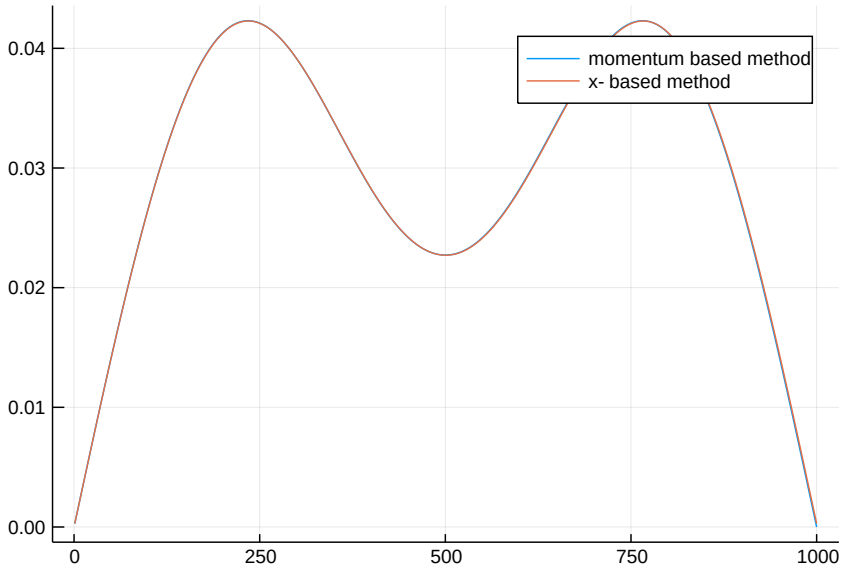

図 2.13　波数表示で解いた最低エネルギー固有値の波動関数と、同じパラメータでの差分化した方法で解いた波動関数の比較。

差分化した結果と重ねたプロットしたものが図 2.13 である。二つの方法は完全に同じ解を出している。また、ソースコード 2.12 と同様に、ポテンシャルの強度依存性を調べることもできる。

第 3 章

時間に依存しない 1 次元シュレーディンガー方程式:無限系散乱問題

3.1　散乱問題

量子力学の演習問題として定番のものとして、無限遠方から入射波がやってきて、何かポテンシャルに衝突し、反射と透過が起きる問題がある。一番簡単な問題としては、1 次元系において $x = 0$ にデルタ関数型ポテンシャル $V(x) = V_0\delta(x)$ が存在するシュレーディンガー方程式：

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V_0\delta(x)\right)\psi(x) = \epsilon\psi(x) \tag{3.1}$$

において、左側 $(x < 0)$ から平面波 $\exp(ikx)$ が入射した時に、$x = 0$ においての反射の様子と、右側 $(x > 0)$ における透過の様子を調べる問題がある。まず標準的なやり方を用いてこの問題を解き、その後より一般的な問題へと拡張してみよう。

なお、以後は、これまでと同様にシュレーディンガー方程式を

$$\left(-\frac{d^2}{dx^2} + V_0\delta(x)\right)\psi(x) = \epsilon\psi(x) \tag{3.2}$$

として無次元化しておく。

3.2　標準的なやり方

3.2.1　デルタ関数型ポテンシャルの場合

まずはじめに、標準的なやり方でデルタ関数型ポテンシャルの存在する散乱問題を解いてみよう。$x > 0$ と $x < 0$ の領域ではポテンシャルが存在しないため、あるエネルギー固

有値 $\epsilon = k^2$ を持つ平面波の線型結合で解を表すことができる。よって、$k > 0$ とすると波数 $k = \pm\sqrt{\epsilon}$ の二つの平面波で構成されている。そこで、それぞれの領域の解を

$$\psi_L(x) = A_L \exp(ikx) + B_L \exp(-ikx) \tag{3.3}$$
$$\psi_R(x) = A_R \exp(ikx) + B_R \exp(-ikx) \tag{3.4}$$

とおく。そして、今考えている問題は「左側 $(x < 0)$ から平面波 $\exp(ikx)$ が入射した時」であるので、右側には右向きの平面波しか存在せず、$B_R = 0$ である。そして、平面波の大きさを $|A_L|^2 = 1$ とすることができる。これを求めるためには、$x = 0$ でどのように解が接続されるかを調べる必要がある。しかし、$x = 0$ でのシュレーディンガー方程式はデルタ関数があるので無限大が出てきて扱うことができない。そこで、$x = 0$ から微小にずらした領域 $-\Delta x < x < \Delta$ を考えることとする。シュレーディンガー方程式をこの区間で積分してみると、

$$\int_{-\Delta x}^{\Delta x} dx \left(-\frac{d^2}{dx^2} + V_0\delta(x) \right) \psi(x) = \epsilon \int_{-\Delta x}^{\Delta x} dx\psi(x) \tag{3.5}$$
$$\left[\frac{d\psi(x)}{dx} \right]_{-\Delta x}^{\Delta x} + V_0\psi(0) = \epsilon \int_{-\Delta x}^{\Delta x} dx\psi(x) \tag{3.6}$$

となる。ここで、$\Delta x \to 0$ とすると、右辺がゼロになり、

$$\frac{d\psi_R(x)}{dx}\Big|_{x=0} - \frac{d\psi_L(x)}{dx}\Big|_{x=0} = V_0\psi(0) \tag{3.7}$$

という式が得られる。この式が意味するのは、「$x = 0$ において、一階微分が不連続になっている」ということである。波動関数自体の値は連続なので、$x = 0$ において

$$\psi_L(x = 0) = \psi_R(x = 0) \tag{3.8}$$

という条件式も存在している。よって、

$$A_L + B_L = A_R \tag{3.9}$$
$$ikA_L - ikB_L = ikA_R + V_0A_R \tag{3.10}$$

という式が得られる。これを整理すると、

$$ikA_L + ikB_L = ikA_R \tag{3.11}$$
$$ikA_L - ikB_L = (ik + V_0)A_R \tag{3.12}$$

より、

$$2ikA_L = (2ik + V_0)A_R \tag{3.13}$$
$$V_0A_L + (-2k^2 + V_0)B_L = 0 \tag{3.14}$$

となり、

$$\frac{A_R}{A_L} = \frac{2ik}{2ik + V_0} \tag{3.15}$$

$$\frac{B_L}{A_L} = \frac{-V_0}{2ik + V_0} \tag{3.16}$$

が得られる。この二つの絶対値の二乗は

$$|\frac{A_R}{A_L}|^2 = \frac{4k^2}{4k^2 + V_0^2} \tag{3.17}$$

$$|\frac{B_L}{A_L}|^2 = \frac{V_0^2}{4k^2 + V_0^2} \tag{3.18}$$

となる。これを透過率、および反射率、と呼ぶことができる。なお、透過率と反射率を足せば 1 になることは、この二つを足せば 1 になることから容易にわかる。量子力学の興味深い点として、どんなに小さなエネルギー（$k = \sqrt{\epsilon}$ なのでどんなに小さな波数）でも、壁の透過率が 0 ではない、という点がある。デルタ関数型ポテンシャルの強度 V_0 が無限大でない限り、どんなに大きなポテンシャルでも電子は透過することができる。これを「トンネル効果」と呼ぶ。

3.2.2 転送行列による方法

量子力学でよく扱われる散乱問題としては、デルタ関数型ポテンシャルの他には、矩形ポテンシャルによる散乱問題がある。これらのポテンシャルはある点でポテンシャルが不連続に変化するため、ポテンシャルがない領域とある領域にわけることができる。そこで、$x < -L/2$、$-L/2 < x < L/2$、$L/2 < x$ の三つの領域に分け、$x < -L/2$ と $L/2 < x$ ではポテンシャルがないとすると、どんな問題を考えていても、この二つの領域での波動関数は常に

$$\psi_L(x) = A_L e^{ikx} + B_L e^{-ikx} \tag{3.19}$$

$$\psi_R(x) = A_R e^{ikx} + B_R e^{-ikx} \tag{3.20}$$

と書ける。真ん中の領域での波動関数を ψ_M とすると、この領域の波動関数は解かなければわからない。そこで、

$$\begin{pmatrix} A_R \\ B_R \end{pmatrix} = \hat{T} \begin{pmatrix} A_L \\ B_L \end{pmatrix} \tag{3.21}$$

と書けば、行列 $\hat{T}$ を求めることさえできれば、問題を解いたことになる。もちろん、この行列 $\hat{T}$ を一般的に求めることは難しい。この形で書くことによって、ポテンシャルが不連続に変化する場合以外でも、ある領域 $-L/2 < x < L/2$ に局在する場合においても同様に扱うことができる。この $\hat{T}$ を転送行列と呼ぶ。

3.3　差分化を用いて散乱問題を解く

3.3.1　差分化されたシュレーディンガー方程式と散乱問題

上記の散乱問題を微分を差分化して解いてみよう。差分化された 1 次元シュレーディンガー方程式はこれまでも出てきたように

$$-\frac{1}{a^2}\left(\psi(x_{i+1})+\psi(x_{i-1})\right)+\left(\frac{2}{a^2}+V(x_i)\right)\psi(x_i)=\epsilon\psi(x_i) \tag{3.22}$$

である。前章で出てきた長さ L の壁に挟まれた系の場合には離散点の数は有限個であり、ハミルトニアンを行列として表すことができた。散乱問題の場合「無限遠方から電子がやってくる」という条件を満たすためには、何らかの方法で無限遠方を数値的に取り扱わなければならない。これは可能だろうか？

散乱問題を取り扱う方法を探るには、境界でのシュレーディンガー方程式を眺めればよい。いま、$x<-L/2$、$-L/2<x<L/2$、$L/2<x$ の三つの領域にわけるとする。そして、$x<-L/2$ と $L/2<x$ ではポテンシャルがないとする。$x=-L/2$ となる点を x_1、$x=L/2$ となる点を x_N とする。この時、x_1 におけるシュレーディンガー方程式は

$$-\frac{1}{a^2}\left(\psi(x_2)+\psi(x_0)\right)+\left(\frac{2}{a^2}\right)\psi(x_1)=\epsilon\psi(x_1) \tag{3.23}$$

となる。x_2 や他の点でのシュレーディンガー方程式と合わせて書くと、

$$-\frac{1}{a^2}\psi(x_2)+\left(\frac{2}{a^2}-\epsilon\right)\psi(x_1)=\frac{1}{a^2}\psi(x_0) \tag{3.24}$$

$$-\frac{1}{a^2}\left(\psi(x_3)+\psi(x_1)\right)+\left(\frac{2}{a^2}-\epsilon+V(x_2)\right)\psi(x_2)=0 \tag{3.25}$$

$$-\frac{1}{a^2}\left(\psi(x_4)+\psi(x_2)\right)+\left(\frac{2}{a^2}-\epsilon+V(x_3)\right)\psi(x_3)=0 \tag{3.26}$$

$$\vdots \tag{3.27}$$

となり、x_N においては

$$-\frac{1}{a^2}\left(\psi(x_{N+1})+\psi(x_{N-1})\right)+\left(\frac{2}{a^2}-\epsilon\right)\psi(x_N)=0 \tag{3.28}$$

となる。これらの式は互いに関連しているが、$\psi(x_0)$ の値と $\psi(x_{N+1})$ の値がどのようになっているかがわかれば、他の全ての領域での値がわかるようになっている。ここで、ϵ がわからないように思えるかもしれない。しかし、散乱問題においては「左側 $(x<0)$ から平面波 $\exp(ikx)$ が入射した時」のように波数 k が指定されており、これはつまりエネ

ルギーが指定されていることを意味している。つまり、左端と右端の状況さえ適切に設定すれば、この問題は解けることになる。これはつまり、境界条件を設定して微分方程式を解いていることになる。

3.3.2 解の接続

では、$\psi(x_0)$ と $\psi(x_{N+1})$ の値について考察してみよう。まず、$x > L/2$ ではポテンシャルがないので、その領域では解は

$$\psi_R(x) = A_R e^{ikx} + B_R e^{-ikx} \tag{3.29}$$

と書ける。そして、「左側 ($x < 0$) から平面波 $\exp(ikx)$ が入射した時」を考えているので、右側には左方向の平面波は存在しない。そのため、$B_R = 0$ である。$x = x_{N+1}$ は $x > L/2$ の領域なので、

$$\psi(x_{N+1}) = A_R e^{ikx_{N+1}} \tag{3.30}$$

と書ける。一方、$x = x_N$ においてもポテンシャルがないとすれば、

$$\psi(x_N) = A_R e^{ikx_N} \tag{3.31}$$

と書ける。よって、

$$\psi(x_{N+1}) = e^{ika}\psi(x_N) \tag{3.32}$$

という関係式が存在している。これで、$\psi(x_{N+1})$ を $\psi(x_N)$ を用いて書くことができた。

次に、$\psi(x_1)$ である。$x = x_1$ において波動関数が

$$\psi(x_1) = A_L e^{ikx_1} + B_L e^{-ikx_1} \tag{3.33}$$

とする。ここで、この点を基準として考えるとして、$A_L = A'_L e^{-ikx_1}$、$B_L = B'_L e^{ikx_1}$ とおいて $B'_L \to B_L$ とおきなおすと

$$\psi(x_1) = A_L + B_L \tag{3.34}$$

となる。$x = x_0$ では、右向きと左向きの平面波が存在しているが、これは A_L と B_L を用いて

$$\psi(x_0) = A_L e^{-ika} + B_L e^{ika} \tag{3.35}$$

と書ける。よって、

$$\psi(x_0) = A_L e^{-ika} + (\psi(x_1) - A_L)e^{ika} = \psi(x_1)e^{ika} + A_L(e^{-ika} - e^{ika}) \tag{3.36}$$

となる。これにより、シュレーディンガー方程式は

$$-\frac{1}{a^2}\psi(x_2)+\left(\frac{2}{a^2}-\epsilon\right)\psi(x_1)-\frac{1}{a^2}e^{ika}\psi(x_1)=\frac{1}{a^2}(e^{-ika}-e^{ika})A_L \tag{3.37}$$

$$-\frac{1}{a^2}\left(\psi(x_3)+\psi(x_1)\right)+\left(\frac{2}{a^2}-\epsilon+V(x_2)\right)\psi(x_2)=0 \tag{3.38}$$

$$-\frac{1}{a^2}\left(\psi(x_4)+\psi(x_2)\right)+\left(\frac{2}{a^2}-\epsilon+V(x_3)\right)\psi(x_3)=0 \tag{3.39}$$

$$\vdots \tag{3.40}$$

$$-\frac{1}{a^2}\psi(x_{N-1})+\left(\frac{2}{a^2}-\epsilon\right)\psi(x_N)-\frac{1}{a^2}e^{ika}\psi(x_N)=0 \tag{3.41}$$

となる。これを行列表示すると、

$$\begin{pmatrix} -\left(e^{ika}-2\right)-\epsilon a^2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2+(V(x_2)-\epsilon)a^2 & -1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & \cdots & 0 & -1 & -\left(e^{ika}-2\right)-\epsilon a^2 & \end{pmatrix}\begin{pmatrix} \psi(x_1) \\ \psi(x_2) \\ \psi(x_3) \\ \psi(x_4) \\ \vdots \\ \psi(x_N) \end{pmatrix}$$

$$=\begin{pmatrix} \left(e^{-ika}-e^{ika}\right)A_L \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{3.42}$$

という形に書ける。これは、$Ax=b$ の形をしているので、連立方程式である。入射波を A_L で与えれば、それよりも右側の波動関数の様子は連立方程式を解くことで求めることができる。シュレーディンガー方程式をこの形で書くと、$-L/2<x<L/2$ にどんなポテンシャルが入っていたとしても、数値的に解を求めることができる。

3.3.3　数値計算結果

上記の方法を用いて、矩形ポテンシャルの場合を考えてみよう。矩形ポテンシャルとして、$-R<x<R$ でのみ $V(x)=V_0$ であるものを考える。式 (3.42) の左辺の”ハミルトニアン”行列を作成するコードはソースコード 3.1 となる。なお、このハミルトニアンは対角要素に複素数が入っているためエルミートではなく、いわゆる普通の量子力学で出てくるエルミート行列とは異なっている。このコードでは spzeros を使っている。これは Julia で疎行列を定義するためのものである。疎行列とは、行列の要素のほとんどがゼ

ロである行列のことで、わざわざゼロの部分をメモリとして持っておく必要がないために、疎行列専用のライブラリを使うと固有値問題や連立方程式を高速に解くことができる。Fortran や C でも使うことができるが、疎行列を作成する方法と、疎行列用のライブラリの呼び出す方法を理解しておかないと、使うのが難しい。Julia では、SparseArrays を使うことで、普通の密行列と同程度の操作で疎行列を扱うことができる。

```
using LinearAlgebra # 線形代数ライブラリを呼び出す
using SparseArrays # 疎行列ライブラリを呼び出す
using Plots # プロットライブラリを呼び出す

function make_mat(N,k,R,L,V0)
    mat_H = spzeros(ComplexF64,N,N) #疎行列を用意
    a = L/(N-1) # 刻み幅
    ε = k^2 # エネルギー
    for i=1:N
        x = (i-1)*a -L/2
        if abs(x) < R # |x| < Rに矩形ポテンシャル
            V = V0
        else
            V = 0
        end
        j = i
        if i == 1
            mat_H[i,j] =-(exp(im*k*a)-2)/a^2- ε
        elseif i==N
            mat_H[i,j] = -(exp(im*k*a)-2)/a^2 - ε
        else
            mat_H[i,j] = 2/a^2+(V- ε )
        end
        j = i + 1
        if j < N+1
            mat_H[i,j] = -1/a^2
        end
        j = i - 1
        if j > 0
            mat_H[i,j] = -1/a^2
        end

    end
    return mat_H
end
```

ソースコード 3.1　散乱問題での”ハミルトニアン”

左辺の行列さえできれば、右辺を作成して連立方程式を解けばよいこととなる。Julia では、$Ax = b$ という連立方程式は `x = A \ b` とすれば解くことができる。これは密行列でも疎行列でも変わらない。エネルギーを変化させて透過率がどのように変化するかを調べることとする。透過率は $|\psi(x_N)|^2$ によって得ることができる。よって、コードはソースコード 3.2 となる。

```

function calc_T(N,ne,R,L,V0,emin,emax)
    a = L/(N-1)
    vec_e = range(emin, stop=emax, length=ne)
    T = zeros(Float64,ne) # 透過率
    for ie = 1:ne
        k = sqrt(vec_e[ie])
        mat_H = make_mat(N,k,R,L,V0)
        b = zeros(ComplexF64,N)
        b[1] = (exp(-im*k*a)-exp(im*k*a))/a^2
        x = mat_H \ b  # 連立方程式 H x = bを解く
        T[ie] = abs(x[N])^2
    end
    return T,vec_e./V0
```

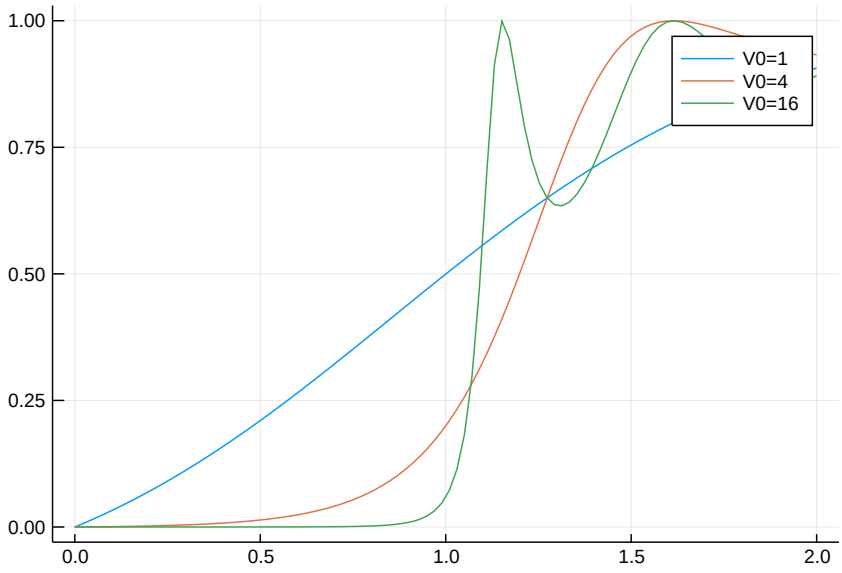

図 3.1　矩形ポテンシャルが存在する場合のエネルギー依存性。ポテンシャルの高さを変化させた。ときどき透過率が 1 となっており、「完全透過」が起きていることがわかる。

```
15 end
```

ソースコード 3.2　散乱問題を解き、透過率を計算するコード

これを使って、V_0 の大きさ依存性を調べるコードはソースコード**??**である。ここで、`plot!`というものを使っているが、これは `plot` のあとに書くことでグラフに追記ができるものである。`savefig("T.pdf")` は pdf 形式にして図を保存するコマンドである。

```
1  N = 1000 # 差分の点の数
2  L = 10 # システムサイズ
3  a = L/(N-1) # 刻み幅
4  V0 = 1
5  emin = 0.0 # 最小のエネルギー
6  emax = 2.0 # 最大のエネルギー
7  ne = 100 # 計算するエネルギーの点の数
8  R= 1.0 # 矩形ポテンシャルの幅
9  T1,ene1 = calc_T(N,ne,R,L,V0,emin,emax)
10 plot(ene1,T1,label="V0=1")
11 V0 = 4
12 emin = 0.0 # 最小のエネルギー
13 emax = 2.0*V0 # 最大のエネルギー
14 T2,ene2 = calc_T(N,ne,R,L,V0,emin,emax)
15 plot!(ene2,T2,label="V0=4") # plot!はplotにつづけて重ねる
16 V0 = 16
17 emin = 0.0 # 最小のエネルギー
18 emax = 2.0*V0 # 最大のエネルギー
19 T3,ene3 = calc_T(N,ne,R,L,V0,emin,emax)
20 plot!(ene3,T3,label="V0=16")
21 savefig("T.pdf")
```

ソースコード 3.3　V_0 依存性をプロットする

得られた結果を図 3.1 に示す。この図は透過率のエネルギー依存性であり、$E < V_0$ で

も有限の透過率があることから、「トンネル効果」によって粒子が矩形ポテンシャルを通り抜けていることがわかる。また、V_0 が大きい時に透過率が 1 となる「完全透過」が起きていることがわかる。ここで、完全透過は矩形ポテンシャル中の波数がポテンシャルの形にマッチングしている時に起きることを見てみよう。矩形ポテンシャル中の波数 κ は、エネルギーが V_0 シフトしているとみなせるので、$\kappa^2 = k^2 - V_0$ と書ける。もし、$k^2 < V_0$ であれば、矩形ポテンシャル中では κ は純虚数となり、減衰関数となる。$k^2 > V_0$ のとき、κ がうまくポテンシャルのサイズにフィットすると何が起こるだろうか？コードをソースコード 3.4 に示す。この図では、波動関数の様子をプロットしている。

```
# 完全透過のチェック
V0 = 16
κ = π # 矩形ポテンシャルの波数がマッチした時
k = sqrt(κ^2+V0)
b = zeros(ComplexF64,N)
b[1] = (exp(-im*k*a)-exp(im*k*a))/a^2
mat_H = make_mat(N,k,R,L,V0)
x = mat_H \ b  # 連立方程式 H x = bを解く
plot(real.(x),label="Kappa = pi: Real part")
plot!(imag.(x),label="Kappa = pi: Imaginary part")

V0 = 16
κ = π*0.9 # 矩形ポテンシャルの波数がマッチしていない時
k = sqrt(κ^2+V0)
b = zeros(ComplexF64,N)
b[1] = (exp(-im*k*a)-exp(im*k*a))/a^2
mat_H = make_mat(N,k,R,L,V0)
x = mat_H \ b  # 連立方程式 H x = bを解く
plot!(real.(x),label="Kappa = 0.9pi: Real part")
plot!(imag.(x),label="Kappa = 0.9pi: Imaginary part")

savefig("xs.pdf")
```

ソースコード 3.4　波動関数の様子

得られた図 3.2 に示す。κ がちょうど矩形ポテンシャルにマッチしていると、矩形ポテンシャルの両側で同じ振幅になることがわかる。振幅が同じということは、左から入射した波がそのまま右に抜けていることを意味していおり、これが完全透過である。

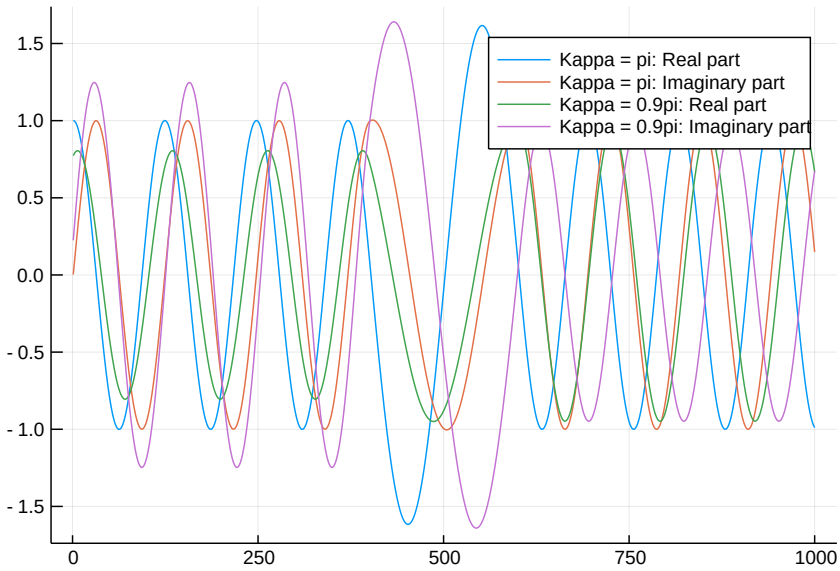

図 3.2　波動関数の様子。実部と虚部。

第 4 章

時間に依存しない 2 次元シュレーディンガー方程式

4.1　二次元シュレーディンガー方程式

二次元シュレーディンガー方程式を考えてみよう。二次元シュレーディンガー方程式は

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r})\right)\psi(\mathbf{r}) = \epsilon\psi(\mathbf{r}) \tag{4.1}$$

となる。ここで、ナブラ演算子 ∇^2 は、直交座標系では

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{4.2}$$

である。そして、円筒座標系：

$$x = r\cos\theta \tag{4.3}$$

$$y = r\sin\theta \tag{4.4}$$

では、

$$\nabla^2 = \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial}{\partial \theta^2} \tag{4.5}$$

である。ここでは、ポテンシャルが

$$V(\mathbf{r}) = V(r) \tag{4.6}$$

と r にのみ依存している形を仮定する。このとき、変数分離を行うことができて、解を

$$\psi(\mathbf{r}) = \xi(r)\Psi(\theta) \tag{4.7}$$

とすると、

$$-\frac{\hbar^2}{2m}\Psi(\theta)\left(\frac{\partial^2}{\partial r^2}+\frac{1}{r}\frac{\partial}{\partial r}\right)\xi(r)+V(r)\xi(r)\Psi(\theta)-\frac{\hbar^2}{2m}\xi(r)\frac{1}{r^2}\frac{\partial}{\partial\theta^2}\Psi(\theta)=\epsilon\xi(r)\Psi(\theta) \tag{4.8}$$

となり、両辺を $\xi(r)\Psi(\theta)$ で割って r^2 をかけると

$$-\frac{\hbar^2}{2m}\frac{r^2}{\xi(r)}\left(\frac{\partial^2}{\partial r^2}+\frac{1}{r}\frac{\partial}{\partial r}\right)\xi(r)+r^2V(r)-\frac{\hbar^2}{2m}\frac{1}{\Psi(\theta)}\frac{\partial}{\partial\theta^2}\Psi(\theta)=\epsilon r^2 \tag{4.9}$$

となる。ここで、左辺第三項は r に依存しておらず、その他の項は θ に依存していないため、方程式が解を持つためには、左辺第三項が定数でなければならない。つまり、

$$\frac{1}{\Psi(\theta)}\frac{\partial}{\partial\theta^2}\Psi(\theta)=A \tag{4.10}$$

とおける。この微分方程式を満たす解は

$$\Psi(\theta)=\exp(in\theta) \tag{4.11}$$

であり、

$$A=-n^2 \tag{4.12}$$

である。また、

$$\Psi(\theta)=\Psi(\theta+2\pi) \tag{4.13}$$

である必要があるため、$n=0,\pm1,\pm2,\cdots$ という整数でなければならない。以上から、$R(r)$ に関する微分方程式は

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial r^2}+\frac{1}{r}\frac{\partial}{\partial r}\right)\xi(r)+V(r)\xi(r)-\frac{\hbar^2}{2m}\frac{1}{r^2}(-n^2)\xi(r)=\epsilon\xi(r) \tag{4.14}$$

$$\left[-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial r^2}+\frac{1}{r}\frac{\partial}{\partial r}\right)+\frac{\hbar^2n^2}{2mr^2}+V(r)\right]\xi(r)=\epsilon\xi(r) \tag{4.15}$$

となる。これが、二次元円筒座標系におけるシュレーディンガー方程式である。

4.1.1　境界条件

半径 R のディスク状の領域を考える。このとき、境界条件は

$$\xi(r=R)=0 \tag{4.16}$$

である。

4.2 ベッセルの微分方程式

この方程式を解くために、すこし整理する。無次元化を行い、

$$r = \sqrt{\frac{2m}{\hbar^2}} r' \tag{4.17}$$

とする。また、以後は r' は r とおくことにする。このとき、方程式は

$$\left[-\left(\frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} \right) + \frac{n^2}{r^2} + V(r) \right] \xi(r) = \epsilon \xi(r) \tag{4.18}$$

$$\left[\frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \epsilon - \frac{n^2}{r^2} - V(r) \right] \xi(r) = 0 \tag{4.19}$$

$$\left[r^2\frac{\partial^2}{\partial r^2} + r\frac{\partial}{\partial r} + r^2\epsilon - n^2 - V(r)r^2 \right] \xi(r) = 0 \tag{4.20}$$

となる。さらに、$r = r'/\sqrt{\epsilon}$ とおくと（そして r と置き直すと）

$$\left[r^2\frac{\partial^2}{\partial r^2} + r\frac{\partial}{\partial r} + (r^2 - n^2) \right] \xi(r) - V(r)\frac{r^2}{\epsilon}\xi(r) = 0 \tag{4.21}$$

となる。さて、この形の方程式は、ベッセルの微分方程式

$$x^2\frac{\partial^2}{\partial x^2}y(x) + x\frac{\partial}{\partial x}y(x) + (x^2 - n^2)y(x) = 0 \tag{4.22}$$

ととてもよく似た形をしている。この方程式の解はベッセル関数と呼ばれ、n が整数のとき、方程式の解は第 1 種ベッセル関数 $J_n(x)$ と第 2 種ベッセル関数 $Y_n(x)$ の線形結合

$$y(x) = C_1 J_n(x) + C_2 Y_n(x) \tag{4.23}$$

と書かれることが知られている。**未完成**