公立はこだて未来大学 2025 年度 システム情報科学実習
グループ報告書

Future University Hakodate 2025 Systems Information Science Practice
Group Report

プロジェクト番号/Project No.

7

プロジェクト名

ダイナミクスインサイト：複雑パターンの解明

Project Name

Dynamics Insightss: Uncovering Complex Patterns

グループ名

物理班

Group Name

Physics group

プロジェクトリーダー/Project Leader

白木慶汰　Keita Shiraki

グループリーダー/Group Leader

大場英輝　Hideki Oba

グループメンバー/Group Member

大場英輝　Hideki Oba
友杉悠愛　Yuuna Tomosugi
山崎大紀　Daiki Yamazaki

指導教員

加藤譲　義永那津人　栗川知己　リアボフ・ウラジミール　リヴァーズ・ダミアン

Advisor

Yuzuru Kato　Natsuhiko Yoshinaga　Tomoki Kurikawa　Volodymyr Riabov
Damian Rivers

提出日

2026 年 1 月 21 日

Date of Submission

Jan. 21, 2026

概要

近年量子コンピュータは、古典流体シミュレーションにおける新しい方法として検討されている。量子ビットによって指数的に大きな状態空間を表現できること、および量子もつれの効果によって、特定の問題においては古典シミュレーションを超える効率が期待される。しかし、量子回路で非可逆性および非線形性のある計算をどのように扱うかについて、一般的な方法は確立されていない。本班では、量子ウォークの時間発展演算子に (1) 情報の破棄 (2) 密度依存の位相変化を導入することで、非可逆性および非線形性を表現する手法を提案する。本手法により、古典流体のシミュレーションを行う際も、古典シミュレーションよりも効率的なシミュレーションが可能になると期待される。さらに本手法の結果は、流体に限らず、非可逆性または非線形性に起因する多様な現象を量子コンピュータでシミュレーションできる可能性を示す。

（※文責: 友杉悠愛）

キーワード　Dewetting、カルマン渦、擬似スペクトル法、格子ボルツマン法、シミュレーション、流体力学、量子アルゴリズム、量子コンピュータ

Abstract

Recently, quantum computers have been explored as a new approach to fluid simulation. Quantum computers can represent an exponentially large state space, which is expected to achieve higher efficiency than classical simulations for certain problems. However, general methods for treating irreversibility and nonlinearity within quantum circuit models have not been established yet. Here we develop a framework to incorporate irreversibility and nonlinearity into quantum circuits by (i) discarding information and (ii) introducing density-dependent phase shifts. This framework is expected to enable the simulation of the governing equations more efficiently than classical simulations. Furthermore, our results suggest that quantum computers have the potential to simulate a wide range of complex phenomena arising from irreversibility or nonlinearity.

Keyword Dewetting, Kármán vortex, Pseudo-spectral method, Lattice Boltzmann Method, Simulation, Fluid dynamics, Quantum algorithm, Quantum computer

目次

第 1 章　イントロダクション

近年量子コンピュータは、古典流体シミュレーションにおける新しい方法として検討されている。量子コンピュータは、n 個の量子ビットに対して 2^n の自由度を持つ。量子ビットによって指数的に大きな状態空間を表現できること、および量子もつれの効果によって、特定の問題においては古典シミュレーションを超える効率が期待される。

しかし、流体の粘性による非可逆性、および移流による非線形性を量子回路でどのように扱うかについて、一般的な方法は確立されていないことが課題である。量子コンピュータにおいては、時間発展演算子にユニタリ性が要請されることにより、可逆性が課される。また、時間発展は線形演算によって行われる。ゆえに、量子コンピュータでは、流体の非可逆性と非線形性を表現することが難しい。

本班では、量子ウォークの時間発展演算子に (1) 情報の破棄 (2) 密度依存の位相変化を導入することで、非可逆性および非線形性を表現する手法を提案する。(1) については、開放した量子系の理論を導入することによって解決する。熱浴の状態に対応するビットを系の粒子に対応するビットと相互作用させ、その後熱浴の情報を破棄することで、非可逆性を導入する。(2) については、光 Kerr 効果に基づき密度に応じて、流れに対応する位相勾配を発生させて非線形性を導入する。

本手法は、流体の粘性と移流についての課題を解決することにつながる。古典流体のシミュレーションを行う際も、古典シミュレーションよりも効率的なシミュレーションが可能になると期待される。

さらに本手法の結果は、流体に限らず、非可逆性または非線形性に起因する多様な現象をシミュレーションできる可能性が量子コンピュータにはあることを示している。

本報告書では、2 章から 3 章で古典コンピュータでの流体シミュレーションについて紹介する。具体的な流体現象の例として、2 章では dewetting、3 章ではカルマン渦を取り上げる。古典コンピュータでの結果を示した後、4 章にて量子コンピュータによる結果を示す。4 章の初めに、dewetting を題材に、量子リザバーコンピューティングを用いて、量子回路に流体シミュレーションを行う能力があることを確認する。その後、カルマン渦を題材に、量子ウォークへの非可逆性、非線形性の導入によって古典流体のシミュレーションを行うことができる可能性があることを示す。

（※文責: 友杉悠愛）

第 2 章　薄膜のシミュレーション

2.1　背景

人間、生きていれば液体をこぼしてしまうこともあるだろう。たいていの人は、こぼしてもすぐに拭き取ってしまうが、ここでは拭かずに様子をじっくり見てみよう。すると、液体が基盤から大きな角度を作ることもあれば、小さな角度を作ることもあるということがわかる。このような現象を、ぬれ (wetting) と呼ぶ。液体が基盤からなす角度が小さいほど、液体は基盤によくぬれていると理解できる。

ぬれは、産業や工学への応用という観点からも重要であるが、純粋にそれ自体が面白い現象である。そこで、ぬれの現象の中でも dewetting という現象を取り上げることにした。この現象は、基盤と液体の相互作用により、液膜が「不安定」になっていることによって起こる。相分離の文脈では、「不安定」から「安定」への動的移行を「スピノーダル分解」と呼ぶことに由来して、スピノーダル dewetting ということもある。すなわち、wetting も dewetting も、広い意味で考えれば、固相-気相-液相の三相間における相分離のようなものなのである。

静的な系であれば、液滴の形状は、単純に自由エネルギーの最小化問題を考えればよいが[*1]、ここで気になるのは、時間発展によって形状がどのように変わるかということである。自由エネルギー的な考察だけでは、時間発展はよくわからない。しかし、dewetting も含む液膜の広がりについてのダイナミクスは、流体のダイナミクスであるから、流体力学の支配方程式を用いて考えることができる。dewetting の流体力学的支配方程式をシミュレーションし、それが実際の様子と比較してもっともらしい振る舞いを見せているかを確認した。

2.2　手法

[4] によると、dewetting の支配方程式は、Navier-Stokes 方程式と連続の式から導出され、次の通りである[*2]。

$$\frac{\partial h}{\partial t} = \nabla \cdot [\frac{h^2}{3}(3\beta + h)\nabla(p + \phi)] \tag{2.1}$$

ただし、

$$p = -\nabla^2 h \tag{2.2}$$

$$\phi = \frac{1}{h^3} - S\exp\left[\delta(1 - \frac{h}{h_{\min}})\right] \tag{2.3}$$

[*1] 自由エネルギーは平衡状態において最小値をとる。従来の熱力学では自由エネルギーは極小値と最小値が一致する。ということは、従来の熱力学では、不安定点や準安定点は記述できないということになる。しかし、これを拡張して、自由エネルギーが擬似的に、秩序パラメータについて準安定点や不安定点をもつとして、それが熱揺らぎなどの摂動で最小値 (安定点) に落ちるという考え方がある。ただし、その擬似的な自由エネルギーの準安定点や不安定点には物理的な意味があると保証されているわけではない。しかし、相分離 (を含む相転移一般) の振る舞いを、もっともらしく記述できるし、感覚的でもあると思う。

[*2] 導出については前期の報告書を参照されたい。ただし、界面張力の寄与が主体となる流れを考えるため、重力の寄与などの項は落としている。

なお、$\beta, S, h_{\min}$ は定数である。(2.1) について、$\frac{h^2}{3}(3\beta + h)$ の項は流体の流れを表しており、β は基盤の滑りやすさを表現するために含まれている。圧力 p は、液体-気体界面張力 (表面張力) によって、ゆらぎを安定化させるように働く[*3]。ϕ は液体-固体間についての自由エネルギーの h に関する一階微分である。ϕ の積分した形は、定数項や細かいパラメータを除けば $-\frac{1}{2h^2} + e^{-h}$ の形になるから、自由エネルギーを小さくするという観点から、$-\frac{1}{2h^2}$ は膜を薄くする効果、e^{-h} は膜を厚くする効果があることがわかる[*4]。この項の寄与によって、安定であるはずの平坦な形状よりも[*5]、薄くなった方がエネルギー的に有利になる点が現れる。そのような点で $h \to 0$ となり、膜の破裂が起こる。それによりできた穴の部分が広がり、山の部分が互いに近づいていく。そして最終的に、凹凸のある形状に収束する。

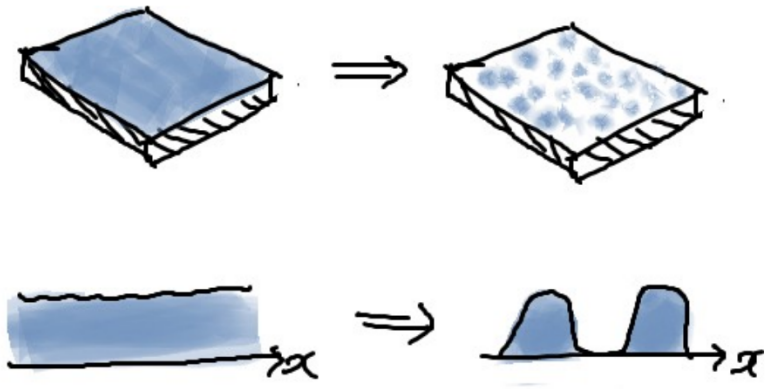


図 2.1　dewetitng

ところで先ほど、自由エネルギーを小さくするために、$\frac{1}{h^3}$ の寄与によって、$h \to 0$ となると述べた。これは物理的に正当な振る舞いである。しかし、コンピュータで計算する際には、$\frac{1}{h^3}$ が非常に大きな値となり、数値的な限界で発散が起きてしまう。そのため、まともに (2.1) を解こうとすると、膜の破裂が起こる際に数値的な発散が起きて計算できない。そこで、自由エネルギーが発散しない形で書いてみることにする。たとえば、液体-基盤間の自由エネルギーを f として、$f = -\frac{1}{2h^2} + e^{-h}$ の代わりに、次のような形が考えられる。

$$f = -\frac{1}{2}\frac{1}{h^2 + \epsilon} + e^{-h} \tag{2.4}$$

ただし、ϵ は微小量である。

第一項は、$h = 0$ で有限の最小値をもち、数値的に発散しない。これを h で一階微分したものが ϕ であり、

$$\phi = \frac{h}{(h^2 + \epsilon)^2} - e^{-h} \tag{2.5}$$

となる。

このように、項の役割を変えずに自由エネルギーの形を少し書き換えることにより、正当な正則化を行うことができる。したがって、(2.3) の代わりに (2.5) を用いる。

[*3] Laplace 圧に由来する。

[*4] $\frac{1}{h^3}$ は hamaker 定数が負の場合の van der Waals 相互作用エネルギーに由来し、e^{-h} は排除体積効果としての分子間の斥力に由来する。

[*5] ゆらぎのある平面上の液膜は、表面の自由エネルギーを最小化するべく、平坦になりたがる。

数値計算の手法としては、空間微分には擬似スペクトル法を用いた。擬似スペクトル法とは、微分を実空間ではなく、波数空間で行う手法のことである。たとえば、Fourier 変換を用いれば、$h(x)$ の n 階微分は、

$$\frac{\partial^n h}{\partial x^n} = \frac{\partial^n}{\partial x^n}\int dk\ \hat{h}e^{ikx} = \int dk[(ik)^n\hat{h}]e^{ikx} =: \mathcal{F}^{-1}[(ik)^n\mathcal{F}[h]]$$

と書ける。このように、微分の演算を行うことなく、単純に ik を掛けるだけで計算できる。ただし、コンピュータで計算する際は積分は和となる[*6]。さらに、計算コストを減らすために高速 Fourier 変換を用いる。python のライブラリに、numpy.fft.fft($\mathcal{F}$ に対応) や、numpy.fft.ifft($\mathcal{F}^{-1}$ に対応) があるため、実装は非常に簡単である。つまり、n 階微分は ifft((1.0j * k)**n * fft(h)) のように書けばよい[*7]。

非線形項の Fourier 変換はどうなるのか、めまいのするような畳み込みを計算しなければならないのか、という心配はせず上記の h の部分に非線形部分を当てはめてそのまま計算した。そうすると aliasing 誤差が発生するという、何とも細々とした話があるが、高周波数成分を落とすフィルターを掛けたり、波数領域の周りを 0 で padding したりすると誤差は少なくなる。とはいうものの、今回はそこまで高周波数成分は現れない現象であろうため、その辺りは特に考慮していない[*8]。

なお、時間発展は、python のライブラリである scipy.integrate.solve_ivp を用いた。引数の指定一つで、時間の評価方法を指定できたり、時間刻みを adaptive にできたりと、非常に便利なライブラリである。

2.3 結果

python でシミュレーションを行った結果、$h = h(x,t)$ とした場合は次の結果となった。なお、横軸が x、縦軸が h である。動画として出力したもののスナップショットを撮ったものを、動画の代わりに載せる。

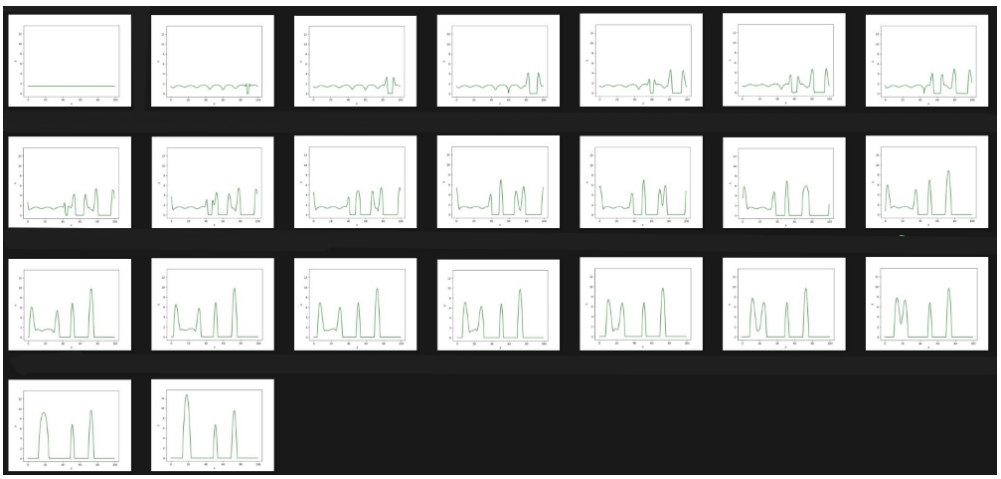


図 2.2　1 次元のシミュレーション結果

以下は、液滴の高さ $h = h(x,y,t)$ を密度プロットしたものの時間変化について、スナップショットを撮影したものである。

[*6] すなわち、$\Delta k \to 0$ の極限を取る前の Fourier 級数展開に対応する。したがって、周期境界条件が課されることになる。

[*7] python では、虚数単位は i ではなく j である。

[*8] 乱流などの場合は、そのあたりがシビアになってくるのかもしれない。

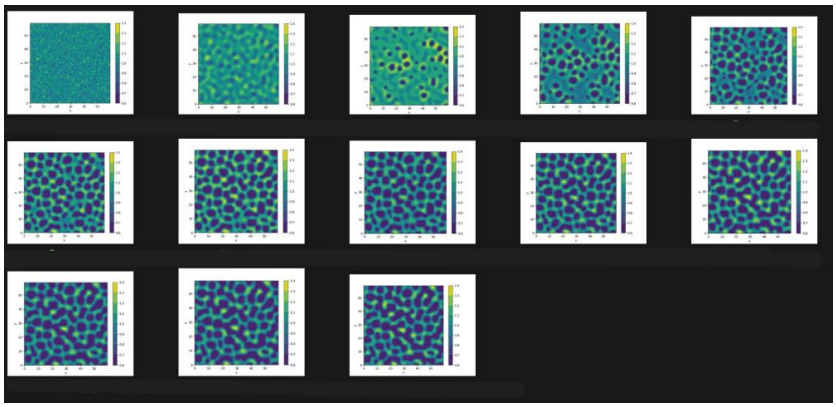


図 2.3　2 次元のシミュレーション結果

なお、1 変数の場合のコードは次の通りである。

```
import numpy as np
from numpy.fft import rfft, irfft, fftfreq
from scipy.integrate import solve_ivp

# 微分
def diff(h,k):
  return irfft((1.0j * k) * rfft(h))
def diff2(h,k):
  return irfft((1.0j * k)**2 * rfft(h))
def diff3(h,k):
  return irfft((1.0j * k)**3 * rfft(h))

# 支配方程式
def eq(t, h, k, beta):

    epsilon = 1.0e-2
    phi = h/(h**2 + epsilon)**2 - np.exp(-h)

    term1 = h**2/3 * (3*beta + h)
    term2 = -diff3(h, k) + diff(phi, k)
    flux = term1 * term2

    return diff(flux, k)


L = 100.0 # x軸長さ
T = 2000.0 # 時間
N = 250 # 格子数
dx = L/N
print("dx=", dx)
```

```
x = np.linspace(0, L, N, endpoint=False)
k = 2 * np.pi * np.fft.rfftfreq(N, d=dx)

# パラメータ
beta = 0.5

# 初期値
np.random.seed(0)
h0 = 1.5*np.ones(len(x)) - 0.01*np.random.rand(len(x))


sol = solve_ivp(eq, (0, T), h0,'Radau',dense_output=True, args=(k,
    beta))
print(sol.message)
print(" Number of time steps :", sol.t.size)
print(" Minimam time step    :", min(np.diff(sol.t)))
print(" Maximum time step    :", max(np.diff(sol.t)))

# 計算結果
h = sol.y

# 可視化
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
fig, ax = plt.subplots()
ims = []

for hh in h.T[::20]:
    im = plt.plot(x, hh, 'g')
    plt.xlabel("$x$")
    plt.ylabel("$h$")
    ims.append(im)
ani = animation.ArtistAnimation(fig, ims, interval=100)
HTML(ani.to_html5_video())
```

2 変数の場合は、次の通りである。

```
import numpy as np
from numpy.fft import fft2, ifft2, fftfreq
from scipy.integrate import solve_ivp


```

```
# laplacian
def lap(h, kx, ky):
    return ifft2((1.0j * kx)**2 * fft2(h)).real + ifft2((1.0j * ky)**2 * fft2(h)).real

# gradient
def grad(h, kx, ky):
    return np.array([ifft2((1.0j * kx) * fft2(h)).real, ifft2((1.0j * ky) * fft2(h)).real])

# divergence
def div(a, kx, ky):
    return ifft2((1.0j * kx) * fft2(a[0])).real + ifft2((1.0j * ky) * fft2(a[1])).real

def eq(t, h_flat, kx, ky, beta, N):

    h = h_flat.reshape(N, N)
    term1 = h**2/3 * (3*beta + h)
    p = -lap(h, kx, ky)
    phi = h/(h**2 + 1.0e-1)**2 - np.exp(-h)

    term2 = grad(p + phi, kx, ky)

    flux = term1 * term2

    return div(flux, kx, ky).flatten()


L = 60.0
T = 10.0
N = 80
dx = L / N
print("dx=", dx)
x = np.linspace(0, L, N, endpoint=False)
y = np.linspace(0, L, N, endpoint=False)

k=fftfreq(N)*2*np.pi*N/L
kx=np.array([[k[j] for j in range(N)] for i in range(N)])
ky=kx.T

beta = 0.5
```

```
45
46 np.random.seed(0)
47 h0 = np.random.normal(1.0, 0.1, (N, N))
48 h0 = h0.flatten()
49
50 sol = solve_ivp(eq, (0, T), h0, method='RK45', dense_output=True,
       args=(kx, ky, beta, N), rtol=1.0e-7,atol=1.0e-6)
51
52 print(sol.message)
53 print(" Number of time steps :", sol.t.size)
54 print(" Minimam time step    :", min(np.diff(sol.t)))
55 print(" Maximum time step    :", max(np.diff(sol.t)))
56
57 #####
58
59 import matplotlib.pyplot as plt
60 import matplotlib.animation as animation
61 from IPython.display import HTML
62
63 fig, ax = plt.subplots()
64 ims = []
65 flag = True
66
67 ax.set(xlabel='x', ylabel='y', aspect='equal')
68
69 for h in sol.y.T[::50]:
70   h = h.reshape(N,N)
71   im = ax.contourf(x, y, h)
72
73   # カラーバー大量発生を防止
74   if flag == True:
75     fig.colorbar(im, ax=ax)
76     flag = False
77
78   ims.append([im]) # リストのリストにしなければならないらしい。
79
80 ani = animation.ArtistAnimation(fig, ims, interval=100)
81 HTML(ani.to_html5_video())
```

なお、2 変数に対する高さをプロットするコードも載せておくことにする。

```
1 fig = plt.figure()
2 ax = fig.add_subplot(111, projection='3d')
```

```
ims = []
x, y = np.meshgrid(x, y)

ax.view_init(elev=70)
ax.set(xlabel='x', ylabel='y', zlabel='h')

for h in sol.y.T[::50]:
  h = h.reshape(N,N)
  im = ax.plot_surface(x, y, h,color='g', edgecolor='royalblue', lw
      =0.5, rstride=8, cstride=8,alpha=0.5)
  ims.append([im])

ani = animation.ArtistAnimation(fig, ims, interval=100)
HTML(ani.to_html5_video())
```

2.4 考察

結果を見ると、どちらも膜の破裂と穴の形成が確認できる。したがって、シミュレーションはおおよそうまくいったことがわかる。自由エネルギーを発散のしない形に修正したことにより、数値的な発散も起きない。支配方程式 (2.1) を見れば、これは体積 (総量) を保存する形であるが、結果を見ても、体積を保存するように形を変えていることがわかる。

このシミュレーションは、「数理モデリング」という講義と関係がある。この講義には、偏微分方程式の数値計算手法についてのパートがある。

これでめでたくシミュレーションを行うことができたが、残る疑問は、この系における自由エネルギーの形である。しかし、流体力学から出発したため、系の自由エネルギーを最小化するということを考えなくても、支配方程式が導出できてしまう。したがって、方程式を眺めているだけでは、系の自由エネルギーを求めることができない。この点について、補遺の最後に触れる。今後の課題としては、流体力学的に得られた支配方程式と統計力学的に得られた支配方程式がどのように関係しているのかについて理解を深めることが挙げられる。

2.5 dewetting の方程式について - 補遺

2.5.1 なんのための補遺

本編では、dewetting という現象に着目した。その名の通り de-wetting な現象で、ぬれている状態からぬれていない状態への動的な移行である。

系はどのような形状に落ち着くか？平衡状態で自由エネルギーが最小になるという熱力学の変分原理を踏まえれば[*9]、静的な状況では自由エネルギーの最小化問題に帰着するのではと予想ができる。しかし、一般的な熱力学や統計力学の教科書を開いても、形状の話題は出てこない。では逆に界面関連の教科書にはどう書かれているかを見てみると、あまり熱力学的に厳密には書かれていないような気がする[*10]。さらに、今知りたいのは動的な状況である。すなわち、時間発展でどう形状が変化していくのかに興味がある。本編では、流体の支配方程式を用いることによって、この問題を解決した。しかし、この方法だと全体の自由エネルギーの形がどのような形なのか、あるいはどこから来たのかが今一つよくわからないため、何とも言えないもどかしさがある。

そこで、この補遺では wetting という 3 相間の相分離よりも少ない、2 相間の相分離の話をまとめ、そこから再び dewetting の方程式について考えてみたい[*11]。まずは、面を含む熱力学的系の自由エネルギーを導入する。その後、自由エネルギーの形がどのように書けるかを紹介した後、時間発展を記述する方程式を紹介する。最後に dewetting の方程式が自由エネルギーを小さくする方向に動く式であることを確認する。なお、言うまでもないが、本編に加えこの補遺における間違いも、すべて著者の理解不足によるものである[*12]。

2.5.2 界面と自由エネルギー

熱平衡状態と熱力学

ぽかぽか温かいお部屋がある。このお部屋は十分広く、温度は常に一定である。もちろん、家主は戸締りをしっかりしているし、立派なお部屋なので隙間風などもない。壁は断熱性にも優れている。そんなお部屋の中央に、キンキン冷えた水がビチビチに入ったペットボトル (キャップは閉まっている) を放置してみる。すると、そのうちに、ペットボトルの水は、ぬるくて歯にしみない適温になる。このとき、ペットボトルの中の水は「熱平衡状態」に達したという。以下、「熱平衡状態」のことを単に「平衡状態」と呼ぶ。

…というのは、さすがにあんまりなので、もう少しちゃんとした言葉で言い換えよう。お部屋は外界から「孤立」した「温度一定の環境」であるとする。「孤立」とは、さらにその外の世界とエネルギーのやり取りがないことを指す。この「環境」下で、ペットボトル (密閉容器) を置く。これを「熱力学的系」と呼ぶことにする。系を放置して時間が十分経過すると、系はマクロには変化しなくなる。この状態が「平衡状態」であり、「熱力学」における議論の舞台である。

熱力学では、平衡状態で系がどのように振舞うのかを考える。平衡状態は、容器 (流体) の体積

[*9] 軽く後で触れる (ハズ)。

[*10] 「気がする」というのは、この世に出回っている様々な教科書を、血眼になって探したわけではないためである。何かいい本を知っていたら誰か教えてほしい。

[*11] とはいうものの、初めはこのような補遺を書くつもりはなかった。しかし、班長が担当した文量が想像以上であったため、文量のつりあいをとるべく補遺で文量を増やそうという魂胆から、書くことにしたのである。そもそも、班長の文量が多くなってしまうのは、プロジェクトにおけるタスク量に差があったからなのだが…。

[*12] 数年後に見直したら、どこもかしこも間違いまみれだと気が付いて絶望… ということがないように祈ろう。

や、流体の物質量などといった「示量変数」と、環境の温度などといった「示強変数」の組で、ただ一つ指定される。上で挙げた例は、容器の体積 V、物質量 N、温度 T を指定できる[*13]。つまり、変数の組 $(T;V,N)$ によって系の平衡状態を指定でき、これを $(T;V,N)$ 表示と呼ぶ。このような変数からなる相空間上で定義される関数を、「熱力学関数」という。熱力学では、平衡状態を知るために熱力学関数を調べようというわけである。とりわけ、$(T;V,N)$ 表示においては、「Helmholtz の自由エネルギー」という熱力学関数が重要で、平衡状態ではこれは凸関数である。

まずは、$(T;V,N)$ 表示の熱力学について紹介する。そのためには、「等温操作」という操作の概念を導入する必要がある。

等温操作とは、温度一定の環境下で示量変数を変化させる操作を指す。たとえば、温度 T の環境下で、ピストンを押して、$(V,N) \to (V',N)$ に変化させる、などが考えられる。このとき、操作が実にゆっくりで、操作中はいつでも平衡状態とみなせるような場合は、「等温『準静』操作」という。直観的には、操作の「ゆっくり極限をとる」と思うとよい。

Helmholtz の自由エネルギーは、等温準静操作から定義される。以下、[6] に基づいて自由エネルギーを導入しよう[*14]。系に等温準静操作を行ったとき、「系が外界にする仕事」を $W_{\max}$ とする[*15]。エネルギーの基準点を (V_0,N_0) とすると、Helmholtz の自由エネルギー $F[T;V,N]$ は、基準点からの等温準静操作における仕事として、次のように定義される[*16]。

$$F[T;V,N] := W_{\max}(T;(V,N) \to (V_0,N_0))$$

$(T;V_1,N_1) \to (T;V_2,N_2)$ と等温準静操作を行ったときの、系が外界にする仕事は、

$$W_{\max}(T;(V_1,N_1) \to (V_2,N_2)) = W_{\max}(T;(V_1,N_1) \to (V_0,N_0)) + W_{\max}(T;(V_0,N_0) \to (V_2,N_2))$$

であるから、

$$W_{\max}(T;(V_1,N_1) \to (V_2,N_2)) = F[T;(V_1,N_1)] - F[T;(V_2,N_2)]$$

という関係が成立することもわかる。

さて、先ほど「$(T;V,N)$ 表示において、Helmholtz の自由エネルギーはとりわけ重要」と述べた。実は、$(T;V,N)$ 表示において Helmholtz の自由エネルギーは、それさえ得られれば他のいかなる熱力学関数も得られるという、「完全な熱力学関数」とよばれる特別な量である。言い換えると、$(T;V,N)$ 表示では Helmholtz の自由エネルギーは、系の情報を完全に含んでいる。たとえば、F の T 微分はエントロピー S という量になっており (ただし符号はマイナス)、V 微分は圧力 p になっている (ただし符号はマイナス)。N 微分は化学ポテンシャルとして定義される。これらをまとめて、

$$dF = -SdT - pdV + \mu dN$$

と書くことも多い[*17]。

もう少し、Helmholtz の自由エネルギーについて説明しておく。F の V 微分は圧力 p になっていると述べたが、それは下の図を考えるとわかる。

[*13] とはいうものの、N は自在には変えられない。

[*14] したがって、細かい点を知りたければ [6] を参照されたい。

[*15] 等温操作では、準静操作の仕事が最も大きいため、W に max を添えている。

[*16] 古典力学におけるポテンシャルの定義を思い出せば、何も受け入れ難い話ではない。

[*17] ここでは、微分形式は、何の微分が何だったかをわかりやすくするための記法として使っているだけである。そんなに深い意味はない (と思う)。

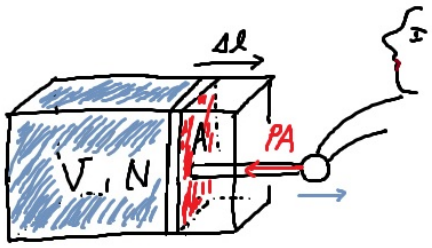


図 2.4

等温準静操作において、系が外界にする仕事は、

$$\Delta W_{\max} := W_{\max}(T;(V,N)\to(V+\Delta V,N)) = F[T;(V,N)] - F[T;(V+\Delta V,N)] =: -\Delta F$$

一方、ピストンには左方向に力 pA を加えている。右方向へ仕事をするとき、外界から系にする仕事は $-pA\Delta l = -p\Delta V$ だから、系が外界にする仕事は、

$$\Delta W_{\max} = p\Delta V$$

とも書ける。これより、$\frac{\Delta F}{\Delta V} = -p$、すなわち

$$\frac{\partial F}{\partial V} = -p$$

だったわけである。

等温準静操作については、系が外界に対して $p\Delta V$ だけ仕事をするために ΔF だけ仕事を与える必要がある。逆に見れば、$p\Delta V$ の仕事を系が外界に対して行うと、その分である ΔF のエネルギーを取り出すことができるともいえる。つまり、Helmholtz の自由エネルギーとは、系から「自由」に取り出すことのできる分のエネルギーと考えられる[*18]。

では、「自由」に取り出すことのできないエネルギーもあるのだろう。 さらに、系のエネルギー U、エントロピー S という量を導入すると、次の関係が成立する[*19]。

$$F = U - TS \tag{2.6}$$

微分形式で書くと、

$$dF = dU - TdS$$

「自由」の意味を考えるために、引き続き等温準静操作に限定しよう。すると上の式は、

$$-dW_{\max} = dU - TdS$$

となる。

$dW_{\max}$ とは、等温準静操作によって系が外界にする仕事であったから、符号をマイナスにすれば、それは外界が系にする仕事である。つまり上式は、外界から系に対して仕事をしても、全てがエネルギーの変化 dU になるわけではなく、謎のエントロピーを含む項「TdS」という分だけ差が

[*18] 自由に取り出せるエネルギーが圧力の仕事に関係しているということが、流体力学における圧力と粘性応力の違いを生んでいるように感じられる。

[*19] [6] では、F、U は操作から定義されるため、この式によってエントロピーが操作的に定義される。

あることを表している[*20]。取り出せるエネルギーと系のエネルギーの変化は必ずしも一致しない。自由に取り出すことができず、系と外界が勝手にやりとりするエネルギーのことを熱と呼ぶ。

エントロピーには様々な顔がある。その一つとして、「系の状態の偏り方を表す指標」という見方がある。それを踏まえて、F、U、S の関係式 (2.6) をもう一度見てみよう。

$$F = U - TS$$

平衡状態において、Helmholtz の自由エネルギーは最小化されると述べた。となると、Helmholtz の自由エネルギーを小さくするには、系のエネルギー U を小さくするか、エントロピー S を大きくすればよいことがわかる。では一体、エントロピーを大きくするとはどういうことだろうか？感覚的に理解するには、情報理論のシャノンエントロピーを思い出してほしい[*21]。エントロピーを大きくするとは、すなわち系の偏りをなくして均一にしようとすると言い換えられる。

界面張力と熱力学

液体とは、分子同士が凝集しているものである。バルク部分の分子達は、仲間たちに囲まれてお気楽だ。しかし、外側にいる分子達はどうだろう。コウテイペンギンのハドル形成を思い浮かべてほしい[*22]。外側にいるペンギンたちは、南極の極寒の空気にさらされている。誤解を恐れず言ってしまえば、液体も同じようなものである。ドゥジェンヌ[*23][3] の言葉を借りれば、液体表面の分子は「不幸」な状態にある。系はなるべくこの「不幸」なものたちを減らしたがる。この「不幸」なものたちがいる一帯を「界面」という。

とはいうものの、ハドルの外側のペンギンたちがいつまでも外側で寒さを耐えているかというと、そういうわけではなく、順番に入れ替わる (らしい)。それと同様に、液体-気体界面の分子が常にそこで「不幸」を耐えているかというと、そういうわけではないだろう。マクロには変化がなくても、ミクロには液体から飛び出して行ったり、逆に入ってくる分子がいるハズである。つまり、界面の「不幸」を最小化するには、系の単なる「エネルギー」だけではなく、「偏り具合」も考慮に入れる必要がある。

では、「不幸」の具合をどう定量化するか。ここで、先ほどの熱力学の話を思い出したい。系の「エネルギー」と「偏り具合」を評価しなければならないならば、「自由エネルギー」の概念がピッタリなのではないだろうか。熱力学的な視点から言えば、平衡状態において系は、「自由エネルギー」を最小化した状態に落ち着く[*24]。ならば、熱力学における「不幸」とは、自由エネルギー最小化を妨げる方向に働く何かであると考えられる。というわけで、「不幸」を定量化するには、界面を含んだ熱力学的系を考え、その界面が広がると自由エネルギーが大きくなるようにペナルティを与えてあげればよい。

Helmholtz の自由エネルギーは等温準静操作による仕事から定義された。ならば、界面の「不幸」による自由エネルギーも同様に導入できるだろう。界面の単位長さあたりに界面張力 γ という「力」が働くものとし、次のような理想的な装置を考える。なお、この装置は [1] を参考にした[*25]。

[*20] この差が熱の実態である。この場合、TdS とは最大吸熱量を表している。

[*21] 具体的に計算して、どのようなときにエントロピーが最大になるかという問題を、2 年生の情報理論の授業で解いたはずである。

[*22] 「コウテイペンギン ハドル」 で検索すれば直ちにヒットする。

[*23] ピエール＝ジル・ドゥジェンヌ (Pierre-Gilles de Gennes)。フランスの物理学者。1991 年にノーベル物理学賞を受賞。噂によると、God と呼ばれるほどにすごい方であったのだとか。

[*24] Gibbs の変分原理より。

[*25] ただし、装置を参考にしただけで、本文はほとんど (いや、まったく) 読んでいない。オンラインの試し読みでちょうど該当ページを見ることができただけである。

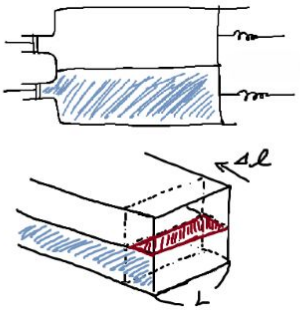


図 2.5

装置の中には 2 種類のつり合った流体が入っている。両者の界面は真っ平で、U 字管のカーブしている部分の高さにあるものとする。左側には補助的なピストンが 2 つ付いている。右側にはバネが二つ付いたピストンが付いていて、これは水平に動かすこともできて、加えて斜めに傾けることもできる。

流体はつり合っていて界面は真っ平だから、左側の 2 つの補助ピストンにかかる圧力は等しい。これを p とおく。つり合いを保つためには、各ピストンにはそれぞれ、右方向に $p\cdot$(ピストンの面積) だけ力を加えていなければならない。一方、右側のピストンは、左側のピストンにかかる力よりも小さくなる。この左側とのピストン差を、界面との接触部分において「界面張力 γ」が作用しているためと考える。すなわち、界面の単位長さあたりにかかる力 γ によって、右側のピストンには γL だけの力が右方向へ加わっている。すると、つり合いを保つためには、ピストンに左方向に γL だけ力を加えていなければならない。

ここで、右側のピストンを準静的に左へ Δl だけ動かし、同時に、界面の面積を任意に変化させるために、右側のピストンを傾ける[*26]。ただし、それぞれの流体と装置の接触面積が一定になるように左側のピストンも左に動かす。これも先ほどと同じように系が外界にする仕事と自由エネルギーの変化分を考慮すれば、$\Delta F = -p\Delta V + \gamma\Delta A$ となり、すなわち、

$$\frac{\partial F}{\partial A} = \gamma$$

が得られる。よって、界面張力 γ は「単位面積あたりの Helmholtz の自由エネルギー」とみることもできる。

以上より、Helmholtz の自由エネルギーは、微分形式で次のように書けることがわかった。

$$dF = -SdT - pdV + \mu dN + \gamma dA$$

ただし、ここでは界面は「真っ平」であったが、現実問題を考えるときは、界面が曲がっていることも多い。その場合は界面の曲率を考慮しなくてはならない。加えて、界面は「線」ではない。ある程度の厚みがある。ここでは理想化して厚みのないものとして扱っている。

なお、ここでは触れなかったが、系が環境と粒子のやりとりをする場合は、Helmholtz の自由エネルギーの代わりにグランドポテンシャルを考える。その場合、温度 (逆温度) が系のエネルギーを調整する未定乗数であったように、「化学ポテンシャル」という量が系の粒子数を調整する未定乗数に相当する。数学的には、グランドポテンシャルは、Helmholtz の自由エネルギーを Legendre 変換したものとも考えられる。

[*26] ピストンを動かすと、それに伴って界面の面積は変化する。これだと人間が意図して面積を変化させられないため、ピストンを傾けて面積を制御するできるようにする。

古典的平衡統計力学

平衡統計力学とは、ミクロな情報をもとに、マクロな熱力学の理論と整合するように組み立てられた理論である。とはいうものの、ミクロな情報といっても真にミクロな粒子の情報を用いるわけではなく、都合の良い確率モデルを考え、それについての期待値がマクロな熱力学的量と一致する (ゆらぎがほぼゼロ) という話である[*27]。したがって基本的には、統計力学の確率モデルに従って現実の粒子が一つ一つ動いているというふうに考えてはいけない。

細かく厳密な導出は、田崎晴明著「統計力学 I」、「統計力学 II」に書かれているため紹介しない。ここでは厳密さを重視せず、古典的[*28]な平衡統計力学のエッセンスのみを紹介する。

古典的な粒子の運動は、位置 $\boldsymbol{r}$ と運動量 $\boldsymbol{p}$ さえわかれば完全に理解できる。粒子が N 個あり、それぞれを $i = 1, 2, ..., N$ とラベル付けする。すると、この粒子系全体の状態は、組 $\{\boldsymbol{r}_1, \boldsymbol{r}_2, ..., \boldsymbol{r}_i, ..., \boldsymbol{r}_N, \boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_i, ..., \boldsymbol{p}_N\}$ で指定できる。以下この組を、$\{\boldsymbol{r}_i, \boldsymbol{p}_i\}$ と書くことにする。

しかし、たくさんある粒子の位置や運動量を測定するのは大変だ。そのため、その状態が出現する「確率」を考えることにしよう[*29]。つまり、連続な確率変数についての確率分布 $P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\})$ を考える。規格化は、位置については可能な領域全体で、運動量については実数領域全体で行う。

$$\int d\boldsymbol{r}_1 ... \int d\boldsymbol{r}_N \int d\boldsymbol{p}_1 ... \int d\boldsymbol{p}_N P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) = 1$$

ここで、二つの事柄を要請する。一つは、系のエネルギー U がハミルトニアン $\mathcal{H}$ の期待値に一致するものとする。すなわち、

$$U = \langle \mathcal{H} \rangle := \int d\boldsymbol{r}_1 ... \int d\boldsymbol{r}_N \int d\boldsymbol{p}_1 ... \int d\boldsymbol{p}_N \mathcal{H} P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\})$$

もう一つは、確率分布はエントロピーを最大にするような形をとるものとする[*30]。エントロピーは、シャノンエントロピーとして、次のように定義する。

$$S := -\int d\boldsymbol{r}_1 ... \int d\boldsymbol{r}_N \int d\boldsymbol{p}_1 ... \int d\boldsymbol{p}_N P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) \log P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\})$$

まとめると、拘束条件：

(1) 規格化

(2) エネルギーの期待値一定

のもと、エントロピーを最大化する分布が、もっともらしい分布である。これは拘束条件付きの最適化問題であるから、Lagrange の未定乗数法に帰着する。すなわち、

[*27] それが耐えがたく、統計力学はあまり好きではないという人もいるらしい。その典型が班長である。

[*28] 古典的の対義語は量子論的である。モダン (modern) ではない。つまり、量子論的に基礎付けられる事柄を、古典論的に議論するという意味である。後に導出される確率分布は、量子論からの近似としても導出可能である。なお、これから紹介する導出の方法は、非常に直観的でわかりやすいとは思うが、実際のところ、この導出によって情報理論から統計力学を基礎付けることができるわけではないということには注意が必要である。

[*29] 古典粒子に確率を導入するというのが、なんともわかりにくいと思う。初めから量子論的に議論すれば、ここら辺の疑問は生じない。

[*30] これを要請と述べたが、実際はそんなことを要請する必要はない。むしろ、エントロピーが最大になるのは結果だろう。

$$\tilde{S} = -\int d\boldsymbol{r}_1 ... \int d\boldsymbol{r}_N \int d\boldsymbol{p}_1 ... \int d\boldsymbol{p}_N P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) \log P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\})$$
$$+\alpha(\int d\boldsymbol{r}_1 ... \int d\boldsymbol{r}_N \int d\boldsymbol{p}_1 ... \int d\boldsymbol{p}_N P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) - 1) + \beta(U - \int d\boldsymbol{r}_1 ... \int d\boldsymbol{r}_N \int d\boldsymbol{p}_1 ... \int d\boldsymbol{p}_N \mathcal{H} P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}))$$
$$= \int d\boldsymbol{r}_1 ... \int d\boldsymbol{r}_N \int d\boldsymbol{p}_1 ... \int d\boldsymbol{p}_N [-P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) \log P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) + \alpha(P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) - 1) + \beta(U - \mathcal{H} P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\})]$$

(2.7)

ただし、α, β は未定乗数である。

$\tilde{s} := -P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) \log P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) + \alpha(P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) - 1) + \beta(U - \mathcal{H} P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\})$ とおく。$\tilde{S} = \int d\boldsymbol{r}_1 ... \int d\boldsymbol{r}_N \int d\boldsymbol{p}_1 ... \int d\boldsymbol{p}_N \tilde{s}$ を P について汎関数微分すると[*31]、

$$\frac{\delta \tilde{S}}{\delta P} = \frac{\partial \tilde{s}}{\partial P} = \log P - 1 + \alpha - \beta \mathcal{H} \tag{2.8}$$

これが 0 になることより、P について解けば、

$$P = e^{\alpha - 1} e^{-\beta \mathcal{H}}$$

一方、$\tilde{S}$ を α について微分すれば、規格化についての拘束条件が出てくる。これに $P = e^{\alpha-1} e^{-\beta\mathcal{H}}$ を代入すると、$1 = e^{\alpha-1} \int ... \int e^{-\beta\mathcal{H}}$ となる。よって、$e^{\alpha-1} = (\int ... \int e^{-\beta\mathcal{H}})^{-1}$ となることがわかる。これより、$P = \frac{e^{-\beta\mathcal{H}}}{\int ... \int e^{-\beta\mathcal{H}}}$ と求められた。

これをよく見てみると、分子に $e^{-\beta\mathcal{H}}$ があり、分母はこれをすべての位置・運動量について和を取ったものになっている。規格化条件から得られた係数であるから、当然と言えば当然なのであるが、分母は分子を規格化していると言える。この分子の $e^{-\beta\mathcal{H}}$ を Boltzmann 因子と呼び、分母の和を分配関数という。なお、exp の肩に乗っている β は、逆温度という物理的意味を持つ定数である。

さらに欲を言えば、規格化因子は無次元であってほしい。今のところ、この規格化因子は (位置)^{3N} · (運動量)^{3N} の次元を持っている。そのため、プランク定数 $h = 2\pi\hbar$ という (位置) · (運動量) の次元を持つ定数で無次元化する。加えて、都合により[*32]分母を $N!$ という因子で割っておくと、古典的なカノニカル分布：

$$P(\{\boldsymbol{r}_i, \boldsymbol{p}_i\}) = \frac{e^{-\beta\mathcal{H}}}{Z} \tag{2.9}$$

が得られる。ただし、分母は規格化因子である分配関数であり、すべての取り得る状態についての和を取った

$$Z = \frac{1}{N! h^{3N}} \int d\boldsymbol{r}_1 ... \int d\boldsymbol{p}_N e^{-\beta\mathcal{H}} \tag{2.10}$$

である。

[*31] $\tilde{S}$ は、確率分布 P についての汎関数であり、α, β についての関数である。汎関数とは、関数を引数に持つ関数のことである。言い換えると、関数の形状全体を与えて一つ値を返す関数である。関数の形状をわずかに変化させたときの汎関数の値の変化量を、汎関数微分で求めることができる。厳密に計算することもできるが、次の公式を覚えておけば、少なくともこの補遺では困ることはない。$F = \int d\boldsymbol{r}\ f[\psi(\boldsymbol{r}), \nabla\psi(\boldsymbol{r})]$ のとき、$\frac{\delta F}{\delta \psi} = \frac{\partial f}{\partial \psi} - \nabla \cdot \frac{\partial f}{\partial(\nabla\psi)}$ である。つまり、$\psi, \nabla\psi$ を一つの変数として扱えばよい。

[*32] いわゆる Gibbs のパラドックスである。$N!$ を付けても付けなくても得られる値は変わらないが、付けたほうが分配関数から得られる熱力学関数の表記がきれいになるという話である。粒子を区別しないために必要という見方もある。

実は、この分配関数 Z という規格化因子は、単なる規格化因子ではなく、非常にいろいろな情報が詰まった因子である。これさえ求められれば、なんと、あらゆる熱力学関数を求めることができる[*33]。たとえば、Helmholtz の自由エネルギーは、

$$F = -\frac{1}{\beta}\log Z \tag{2.11}$$

と求められる。この関係式は、今後も度々登場する。

2.5.3 相分離

相分離とは

突然だが、ちょっと想像してほしい。A 団体と B 団体は激しく対立しあっているものとする。ある日、団体 A が集会を開いた。あっちを見てもこっちを見ても、団体 A 派の人間しかいない。しかし、悪意ある誰かが、団体 B 派の人間数名を拉致し、会場の中央へ放り投げた。さて、その数名はどのような行動をとるだろうか。ただし、会場は大変込み合っていて、団体 B 派の数名は、とてもじゃないが出口にたどり着くことはできないものとする。

模範解答は、「団体 B 派の数名は身を寄せ合う。」である。もっと言えば、円状にギュウギュウと身を寄せ合う。団体 A 派の人間と接触してしまう「不幸な」人が少しでも減るように、団体 B の数名は知恵を働かせるのだ。

「不幸」を最小化する。おや、どこかで聞いたような話である。そういえば、コウテイペンギンのハドルも似たような話だった。なるほど、この例では、団体 A と団体 B の仲が悪いために「分離」し、自然に界面が形成されているのである。

仲が悪い 2 団体の具体例を挙げよう。それは、サラダにかけるようなドレッシングに見られる。ドレッシングは、基本的に酢と油の混合物である。彼らは仲が悪い。だから使用する前は (悪意をもって) よく振って、彼らを強制的に混ぜ合わせてやるのだ[*34]。しかし、時間が十分経つと、再び酢と油が別れてしまっていることに気が付く。すなわち、「酢」と「油」の「相」に「分離」してしまう。このような現象を「相分離」という。「相」という言葉は、対象の定性的な性質によって区別する。相分離の場合は、溶液の「密度」で区別していると言ってよいだろう。このようなパラメータを、「秩序パラメータ」という。

スピノーダル分解

初めに「不幸」という「不安定」な状態にいたからこそ、「幸福」という「安定」な状態へ向かうことができる[*35]。このような移行過程を、「スピノーダル分解」という[*36]。だが、それをどうやって記述するべきだろうか。従来の熱力学では、自由エネルギーは凸関数だ。安定領域しか考えることができず、不安定領域や準安定領域が存在し得ない。となると、もはや手も足も出ない。

平衡状態において自由エネルギーは最小化されると繰り返し述べてきた。ならば、思い切って複数の極値をもつ自由エネルギーを考えてみてはどうだろうか。たとえば、$y = \frac{1}{2}x^4 - x^3 - x^2$ という形状の自由エネルギーはどうだろう。図を描けば $x = 0$ が不安定点、$x = -\frac{1}{2}$ が準安定点、

[*33] そういう意味で、分配関数はまるで魔法の和である。ただし、統計力学的に求められた熱力学関数と、熱力学における熱力学関数は、厳密には一致するわけではない。

[*34] ただし、振ったところで酢と油が一体になるわけではない。混ぜ合わせたところで、仲の悪さが改善されたりはしないのだ。そういう意味で、ドレッシングは振ろうが振るまいが、分離してしまっている。ただし、売られているドレッシングというのは、企業努力でマクロな意味で分離しにくくなるよう様々な工夫がなされているのだろう。

[*35] 不幸を知らずに幸福を知ることはできないのだ (?)。

[*36] なお、準安定から安定へ向かう移行過程は、「核生成」という。

$x = 2$ が安定点であるということがわかる。非常に感覚的だ。この考え方なら、準安定・不安定領域から安定領域へ落ちる様子が記述できそうだ。ただし、注意しなければいけないのは、このように作り出した自由エネルギーの準安定点や不安定点が、物理的な意味をもつという根拠はどこにもないということなのだが。

上で述べたのは、「Landau 理論 (展開)」という考え方である。自由エネルギーを擬似的に秩序パラメータで展開できるものとして、それの形状を見る[*37]。だがそれよりも身近なのは、微視的モデルを考えるということだろう。簡易的な Ising モデルを考え、「平均場近似」と呼ばれる近似を行うというのは、一般的な統計力学の教科書にも載っている。

Landau 理論も平均場近似も、相の密度が空間的に一様なケースを考えることになる。しかしここで思い出さなければならないことは、実際の界面には「厚み」があるということである。2 つのバルク相 (秩序相) の間がきれいにキッパリと分断しているわけではない。前に、界面の分子が「不幸」を耐え続けているわけではないと述べた。飛び出したり入ってきたりする分子がいる。つまり濃度変化がある。バルク相の狭間にある、界面というある程度の厚みをもった空間で、濃度は連続的に変化している。そういう意味で、Landau 理論も平均場近似も、界面の相分離を記述しきれないのである。

というわけで、まずは、スピノーダル分解が起こるときのもっともらしい自由エネルギーの形はどのようなものかを考えてみたい。紹介するのは、Landau 理論を発展させたとも言える、Ginzburg-Landau 理論 (展開) である。

変分法を用いた自由エネルギーの上限の導出

Ginzburg-Landau 理論を紹介する前に、自由エネルギーについての不等式をひとつ導出しておきたい。

先ほど述べた通り、これからは、従来の熱力学では記述できない現象を考えるため、微視的な格子モデルを考えたい。しかし一般には、そのモデルにおける分配関数は、格子点同士の相互作用のせいで計算が難しいことが多い。平均場近似では、大胆にも相互作用する項がなくなるように全体の平均で置き換えてしまうが、ここではそうはしない。変分法を用いて、そのモデルにおける「真」の自由エネルギーの代わりに、もっと扱いやすい自由エネルギーを導入するということを考える。なお、以下の不等式の導出は [5] を参考にしている。

考えるモデルにおける「真」のハミルトニアンが $\mathcal{H}$ で与えられるとする。すると、「真」の確率分布は、$P = \frac{e^{-\beta\mathcal{H}}}{Z}$ である。ここで、これよりも扱いやすいハミルトニアンがあり、これを $\mathcal{H}_0$ とおく。これらの差を $V := \mathcal{H} - \mathcal{H}_0$ とする。

このとき、「真」の分配関数は、$Z = \int d\boldsymbol{r}_1 ... \int d\boldsymbol{p}_N e^{-\beta\mathcal{H}}$ であるが、$\mathcal{H} = \mathcal{H}_0 - V$ を代入すると、

$$Z = \int d\boldsymbol{r}_1 ... \int d\boldsymbol{p}_N e^{-\beta(\mathcal{H}_0+V)} = \int d\boldsymbol{r}_1 ... \int d\boldsymbol{p}_N e^{-\beta\mathcal{H}_0} \cdot e^{-\beta V} \tag{2.12}$$

ここで、扱いやすいハミルトニアン $\mathcal{H}_0$ の系に対応する確率分布は、$P_0 = \frac{e^{-\beta\mathcal{H}_0}}{Z_0}$、分配関数は $Z_0 = \int d\boldsymbol{r}_1 ... \int d\boldsymbol{p}_N e^{-\beta\mathcal{H}_0}$ である。これを踏まえると、(2.12) は、

[*37] 若干天下り的に秩序パラメータで展開できるというのが基本的な考え方である、という印象だが、微視的モデルからも導出はできる。

$$
\begin{aligned}
Z &= \int d\boldsymbol{r}_1 ... \int d\boldsymbol{p}_N e^{-\beta\mathcal{H}_0} \cdot e^{-\beta V} \\
&= Z_0 \int d\boldsymbol{r}_1 ... \int d\boldsymbol{p}_N \frac{e^{-\beta\mathcal{H}_0}}{Z_0} \cdot e^{-\beta V} \\
&= Z_0 \int d\boldsymbol{r}_1 ... \int d\boldsymbol{p}_N P_0 \cdot e^{-\beta V} \\
&=: \langle e^{-\beta V} \rangle_0 Z_0
\end{aligned}
\tag{2.13}
$$

となる。ここで、(相加平均)≥(相乗平均) の関係から、$\langle e^{-\beta V} \rangle_0 \geq e^{\langle -\beta V \rangle_0}$ となるから[*38]、

$$
\begin{aligned}
& Z \geq e^{\langle -\beta V \rangle_0} Z_0 \\
\Leftrightarrow\ & \frac{Z}{Z_0} \geq e^{\langle -\beta V \rangle_0} \\
\Leftrightarrow\ & \frac{1}{\beta} \log Z - \frac{1}{\beta} \log Z_0 \geq \langle V \rangle_0 \\
\Leftrightarrow\ & -F + F_0 \geq \langle V \rangle_0 \\
\Leftrightarrow\ & F \leq F_0 + \langle \mathcal{H} - \mathcal{H}_0 \rangle_0
\end{aligned}
\tag{2.14}
$$

これより、都合の良いハミルトニアン $\mathcal{H}_0$ の自由エネルギーは、「真」のハミルトニアンよりの小さくなることはない。したがって、右辺の $F_0 + \langle \mathcal{H} - \mathcal{H}_0 \rangle_0$ を最小化することで、「真」の自由エネルギーの最小の上限を求めることができる。この不等式を、Gibbs-Bogoliubov[*39]の不等式という。

Ginzburg-Landau 展開

2 相間の相分離を考えるため、微視的なモデルとして、簡易的な Ising モデルを考える。ここでは、[2] の 2 章の Ising モデルを考えてみよう。

まず、空間を格子状に分割する。それぞれの格子を $i = 1, 2, ...$ とラベル付けする。粒子は 2 種類あり、濃度変数 s_i は、A 粒子なら 0、B 粒子なら 1 をとる。このときのハミルトニアンは、2 粒子間の相互作用を J_{ij} と書くことにすれば、

$$
\mathcal{H} = \frac{1}{2} \sum_{ij} J_{ij} s_i (1 - s_j) \tag{2.15}
$$

のように書けるだろう。しかし、これは、i と j を分けることができず、分配関数を計算するのは容易ではない。そこで、都合の良いハミルトニアンとして、i と j がペアにならないよう、

$$
\mathcal{H}_0 = \frac{1}{\beta} \sum_i b_i s_i
$$

と選んでみる。ただし β は逆温度であり、パラメータ b_i は後に決定される。このハミルトニアンに対応する系の自由エネルギー、分配関数は、次のように書ける。

[*38] 高校数学のアレである。

[*39] カタカナで書けばボゴリューボフ。言いにくい。

$$F_0 = -\frac{1}{\beta}\log Z_0 \tag{2.16}$$

$$Z_0 = \prod_i \sum_{s_i=0,1} e^{-b_i s_i} = \prod_i (1+e^{-b_i}) = \prod_i (\frac{e^{b_i}}{1+e^{b_i}})^{-1} = \prod_i (1-\frac{1}{1+e^{b_i}})^{-1} =: \prod_i (1-\phi_i)^{-1} \tag{2.17}$$

ただし、$\phi_i := (1+e^{b_i})^{-1}$ と置いた。

この系は、組 $\{s_1, s_2, s_3, ...\}$ によって状態が決まる。ゆえに、系の確率分布は $P_0 := \frac{e^{-\beta\mathcal{H}_0}}{Z_0} = \frac{\Pi_i e^{-b_i s_i}}{Z_0}$ と書けて、物理量 (確率変数)A の期待値は、$\langle A\rangle_0 := \prod_i \sum_{s_i=0,1} A_i P_0 = \frac{1}{Z_0}\prod_i \sum_{s_i=0,1} A_i(\Pi_j e^{-b_j s_j})$ と書ける[*40]。

実は、$\phi_i = \langle s_i\rangle_0$ という関係が成立する[*41]。したがって、ϕ_i は、0 から 1 の値をとり得るため、これは、非一様な密度分布を表現できることがわかる。

これより、都合良く選んだ系の自由エネルギー F_0 は、

$$\begin{aligned} F_0 &= -\frac{1}{\beta}\log Z_0 \\ &= -\frac{1}{\beta}\log[\prod_i (1-\phi_i)^{-1}] \\ &= \frac{1}{\beta}\sum_i \log(1-\phi_i) \end{aligned} \tag{2.18}$$

また、$\langle s_i\rangle_0 = \phi_i, \langle s_i s_j\rangle_0 = \phi_i\phi_j$[*42]という関係を用いると、

$$\begin{aligned} \langle \mathcal{H}-\mathcal{H}_0\rangle_0 &= \langle [\frac{1}{2}\sum_{ij} J_{ij}s_i(1-s_j) - \frac{1}{\beta}\sum_i b_i s_i]\rangle_0 \\ &= \frac{1}{2}\sum_{ij} J_{ij}\langle s_i\rangle_0 - \frac{1}{2}\sum_{ij}\langle s_i s_j\rangle_0 - \frac{1}{\beta}\sum_i b_i\langle s_i\rangle_0 \\ &= \frac{1}{2}\sum_{ij} J_{ij}\phi_i - \frac{1}{2}\sum_{ij}\phi_i\phi_j - \frac{1}{\beta}\sum_i b_i\phi_i \\ &= \frac{1}{2}\sum_{ij} J_{ij}\phi_i(1-\phi_j) - \frac{1}{\beta}\sum_i b_i\phi_i \end{aligned} \tag{2.19}$$

となる。

以上より、Gibbs-Bogoliubov の不等式から、「真」の自由エネルギーの上限が求められる。上限を F と書くことにすると、

[*40] $\prod_i \sum_{s_i=0,1} := \sum_{s_1=0,1}\sum_{s_2=0,1}...$ という書き方をしていることに注意。

[*41] 証明：$\sum$ や $\prod$ の記号が若干ややこしいため、混乱しそうになったら実際に書き下して確認するとよい。なお、確率変数 $s_i(s_i = 0, 1)$ についての期待値をとるにあたって、s_i と他のインデックスは分けて計算する。定義より $\langle s_i\rangle_0 = \frac{1}{Z_0}\prod_i \sum_{s_i=0,1} s_i(\Pi_j e^{-b_j s_j}) = \frac{1}{Z_0}\sum_{s_i=0,1} s_i e^{-b_i s_i}\cdot(\prod_{j\neq i}\sum_{s_j=0,1} e^{-b_j s_j}) = \frac{1}{Z_0}1\cdot e^{-b_i\cdot 1}\{\prod_{j\neq i}(e^{-b_j\cdot 0}+e^{-b_j\cdot 1})\} = \frac{1}{Z_0}e^{-b_i}\prod_{j\neq i}(1+e^{-b_j})$. 分配関数 $Z_0 = \prod_j(1+e^{-b_j})$ より、分母と分子で、$\prod_{j\neq i}(1+e^{-b_j})$ が打ち消し合って、分母は i 番目の項のみ残る。以上より、$\langle s_i\rangle_0 = \frac{e^{-b_i}}{1+e^{-b_i}} = \frac{1}{e^{b_i}+1} =: \phi_i$

[*42] $\langle s_i s_j\rangle_0 = \langle s_i\rangle_0\langle s_j\rangle_0$。これも計算により確かめることができる。

$$
\begin{aligned}
F &= F_0 + \langle \mathcal{H} - \mathcal{H}_0 \rangle_0 \\
&= \frac{1}{\beta} \sum_i \log(1-\phi_i) + \frac{1}{2} \sum_{ij} J_{ij} \phi_i (1-\phi_j) - \frac{1}{\beta} \sum_i b_i \phi_i
\end{aligned}
\tag{2.20}
$$

である。

ここで、関係式 $\phi_i = (1+e^{b_i})^{-1}$ より、$e^{b_i} = \frac{1-\phi_i}{\phi_i}$ であるから、両辺対数をとると、$b_i = \log\left(\frac{1-\phi_i}{\phi_i}\right)$ である。これを F に代入すると、

$$
\begin{aligned}
F &= \frac{1}{\beta} \sum_i \log(1-\phi_i) + \frac{1}{2} \sum_{ij} J_{ij} \phi_i (1-\phi_j) - \frac{1}{\beta} \sum_i \phi_i \log\left(\frac{1-\phi_i}{\phi_i}\right) \\
&= \frac{1}{\beta} \sum_i [(1-\phi_i)\log(1-\phi_i) + \phi_i \log \phi_i] + \frac{1}{2} \sum_{ij} J_{ij} \phi_i (1-\phi_j)
\end{aligned}
\tag{2.21}
$$

これが、求める自由エネルギーの上限であり、「真」の自由エネルギーの近似とみなすことができる。したがって以下、「真」の自由エネルギーの代わりにこちらの自由エネルギー (2.21) を最小化することを考えることにする。

とはいうものの、差分方程式より微分方程式の方が何かと扱いやすいため、格子の幅を限りなく小さくして連続極限をとることにする。すると、離散的な平均密度 ϕ_i ではなく、連続的な密度場 $\phi(\boldsymbol{r})$ を用いて自由エネルギーを記述することができる[*43]。

まず、連続極限をとるにあたって、i と j が結合していると変形が難しいため、次の変形を行っておく。

$$
J_{ij}\phi_i(1-\phi_j) = \frac{1}{2}[(\phi_i - \phi_j)^2 - (\phi_i^2 + \phi_j^2) + 2\phi_i] \tag{2.22}
$$

すると、

$$
F = \frac{1}{\beta} \sum_i [(1-\phi_i)\log(1-\phi_i) + \phi_i \log \phi_i] + \frac{1}{4} \sum_{ij} J_{ij}[(\phi_i - \phi_j)^2 - (\phi_i^2 + \phi_j^2) + 2\phi_i] \tag{2.23}
$$

と書ける。

位置についての関数 $\phi(\boldsymbol{r})$ が、格子点 i における位置 $\boldsymbol{r_i}$ で、$\phi(\boldsymbol{r_i}) = \phi_i$ の値をとるとする。また、隣り合った格子点 $i, i+1$ について、$\Delta \boldsymbol{r}_i := (\boldsymbol{r}_{i+1} - \boldsymbol{r}_i)$ とする。ゆえに、格子の幅は $|\Delta \boldsymbol{r}_i|$ である。

ここで、すべての格子点の幅について、系全体の大きさを保ちながら、$|\Delta \boldsymbol{r}_i| \to 0 (i = 1, 2, ...)$ の極限をとる[*44]。すると、

$$
\phi_i \to \phi(\boldsymbol{r})
$$

$$
\sum_i |\Delta \boldsymbol{r}_i|^3 (...) \to \int d\boldsymbol{r} (...)
$$

[*43] 多自由度系において、格子の幅を限りなく小さくして場の方程式を得る過程を思い出すとよい。系の自由エネルギー F は、最終的に自由エネルギー密度の積分の形で書ける。

[*44] たとえば、系の大きさを V、セルの数を N とすると、$V = N|\Delta \boldsymbol{r}_i|^3$ である。ゆえに、$|\Delta \boldsymbol{r}_i| \to$ として系の体積 V が変わらないようにするためには、$N \to \infty$ とする必要がある。したがって、格子点は連続となることがわかる。

また、隣り合った格子点 $i, i+1$ については、

$$\phi_{i+1} - \phi_i = \phi(\boldsymbol{r}_i + \Delta\boldsymbol{r}_i) - \phi(\boldsymbol{r}_i) \approx \nabla\phi(\boldsymbol{r_i}) \cdot \Delta\boldsymbol{r}_i$$

と書けるから、格子点同士の相互作用は最近接相互作用のみを考えることにする。
見やすさのために $a := |\Delta\boldsymbol{r}_i|$ とおく。すると、自由エネルギーは、

$$\begin{aligned}
F &= \frac{1}{\beta}\sum_i[(1-\phi_i)\log(1-\phi_i) + \phi_i\log\phi_i] + \frac{1}{4}\sum_{ij} J_{ij}[(\phi_i-\phi_j)^2 - (\phi_i^2+\phi_j^2) + 2\phi_i] \\
&= \frac{1}{\beta}\sum_i a^3[\frac{1}{a^3}\{(1-\phi_i)\log(1-\phi_i)+\phi_i\log\phi_i\}] + \sum_i a^3[\frac{1}{4a^3}\sum_j J_{ij}(\phi_i-\phi_j)^2 - \frac{2}{4}\frac{\sum_j J_{ij}}{a^3}\phi_i^2 + \frac{2}{4}\frac{\sum_j J_{ij}}{a^3}\phi_i] \\
&\to \frac{1}{\beta}\int d\boldsymbol{r}[\frac{1}{a^3}\{(1-\phi)\log(1-\phi)+\phi\log\phi\}] + \int d\boldsymbol{r}[\frac{a^2}{4a^3}(\sum_j J_{ij})|\nabla\phi|^2 - \frac{1}{2}\frac{\sum_j J_{ij}}{a^3}\phi^2 + \frac{1}{2}\frac{\sum_j J_{ij}}{a^3}\phi]
\end{aligned} \tag{2.24}$$

ここで、$J := \sum_j J_{ij}, B := \frac{J}{2a}$ とすると (以下極限を取った際の $\to$ は省略する)、

$$\begin{aligned}
F &= \frac{1}{\beta}\int d\boldsymbol{r}[\frac{1}{a^3}\{(1-\phi)\log(1-\phi)+\phi\log\phi\}] + \int d\boldsymbol{r}[\frac{1}{2}\frac{J}{2a}|\nabla\phi|^2 - \frac{1}{2}\frac{J}{a^3}\phi^2 + \frac{1}{2}\frac{J}{a^3}\phi] \\
&= \int d\boldsymbol{r}\frac{1}{a^3}[\frac{1}{\beta}\{(1-\phi)\log(1-\phi)+\phi\log\phi\} + \frac{J}{2}\phi(1-\phi)] + \int d\boldsymbol{r}[\frac{B}{2}|\nabla\phi|^2]
\end{aligned} \tag{2.25}$$

このように、密度 ϕ を場の量として、密度勾配を含む汎関数として自由エネルギー $F[\phi, \nabla\phi]$ を書くことができた。密度 $\phi(\boldsymbol{r})$ が一様である場合は、$|\nabla\phi|^2 = 0$ であり、2 つめの積分項は寄与しない。

ここで、Landau 理論を持ち出そう。詳しくは述べないが、Landau 理論では、相の濃度と臨界濃度の差分を秩序パラメータとして自由エネルギーを展開することを考える。そこで、上で得たエネルギー汎関数を、ϕ が臨界濃度 $\frac{1}{2}$ の近傍の値を取るとして展開することを考えてみたい。$\psi := \phi - \frac{1}{2}$ と代入した後、$\psi = 0$ の周りで log を展開する[*45]。

$$\begin{aligned}
F &= \int d\boldsymbol{r}\frac{1}{a^3}[\frac{1}{\beta}\{(\frac{1}{2}-\psi)\log\left(\frac{1}{2}-\psi\right)+(\psi+\frac{1}{2})\log\left(\psi+\frac{1}{2}\right)\}+\frac{J}{2}(\psi+\frac{1}{2})(\frac{1}{2}-\psi)] + \int d\boldsymbol{r}[\frac{B}{2}|\nabla\psi|^2] \\
&= \int d\boldsymbol{r}\frac{1}{a^3}[\frac{1}{\beta}\{\log\frac{1}{2} + 2\psi^2 + \frac{4}{3}\psi^4\} - \frac{J}{2}(\psi^2 - \frac{1}{4})] + \int d\boldsymbol{r}[\frac{B}{2}|\nabla\psi|^2] \\
&= \int d\boldsymbol{r}[\frac{1}{a^3}(\frac{1}{\beta}\log\frac{1}{2} + \frac{J}{8}) + (\frac{2}{a^3\beta} - \frac{J}{2a^3})\psi^2 + \frac{4}{3a^3\beta}\psi^4 + \frac{B}{2}|\nabla\psi|^2]
\end{aligned} \tag{2.26}$$

ここで、定数項を省略し、$\epsilon := \frac{J-4/\beta}{a^3}$、$c := \frac{16}{3a^3\beta}$ と置くと、

[*45] $\frac{\partial^n}{\partial\psi^n}\log\left(\frac{1}{2}+\psi\right) = (-1)^{n-1}(n-1)!2^n \quad (n = 1, 2, ...)$ であり、$\log\left(\frac{1}{2}+\psi\right) = \log\frac{1}{2} + 0!2^1\psi - 1!2^2\frac{\psi^2}{2!} + 2!2^3\frac{\psi^3}{3!} - 3!2^4\frac{\psi^4}{4!} + \mathcal{O}(\psi^5)$ である。ゆえに、$(\frac{1}{2}-\psi)\log\left(\frac{1}{2}-\psi\right) + (\psi+\frac{1}{2})\log\left(\psi+\frac{1}{2}\right) = (\frac{1}{2}-\psi)\{\log\frac{1}{2} - 2\psi - 2\psi^2 - \frac{8}{3}\psi^3 - 4\psi^4\} + (\psi+\frac{1}{2})\{\log\frac{1}{2} + 2\psi - 2\psi^2 + \frac{8}{3}\psi^3 - 4\psi^4\} = \log\frac{1}{2} + 2\psi^2 + \frac{4}{3}\psi^4$

$$
\begin{aligned}
F &= \int d\boldsymbol{r}[\frac{1}{2}\frac{4/\beta - J}{a^3}\psi^2 + \frac{1}{4}\frac{16}{3a^2\beta}\psi^4 + \frac{B}{2}|\nabla\psi|^2] \\
&= \int d\boldsymbol{r}[-\frac{\epsilon}{2}\psi^2 + \frac{1}{4}c\psi^4 + \frac{B}{2}|\nabla\psi|^2]
\end{aligned}
\tag{2.27}
$$

このように、密度勾配を含むようなエネルギー汎関数の展開を、Ginzburg-Landau 展開という。

さて、自由エネルギー汎関数が求められたため、これを最小化することを考えてみたい。そのためには、自由エネルギー F の停留条件を解けばよい[*46]。$f[\psi, \nabla\psi] := -\frac{\epsilon}{2}\psi^2 + \frac{1}{4}c\psi^4 + \frac{B}{2}|\nabla\psi|^2$ とおくと、

$$
\begin{aligned}
&\frac{\delta F}{\delta\psi(\boldsymbol{r})} = \frac{\partial f}{\partial\psi} - \nabla\cdot\frac{\partial f}{\partial(\nabla\psi)} = 0 \\
&\Rightarrow -\epsilon\psi + c\psi^3 - B\nabla^2\psi = 0
\end{aligned}
\tag{2.28}
$$

ここで、界面から十分に離れた点では濃度が一様に平衡の濃度値をとり、濃度勾配はゼロであることを要請する。すると、その平衡値は、

$$
\begin{aligned}
&-\epsilon\psi_0 + c\psi_0^3 = 0 \\
&\Rightarrow \psi_0 = \pm\sqrt{\frac{\epsilon}{c}}
\end{aligned}
\tag{2.29}
$$

とわかる。

では、界面付近ではどのように密度変化が起こっているだろうか？簡単のために z 軸方向のみを考え、(2.28) を解いてみることにしよう。すなわち、

$$
-\epsilon\psi + c\psi^3 - B\frac{\partial^2\psi}{\partial z^2} = 0 \tag{2.30}
$$

を解くことを考える。

両辺に $\frac{\partial\psi}{\partial z} =: \psi'$ をかけると、

$$
\begin{aligned}
&(-\epsilon\psi + c\psi^3)\psi' - B\psi''\psi' = 0 \\
&\Leftrightarrow \frac{d}{dz}(-\frac{\epsilon}{2}\psi^2 + \frac{c}{4}\psi^4) = B\frac{d}{dz}\frac{(\psi')^2}{2}
\end{aligned}
\tag{2.31}
$$

両辺 z で積分すると、

$$
\frac{B}{2}(\frac{d\psi}{dz})^2 = -\frac{\epsilon}{2}\psi^2 + \frac{c}{4}\psi^4 + C \ (C \text{ は積分定数}) \tag{2.32}
$$

ここで、先ほどと同様に、境界条件として、$\psi(z\to\pm\infty) = \pm\sqrt{\frac{\epsilon}{c}}$ を課す。すると、積分定数は $C = \frac{\epsilon^2}{4c}$ と決まり[*47]、

[*46] ここでは簡単のために、系全体の平均濃度が臨界濃度に等しく、化学ポテンシャル項が必要ないケースを考える。

[*47] $0 = -\frac{\epsilon}{2}(\sqrt{\frac{\epsilon}{c}})^2 + \frac{c}{4}(\sqrt{\frac{\epsilon}{c}})^4 + C$

$$\begin{aligned}\frac{B}{2}(\frac{d\psi}{dz})^2 &= \frac{c}{4}\psi^4 - \frac{\epsilon}{2}\psi^2 + \frac{\epsilon^2}{4c} \\ &= \frac{c}{4}(\psi^4 - \frac{2\epsilon}{c}\psi^2 + \frac{\epsilon^2}{c^2}) \\ &= \frac{c}{4}(\psi^2 - \frac{\epsilon}{c})^2\end{aligned} \tag{2.33}$$

よって、

$$\begin{aligned}&\frac{\partial\psi}{\partial z} = \pm\sqrt{\frac{c}{2B}}(\psi^2 - \frac{\epsilon}{c}) \\ &\Leftrightarrow \frac{\psi'}{(\psi - \sqrt{\epsilon/c})(\psi + \sqrt{\epsilon/c})} = \pm\sqrt{\frac{c}{2B}} \\ &\Leftrightarrow \frac{\psi'}{\psi - \sqrt{\epsilon/c}} - \frac{\psi'}{\psi + \sqrt{\epsilon/c}} = \pm 2\sqrt{\frac{\epsilon}{2B}}\end{aligned} \tag{2.34}$$

両辺 z で積分すると、

$$\log|\frac{\psi - \sqrt{\epsilon/c}}{\psi + \sqrt{\epsilon/c}}| = \pm 2\sqrt{\frac{\epsilon}{2B}}z \tag{2.35}$$

ただし、$\psi(z=0)=0$ とし、積分定数を 0 とした。あとは、この式を $\psi(z)$ について解くだけである。ここで、符号について少し議論しておこう。$-\sqrt{\epsilon/c} \le \psi \le \sqrt{\epsilon/c}$ より、$\psi - \sqrt{\epsilon/c} \le 0, \psi + \sqrt{\epsilon/c} \ge 0$ となるから、$|\frac{\psi-\sqrt{\epsilon/c}}{\psi+\sqrt{\epsilon/c}}| = -\frac{\psi-\sqrt{\epsilon/c}}{\psi+\sqrt{\epsilon/c}}$ である。また、境界条件を再び考えると、$\psi(z\to\infty) = \sqrt{\epsilon/c}$ のとき (2.35) の左辺はマイナスに発散する。右辺がマイナスに発散するのは、$\pm$ のうちマイナスのときである。同様に、$\psi(z\to-\infty) = -\sqrt{\epsilon/c}$ のとき (2.35) の左辺はプラスに発散する。右辺がプラスに発散するのは、$\pm$ のうちマイナスのときである。以上を踏まえて符号を選ぶ。

$$\begin{aligned}&\log\left(-\frac{\psi - \sqrt{\epsilon/c}}{\psi + \sqrt{\epsilon/c}}\right) = -2\sqrt{\frac{\epsilon}{2B}}z \\ &\Leftrightarrow -\frac{\psi - \sqrt{\epsilon/c}}{\psi + \sqrt{\epsilon/c}} = e^{-2\sqrt{\frac{\epsilon}{2B}}z} \\ &\Leftrightarrow \psi = \sqrt{\frac{\epsilon}{c}}\frac{1 - e^{-2\sqrt{\frac{\epsilon}{2B}}z}}{1 + e^{-2\sqrt{\frac{\epsilon}{2B}}z}} \\ &\quad = \sqrt{\frac{\epsilon}{c}}\frac{e^{\sqrt{\frac{\epsilon}{2B}}z} - e^{-\sqrt{\frac{\epsilon}{2B}}z}}{e^{\sqrt{\frac{\epsilon}{2B}}z} + e^{-\sqrt{\frac{\epsilon}{2B}}z}} \\ &\quad = \sqrt{\frac{\epsilon}{c}}\tanh\left(\frac{z}{\sqrt{2B/\epsilon}}\right)\end{aligned} \tag{2.36}$$

ここで、$\xi := \sqrt{\frac{2B}{\epsilon}}$ とおくと、結局、

$$\psi(z) = \sqrt{\frac{\epsilon}{c}} \tanh\left(\frac{z}{\xi}\right) \tag{2.37}$$

なお、この ξ は界面領域のスケール値である。グラフにプロットしてみればわかるように、片方の相の平衡値から、もう片方の相の平衡値まで、界面付近でなめらかに推移していることがわかる。どうやら、Ginzburg-Landau 展開から、うまく相分離の様子を記述することができたようだ。

さらに、界面エネルギーを計算することもできる。思ったよりも長くなってしまったため、ここでは省略するが、界面張力は界面の自由エネルギーであるから、全体の自由エネルギー F から密度の一様なバルク部分の自由エネルギーを引いたものとして計算できる。

Allen-Cahn 方程式と Cahn-Hilliard 方程式

さて、前節では、なんだかもっともらしい相分離の様子をみることができた。だが、きっと読者はモヤモヤしていることだろう。スピノーダル分解、すなわち不安定領域から安定領域への移行は、結局どうやって起こっているのだろうか？これは、先のエネルギー汎関数と睨めっこしているだけでは、よくわからない。

そこで、時間発展として自由エネルギーが小さくなる方向に濃度変化が起こると考えて、イイ感じの方程式を模索してみることにしよう。先ほどの Ginzburg-Landau 理論を踏まえて、自由エネルギー汎関数は、

$$F := \int d\boldsymbol{r}[-\frac{\epsilon}{2}\psi^2 + \frac{c}{4}\psi^4 + \frac{B}{2}|\nabla\psi|^2] \tag{2.38}$$

と定義する。濃度 ψ が、時間発展によって自由エネルギーが減少する方向に動けばよいのだから、直観的には、

$$\begin{aligned}
&\frac{\partial\psi}{\partial t} = -\frac{\delta F}{\delta\psi} \\
&= -\{-\epsilon\psi + c\psi^3 + B\nabla^2\psi\} \\
&= -(c\psi^3 - \epsilon\psi) - B\nabla^2\psi
\end{aligned} \tag{2.39}$$

と書けそうだ。

少々 ϵ などのパラメータが煩わしいため、無次元化してみる。ψ のスケールを ψ_0、空間と時間のスケールをそれぞれ r_0, t_0 とおく。すなわち、無次元量にチルダをつけると、

$$\psi = \psi_0\tilde{\psi}$$

$$t = t_0\tilde{t}$$

$$\boldsymbol{r} = r_0\tilde{\boldsymbol{r}}$$

である。すると (2.39) の左辺は、

$$\begin{aligned}\frac{\partial\psi}{\partial t} &= \psi_0\frac{\partial\tilde{\psi}}{\partial t}\\ &= \psi_0\frac{\partial\tilde{\psi}}{\partial\tilde{t}}\frac{\partial\tilde{t}}{\partial t}\\ &= \psi_0\frac{1}{t_0}\frac{\partial\tilde{\psi}}{\partial\tilde{t}}\end{aligned} \tag{2.40}$$

一方、右辺は、

$$\begin{aligned}&-(c\psi^3-\epsilon\psi)-B\nabla^2\psi = -(c\psi_0^3\tilde{\psi}^3-\epsilon\psi_0\tilde{\psi})-B\frac{1}{r_0^2}\psi_0\tilde{\nabla}^2\tilde{\psi}\\ &= -\psi_0(c\psi_0^2\tilde{\psi}^3-\epsilon\tilde{\psi})-\frac{B}{r_0^2}\psi_0\tilde{\nabla}^2\tilde{\psi}\end{aligned} \tag{2.41}$$

よって、(2.39) は、

$$\begin{aligned}&\psi_0\frac{1}{t_0}\frac{\partial\tilde{\psi}}{\partial\tilde{t}} = -\psi_0(c\psi_0^2\tilde{\psi}^3-\epsilon\tilde{\psi})-\frac{B}{r_0^2}\psi_0\tilde{\nabla}^2\tilde{\psi}\\ &\frac{\partial\tilde{\psi}}{\partial\tilde{t}} = -(t_0c\psi_0^2\tilde{\psi}^3-t_0\epsilon\tilde{\psi})-\frac{t_0B}{r_0^2}\tilde{\nabla}^2\tilde{\psi}\end{aligned} \tag{2.42}$$

ここで、それぞれが無次元化されるようにスケールの値を選ぶ。まず、右辺第 2 項に注目して $t_0=\frac{1}{\epsilon}$ と選ぶ。すると、芋づる式に $\psi_0=\sqrt{\frac{\epsilon}{c}}, r_0=\sqrt{\frac{B}{\epsilon}}$ と選べる[*48]。チルダを省略すると、無次元化された方程式は、

$$\frac{\partial\psi}{\partial t} = -(\psi^3-\psi)-\nabla^2\psi \tag{2.43}$$

となる。実はこの形の方程式には名前がついていて、Allen-Cahn 方程式という[*49]。

お名前がついているということは、これはイイ感じに相分離を記述するのだろう。実際、この方程式はイイ感じに相分離を記述する[*50]。とはいうものの、これは物質の総量が保存されない。どういうことかを説明しよう。かなり前に述べた、仲が悪い団体 A、B の話を少し思い出してほしい。団体 A の会場の出入り口が解放されていれば、会場全体の人数は変動するだろう。ゆえに会場に入れられた団体 B の数名は、出口めがけて逃げ出すことができる。一方、出入り口が鉄板で封鎖されていればどうだろう。会場全体の人数の変動はない。ゆえに、団体 B の数名はどこかで縮こまっていることしかできない。容器に入ったドレッシングを見ていたら、いつの間にか油が消滅してただの味付き酢になった、だなんてことはあり得ない。

では、この Allen-Cahn 方程式の「保存系バージョン」を考えることはできないだろうか？そこで思い出すべきは、流体力学や電磁気学など、様々な分野によく現れる保存則 (連続の式) である。

[*48] ψ_0 は密度の平衡値、r_0 は界面幅のスケールになっている。
[*49] Time Dependent Ginzburg-Landau 方程式 (TDGL) の一人でもある。
[*50] 気になる人は調べてみよう。

先ほどから密度の話をしていたため、物質量 (総量) の保存について考えてみよう。物質量が保存しているならば、その密度について、

$$\frac{\partial \psi}{\partial t} = -\nabla \cdot \boldsymbol{J} \tag{2.44}$$

の形で書けるはずである。ただし $\boldsymbol{J}$ は流束である。この流束 $\boldsymbol{J}$ は、物質の流れとその量を表している。物質が流れる方向を $\boldsymbol{v}$ と書けば、$\boldsymbol{J} = \psi \boldsymbol{v}$ とも書けるだろう。では、物質はどこへ向かって流れるのか？それはもちろん、自由エネルギーを小さくする方向へ流れると考えるのが自然だ。というわけで、

$$\boldsymbol{J} := -\nabla \frac{\delta F}{\delta \psi} \tag{2.45}$$

と置いてみる。すると、

$$\begin{aligned}
&\frac{\partial \psi}{\partial t} = \nabla^2 \frac{\delta F}{\delta \psi} \\
&= \nabla^2 [(c\psi^3 - \epsilon\psi) + B\nabla^2 \psi] \\
&= \nabla^2 [c\psi^3 - \epsilon\psi] + B\nabla^4 \psi
\end{aligned} \tag{2.46}$$

パラメータが煩わしいため、これも無次元化しよう。計算方法が先ほどと全く同じであるため、途中計算は省略するが、無次元化すると、

$$\frac{\partial \psi}{\partial t} = \nabla^2 [\psi^3 - \psi] + \nabla^4 \psi \tag{2.47}$$

この形の方程式を Cahn-Hilliard 方程式という。これもシミュレーションを行えばわかるように、保存系の相分離をうまく表現する。

以上より、Ginzburg-Landau 型の自由エネルギー汎関数を最小化するように方程式を組み立てると、どうやら 2 相間の相分離がうまく記述できるようだ。あとはこの方程式がどのように不安定から安定へと進むのかを数式的 (非線形物理的) に解析すれば、スピノーダル分解について理解が進みそうである。非常に感覚的である。

とはいうものの、Ginzburg-Landau 理論において、不安定点や準安定点が物理的な意味を持つのかは、よくわからない。ゆえに、その議論をどれほど信じてよいのかも、よくわからない[*51]。

2.5.4 dewetting の方程式の謎

支配方程式

本編で、流体力学的に得られる dewetting の方程式として次の式を紹介した。

$$\frac{\partial h}{\partial t} = \nabla \cdot [\frac{h^2}{3}(3\beta + h)\nabla(p + \phi)] \tag{2.48}$$

これは体積についての保存則になっている。さて、エネルギー汎関数は、いったいどのような形をしているのだろうか？先ほどまでの話に登場した Ginzburg-Landau 型のエネルギー汎関数は、

[*51] この点に関しても、なんだかモヤっとする。多少のわからないことを受け入れたら、相分離の特徴をうまく記述することができた。しかし、当てずっぽうというわけではない。これが、すべてがわかっていないと何もわからない数学との違いなのかもしれない。が、やはりモヤっとする。難しい。

2 相間の相分離に着目して得た自由エネルギーであった。dewetting はいわば、3 相間の相分離である。Gizburg-Landau 型であるという保証はどこにもない。かといって、この式と睨めっこしていても、やはりなにもわからない。

ならば少し方針を変えて、先に自由エネルギーを導出し、そこから (2.48) がどのような形になっているのか考えてみる。

長距離力が働く液膜の自由エネルギー

液体-気体間の界面張力 (表面張力) を γ、固体-液体間の界面張力を γ_{sl}、固体-気体間の界面張力を γ_{sv} とおく。すると、基板上にある液膜の自由エネルギーは[*52]、

$$F = \int d\boldsymbol{r}[\gamma_{sl} - \gamma_{sv} + \gamma\{1 + \frac{1}{2}(h_x^2 + h_y^2)\}] \tag{2.49}$$

さらに、基盤と液体の間に長距離力による相互作用がある場合、その相互作用によるエネルギー $V[h]$ も加わって、

$$F = \int d\boldsymbol{r}[\gamma_{sl} - \gamma_{sv} + \gamma\{1 + \frac{1}{2}(h_x^2 + h_y^2)\} + V[h]] \tag{2.50}$$

となる。$S := -(\gamma + \gamma_{sl} - \gamma_{sv})$ とすると、

$$F = \int d\boldsymbol{r}[-S + \frac{\gamma}{2}(h_x^2 + h_y^2) + V[h]] \tag{2.51}$$

もう一度、方程式と睨めっこ

今考えたい問題は、保存系において自由エネルギー (2.51) を最小化する動的過程である。したがって、体積の保存則から出発することにしよう。Cahn-Hilliard 方程式の時と同じようにして、

$$\frac{\partial h}{\partial t} =^{?} \nabla \cdot (\nabla \frac{\delta F}{\delta h}) \tag{2.52}$$

となるのではないかと予想される。ところが、

$$\nabla \cdot (\nabla \frac{\delta F}{\delta h}) = \nabla \cdot [\nabla(\gamma \nabla^2 h + \frac{\partial V}{\partial h})] =: \nabla \cdot [\nabla(p + \phi)] \tag{2.53}$$

となる。すなわち、流体の流れを表す mobility の項がないため、(2.52) は間違いである。したがって、dewetting の方程式は、

$$\frac{\partial h}{\partial t} = \nabla \cdot [\frac{h^2}{3}(3\beta + h)\nabla(\frac{\delta F}{\delta h})] \tag{2.54}$$

というように書かれることと予想される。割とコレは妥当である。3.5 節で述べた通り、保存則によれば、右辺 [...] の中身が $-\boldsymbol{J}$ になっているから、

$$\boldsymbol{J} = [\frac{h^2}{3}(3\beta + h)] \cdot [-\nabla(\frac{\delta F}{\delta h})] \tag{2.55}$$

[*52] 液体-気体界面はまっすぐではなく曲がっている。界面張力 (表面張力) は単位面積当たりのエネルギーなのであるから、まがった面を計算してあげる必要がある。急激に曲がり具合が変化していない場合は、その曲面の面積は $\int dx \int dy \sqrt{1 + h_x^2 + h_y^2} =: \int d\boldsymbol{r} \sqrt{1 + h_x^2 + h_y^2}$ で書ける。ゆえに $F = \int d\boldsymbol{r}[\gamma_{sl} - \gamma_{sv} + \gamma\sqrt{1 + h_x^2 + h_y^2}]$ である。ここで、接触角が十分小さいとすると、$\sqrt{1 + h_x^2 + h_y^2} \approx 1 + \frac{1}{2}(h_x^2 + h_y^2)$ と書ける。

これは、Cahn-Hilliard 方程式の流束に、mobility 項 $\frac{h^2}{3}(3\beta + h)$ を掛け合わせたものになっている。mobility 項は言ってしまえばただの数だから、流束の方向が自由エネルギーを小さくする方向を向いていることがわかる。ゆえに、この dewetting の方程式も、自由エネルギーを最小化する方向に動くということがわかる。

（※文責: 友杉悠愛）

第 3 章　カルマン渦のシミュレーション

3.1　背景

数値流体力学は、気象学、航空工学、細胞生物学などさまざまな学問分野において関心の高いテーマである。流体運動の支配方程式であるナビエ・ストークス方程式は、非線形偏微分方程式であり、これを用いた流体のシミュレーションには工夫を必要とする。ナビエ・ストークス方程式に限らず、微分方程式の数値計算法として基本的な方法は、微分方程式を離散化して得られる差分方程式を用いて逐次計算を行い、その時間発展を観察する差分法である。その一方で、格子ボルツマン法（Lattice Boltzmann Method; LBM）では、ナビエ・ストークス方程式を用いるのではなく、気体分子運動論の分野から生まれたボルツマン方程式を、流体運動のシミュレーションに応用している。実際、ボルツマン方程式からナビエ・ストークス方程式を導くことができることが知られている [7]。

LBM の強みとしては、高速計算が可能であることと、障害物の取り扱いが容易であることが挙げられる [7]。本グループでは、LBM を用いたシミュレーションの題材として、カルマン渦（下図参照）を選んだ。カルマン渦を観察するためには、柱状の障害物を設定することが必要である。この場合、障害物の形状はさほど複雑ではないため、差分法を用いてもシミュレーションは十分可能であると考えられるが、ここでは障害物の取り扱いに強みをもつ LBM を利用することとした。

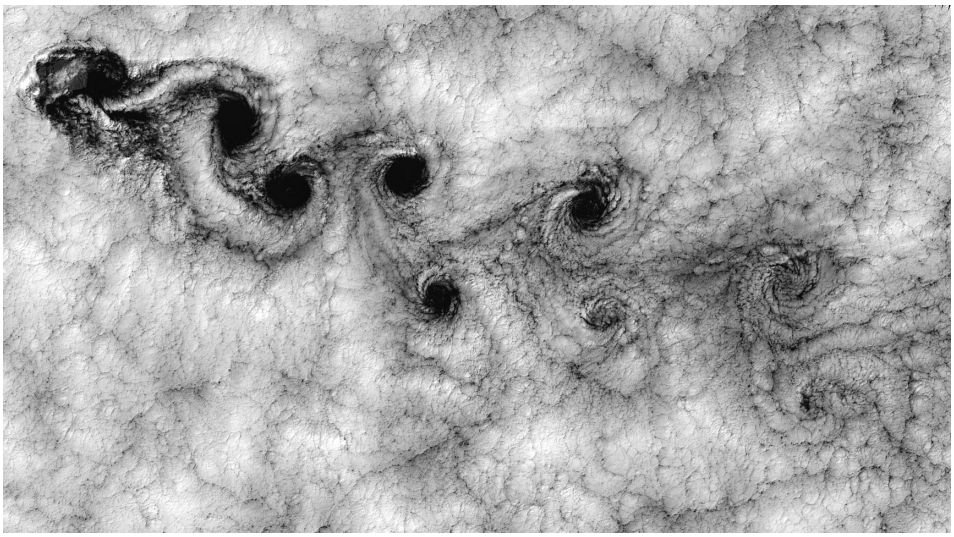


図 3.1　島を障害物として発生した雲のカルマン渦 [8]

3.2　手法

このセクションでは、格子ボルツマン法（Lattice Boltzmann Method; LBM）について説明する。本グループでは、古典コンピュータにおいてカルマン渦のシミュレーションを行うために、LBM を用いた。なお、説明はあくまで技術的側面にとどめ、数学的・物理学的理論に関しては最小限必要なもののみ説明する。

3.2.1 格子ガスオートマトン

LBM の説明の前に、その基礎となった理論である格子ガスオートマトン（Lattice Gas Automata; LGA）について説明する。LGA は、セルオートマトンを流体運動のシミュレーションに応用したものである。

LGA には主に、四角形の格子を用いる HPP モデルと、六角形の格子を用いる FHP モデルとがある。ここでは、流体の運動をより正確に再現することができる FHP モデル [7] について説明する。

FHP モデルでは、正六角形の格子上を仮想的な粒子が運動し、衝突と並進を繰り返すことによってシミュレーションを行う。この際に重要となるのが、同じ格子点上に複数の仮想粒子が存在するとき、粒子どうしの衝突をどのように計算するか、ということである。

ここで、格子点から移動することができる 6 つの方向に対して、1 から 6 までの番号を振る。また、格子点上に、1 から 6 の方向の速度をもつ仮想粒子が存在するかどうかを、それぞれ 0 か 1 かで表そう。すなわち、例えば $(0,0,1,0,1,0)$ と書けば、3 の方向の速度をもつ粒子と、5 の方向の速度をもつ粒子とが格子点上に存在していることになる。

格子点上に存在する粒子が、ある状態であるとき、衝突を計算するための規則は以下のようになる [9]。

二つの粒子が正面衝突するときは、速度を 60° または 120° 回転させる。この際、どちらの回転を採用するかはランダムに決定される。

例）$(1,0,0,1,0,0) \to (0,1,0,0,1,0)$ または $(0,0,1,0,0,1)$

三つの粒子が等しい角度で衝突するときは、速度を 60° 回転させる。

例）$(1,0,1,0,1,0) \to (0,1,0,1,0,1)$

二つの粒子 a,b が正面衝突するのに加えて、もうひとつの粒子 c が衝突するときは、c の速度はそのままにして、a,b の速度を c および c を 180° 回転させた速度と異なる方向に回転させる。

例）$(1,1,0,1,0,0) \to (0,1,1,0,0,1)$

正面衝突する二つの粒子が二組あるときは、速度を 60° または 120° 回転させる。この際、どちらの回転を採用するかはランダムに決定される。

例）$(1,1,0,1,1,0) \to (0,1,1,0,1,1)$ または $(1,0,1,1,0,1)$

状態がランダムに決定されるときは、どちらの状態も等しい確率（2 分の 1）で選択される。なお、上記以外の衝突のパターンでは、方向の変化を計算する必要はない。

上に説明した規則による衝突ステップと、各粒子をその速度の方向に並進移動させる並進ステップを繰り返すことによって、FHP モデルによるシミュレーションは実現される。衝突ステップを Ω_k と表すと、時間ステップ Δt の間に格子距離 $\Delta x = \mathbf{c}_k \Delta t$ だけ移動する粒子の運動方程式は、0 か 1 かのどちらかをとる変数 n_k によって

$$n_k(\mathbf{x} + \mathbf{c}_k \Delta t, t + \Delta t) = n_k(\mathbf{x}, t) + \Omega_k(\mathbf{x}, t)$$

と表される [7]。ただし $k = 1, 2, \cdots, 6$ である。

FHP モデルによるシミュレーションを行う Python のプログラムを付録に掲載した。

3.2.2 格子ボルツマン法

ここから、LBM について説明する。FHP モデルでは、格子上の粒子の状態を 0 か 1 かで表したのに対し、LBM では分布関数 f_k を用いて粒子の存在割合を表す。LBM にも複数のモデルが存在するが、本グループでは 2 次元に 9 方向の速度をもつモデル（D2Q9 モデル）をもっぱら用いた。

LBM の運動方程式は以下のように表される [7]。

$$f_k(\mathbf{x} + \mathbf{c}_k \Delta t, t + \Delta t) = f_k(\mathbf{x}, t) - \frac{1}{\tau}(f_k(\mathbf{x}, t) - f_k^{eq}(\mathbf{x}, t)) \tag{3.1}$$

ここで、τ は緩和時間で、流体の粘性に比例する。すなわち、τ が大きくなるほど流体の粘性は増し、ゆっくりと流れる。

また、f_k^{eq} は平衡分布関数で、マクスウェル-ボルツマン分布をテイラー展開して以下のように得られる [7]。

$$f_k^{eq} = \omega_k \rho \left(1 + \frac{3\mathbf{c}_k \cdot \mathbf{u}}{c^2} + \frac{9(\mathbf{c}_k \cdot \mathbf{u})^2}{2c^4} - \frac{3\mathbf{u} \cdot \mathbf{u}}{2c^2} \right) \tag{3.2}$$

ここで、ρ は密度、c は代表離散速度、$\mathbf{u}$ は流速である。ρ, $\mathbf{u}$ は、シミュレーションの各ステップにおいて、そのつど値を計算する数である。また、c はシミュレーションにあたって自由に設定するパラメータで、一般に $c = \Delta x = \Delta t = 1$ と設定する。

離散速度 $\mathbf{c}_k$ について説明しよう。先述のとおり 9 つの方向があるから、k として $k = 0, 1, 2, \cdots, 8$ をとって、以下のように定義する。

$$\mathbf{c}_0 = (0, 0),\ \mathbf{c}_1 = (c, 0),\ \mathbf{c}_2 = (0, c),\ \mathbf{c}_3 = (-c, 0),\ \mathbf{c}_4 = (0, -c),$$
$$\mathbf{c}_5 = (c, c),\ \mathbf{c}_6 = (-c, c),\ \mathbf{c}_7 = (-c, -c),\ \mathbf{c}_8 = (c, -c)$$

これらの離散速度は、下図のように、ある格子点から隣接する格子点へと向かう方向の速度を表す。なお、その場にとどまる速度 $\mathbf{c}_0$ は、計算を安定させるために用いている。

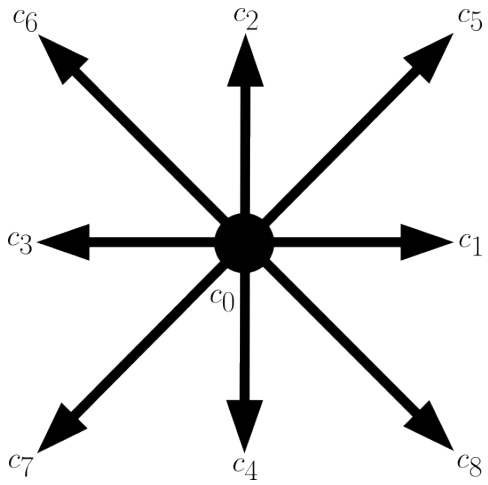


図 3.2　9 方向の離散速度

また、重み係数 ω_k についても説明しよう。これは各方向に対して以下のように与えられる [7]。

$$\begin{aligned}\omega_0 &= \frac{4}{9},\\ \omega_1 = \cdots = \omega_4 &= \frac{1}{9},\\ \omega_5 = \cdots = \omega_8 &= \frac{1}{36}\end{aligned}$$

重み係数の和が $\sum_k \omega_k = 1$ であることに注意されたい。

以上のことを踏まえて、簡単な計算により、各 $k = 0, 1, 2, \cdots, 8$ に対して平衡分布関数は以下のようになることがわかる。ただし、代表離散速度を $c = 1$ と設定し、流速を $\mathbf{u} = (u_x, u_y)$ とする。実際のプログラムの中では、以下の各式を用いることになる。

$$\begin{aligned}
f_0^{eq} &= \frac{4}{9}\rho\left(1 - \frac{3}{2}\mathbf{u}^2\right)\\
f_1^{eq} &= \frac{1}{9}\rho\left(1 + 3u_x + \frac{9}{2}u_x^2 - \frac{3}{2}\mathbf{u}^2\right)\\
f_2^{eq} &= \frac{1}{9}\rho\left(1 + 3u_y + \frac{9}{2}u_y^2 - \frac{3}{2}\mathbf{u}^2\right)\\
f_3^{eq} &= \frac{1}{9}\rho\left(1 - 3u_x + \frac{9}{2}u_x^2 - \frac{3}{2}\mathbf{u}^2\right)\\
f_4^{eq} &= \frac{1}{9}\rho\left(1 - 3u_x + \frac{9}{2}u_x^2 - \frac{3}{2}\mathbf{u}^2\right)\\
f_5^{eq} &= \frac{1}{36}\rho\left(1 + 3(u_x + u_y) + \frac{9}{2}(u_x + u_y)^2 - \frac{3}{2}\mathbf{u}^2\right)\\
f_6^{eq} &= \frac{1}{36}\rho\left(1 + 3(-u_x + u_y) + \frac{9}{2}(-u_x + u_y)^2 - \frac{3}{2}\mathbf{u}^2\right)\\
f_7^{eq} &= \frac{1}{36}\rho\left(1 - 3(u_x + u_y) + \frac{9}{2}(u_x + u_y)^2 - \frac{3}{2}\mathbf{u}^2\right)\\
f_8^{eq} &= \frac{1}{36}\rho\left(1 + 3(u_x - u_y) + \frac{9}{2}(u_x - u_y)^2 - \frac{3}{2}\mathbf{u}^2\right)
\end{aligned} \tag{3.3}$$

ここで、密度は $\rho = \sum_k f_k$ と計算できる [7]。また、x 方向と y 方向の速度はそれぞれ

$$\begin{aligned}u_x &= \frac{f_1 + f_5 + f_8 - f_3 - f_6 - f_7}{\rho}\\ u_y &= \frac{f_2 + f_5 + f_6 - f_4 - f_7 - f_8}{\rho}\end{aligned}$$

と計算できる。対応する $\mathbf{c}_k$ の x 成分が 0 でないものを u_x の計算に用い、y 成分が 0 でないものを u_y の計算に用いていることに注意いただきたい。

ここまで、LBM の運動方程式 (3.1) と、その中に用いられる平衡分布関数 (3.2) について説明した。ここからは、実際に LBM によるシミュレーションを実施するためのアルゴリズムを解説する。

まずはアルゴリズムの全体像を以下に示す。3 つのステップ「並進ステップ・跳ね返りステップ・衝突ステップ」を繰り返すことが、アルゴリズムの基本である。

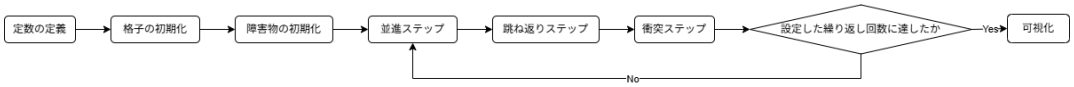


図 3.3　LBM のアルゴリズム

上に挙げたアルゴリズムの各ステップについて、順を追って見ていこう。

＜定数の定義＞

緩和時間 τ と動粘性係数 ν の間には、$\nu = \dfrac{1}{6}(2\tau - 1)c\Delta t$ の関係があるから、$c = \Delta t = 1$ であることに注意して

$$\omega = \frac{1}{\tau} = \frac{1}{3\nu + 0.5}$$

とする。ω は緩和時間の逆数である。動粘性係数は $\nu = 0.02$ 程度の値を与えるとよい。なお、動粘性係数は正の値をとらなければならないから、$\nu > 0$ として $\tau > 0.5$ である必要がある。

余談だが、流れの乱れやすさを表すレイノルズ数 Re は、粘性力と慣性力の比として

$$Re = \frac{U_0 D}{\nu}$$

と表される [10]。U_0 は一様流速であり、その値としては初期流速 u_0 を用いるとよい。また、D は代表長さであり、これは障害物である円柱の直径を用いるとよい。このとき、$\tau \to 0.5 + 0$ とすると、緩和時間 τ と動粘性係数 ν の関係式から、$\nu \to +0$ となる。よって、このとき $Re \to +\infty$ となるから、動粘性係数 ν を 0 に近い値にとるほど、シミュレーションは不安定になる。

そのほか、初期流速は $u_0 = 0.1$ 程度の値を与えるとよい。

＜格子の初期化＞

前提として、各方向の速度を保持するために、9 つの格子（2 次元配列）を用意する。シミュレーションでは、各格子に対してそれぞれアルゴリズムを適用して、計算を進める。

これらの格子の初期化は、各方向に対する平衡分布関数の式 (3.3) によって行う。この際、$\rho = 1$、$u_x = u_y = u_0$ とする。

＜障害物の初期化＞

カルマン渦などの興味深い現象を観察するためには、障害物の設定が必要となる。障害物の設定には、まず障害物の位置を 2 次元配列として設定し（たとえば、障害物の位置を boolean 値で 1 として設定する）、さらに障害物の周囲を同様に 2 次元配列として設定する必要がある。

障害物の周囲は、障害物の上下左右および斜め方向の周囲であることを記憶するために、8 つの 2 次元配列を用意する。以下の図を参照してほしい。黒色の格子が障害物の位置であるとすると、たとえば左図の赤色の格子が右方向の周囲であり、右図の青色の格子が左下方向の周囲である。

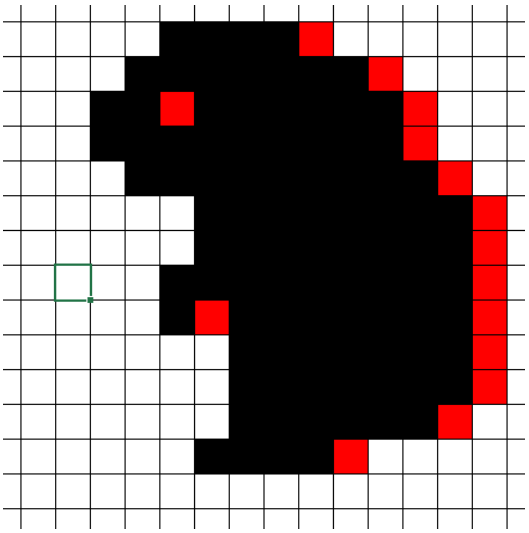


図 3.4　障害物の右方向の周囲

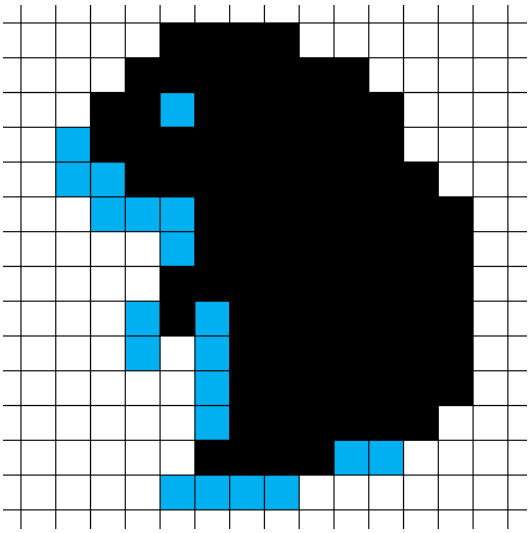


図 3.5　障害物の左下方向の周囲

＜並進ステップ＞
格子の初期化で準備した 9 つの格子を、それぞれの速度の方向に一つずつずらす操作である。たとえば、右向きの速度を保持している格子に対しては、すべての要素を右に一つずつずらす操作を行う。

＜跳ね返りステップ＞
障害物の位置とその周囲を保持している配列を利用して、障害物の表面で粒子を跳ね返す。たとえば、右向きの速度をもった粒子が、障害物の左側面に衝突したときは、左向きの速度で跳ね返る。プログラム上の操作としては、左向きの速度を右向きの速度によって上書きすればよい。

＜衝突ステップ＞
各格子点上で、(3.3) によって粒子の衝突を計算する。

＜可視化＞
お好みの方法で可視化を行う。

以上が、格子ボルツマン法で流体の運動をシミュレーションするための具体的なアルゴリズムである。シミュレーションを行う Python のプログラムを付録に掲載した。なお、このプログラムは『Fluid Dynamics Simulation[11]』を改変したものである。

3.3　結果

格子ガスオートマトン（FHP モデル）では、以下のような様子が確認できた。はじめ左下に密集していた粒子が、徐々に拡散していく様子が見て取れる。

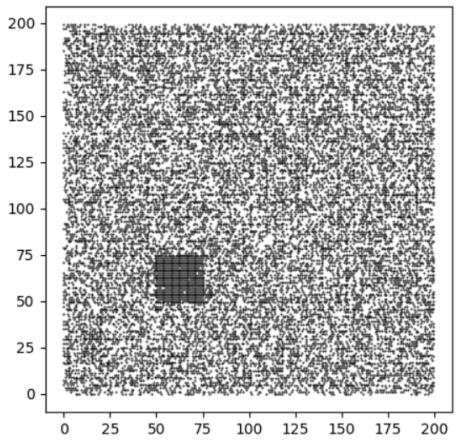


図 3.6　はじめの状態

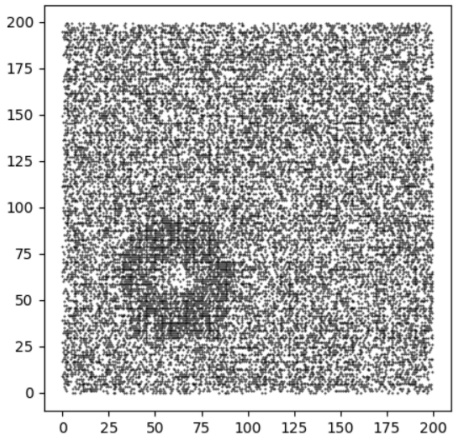


図 3.7　拡散が始まった様子

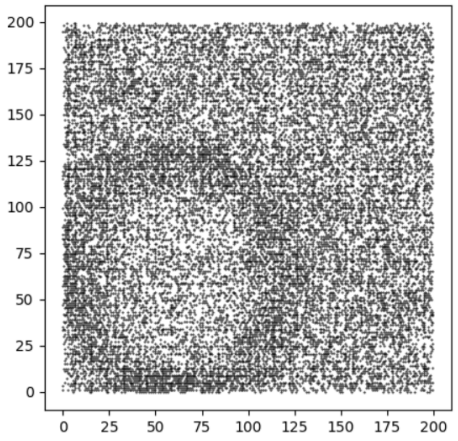


図 3.8　拡散が進んでいく様子

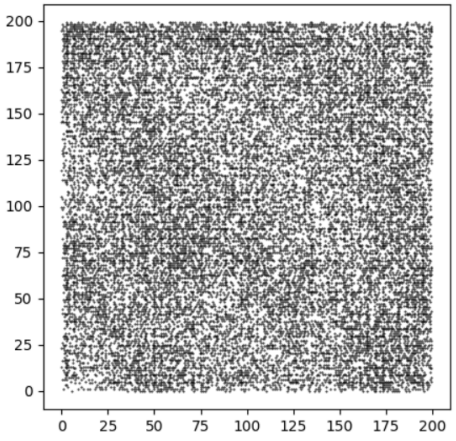


図 3.9　十分に時間が経過したあとの様子

以下の画像は、格子ボルツマン法を用いたシミュレーションの結果である。レイノルズ数が $Re = 67$、$Re = 178$、$Re = 267$、$Re = 533$ の 4 つの場合についてシミュレーションを行った。ここで表示しているのは渦度である。渦度とは、速度ベクトルの回転 $\nabla \times \mathbf{u}$ で、流体が渦を巻く度合いを表す量である。

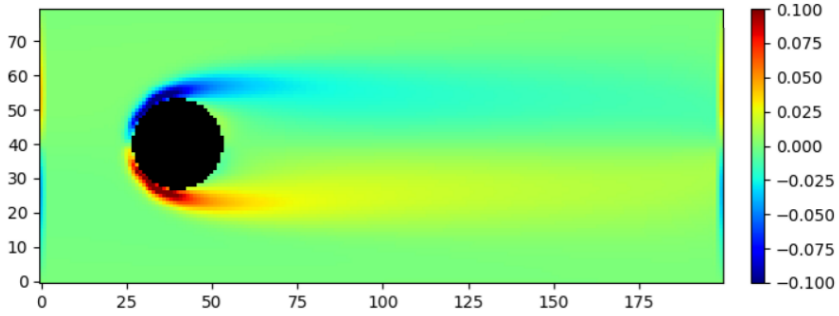


図 3.10　Re=67 の場合

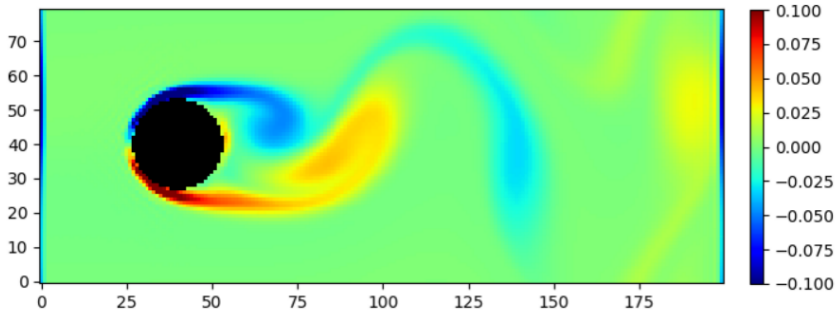


図 3.11　Re=178 の場合

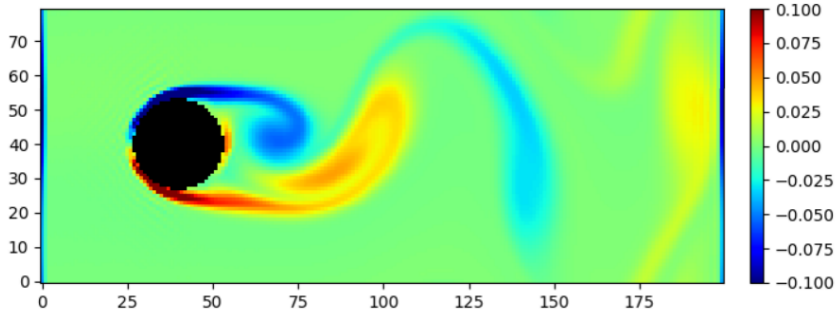


図 3.12　Re=267 の場合

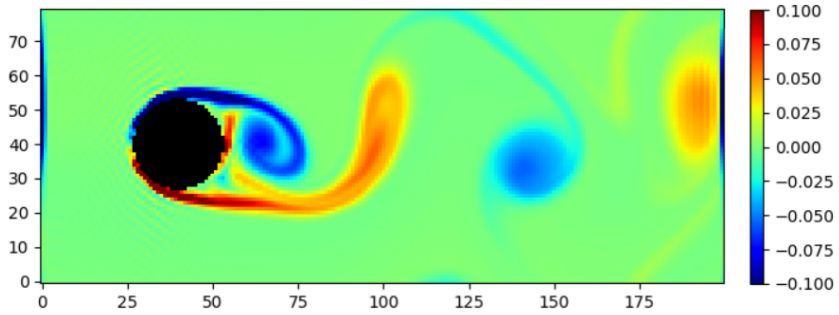


図 3.13　Re=533 の場合

3.4　考察

LBM を用いることで、目標としていたカルマン渦のシミュレーションを行うことができた。レイノルズ数の変化に着目すると、レイノルズ数が低い場合（$Re = 67$）には、渦が観察されなかった。その一方で、レイノルズ数を大きくすると、カルマン渦を観察することができた。さらに、レイノルズ数が大きくなるほど、渦が激しくなる様子が見られた。実際、$Re = 178$ の場合と $Re = 533$ の場合とを見比べると、右端に存在する渦の渦度が、$Re = 533$ の場合のほうがより大きいことがわかる。以上のことから、シミュレーション結果は、レイノルズ数の変化とも整合的であることがわかる。

（※文責: 山崎大紀）

3.5 付録

3.5.1 FHP モデルによるシミュレーションのプログラム

```
1
2  %%time
3
4  %matplotlib nbagg
5
6  import numpy as np
7  import matplotlib
8  import matplotlib.pyplot as plt
9  from matplotlib import animation, rc
10 from matplotlib.animation import ArtistAnimation
11 import random
12 import math
13
14 rc('animation', html='jshtml')
15
16 # 格子の次元
17 width = 200
18 height = 200
19
20 # シミュレーションの繰り返し回数
21 N = 1000
22
23 nNE = np.zeros((height, width))
24 nE = np.zeros((height, width))
25 nSE = np.zeros((height, width))
26 nSW = np.zeros((height, width))
27 nW = np.zeros((height, width))
28 nNW = np.zeros((height, width))
29
30 # 粒子をランダムに初期化する
31 for j in range(width):
32     for i in range(height):
33         nNE[i,j] = int(random.random()*1.1)
34         nE[i,j] = int(random.random()*1.1)
35         nSE[i,j] = int(random.random()*1.1)
36         nSW[i,j] = int(random.random()*1.1)
37         nW[i,j] = int(random.random()*1.1)
```

```
38          nNW[i,j] = int(random.random()*1.1)
39
40  # 粒子の密集した部分を設定する
41  for j in range(50,75):
42      for i in range(50,75):
43          nNE[i,j] = int(random.random()*2)
44          nE[i,j] = int(random.random()*2)
45          nSE[i,j] = int(random.random()*2)
46          nSW[i,j] = int(random.random()*2)
47          nW[i,j] = int(random.random()*2)
48          nNW[i,j] = int(random.random()*2)
49
50  # 粒子が存在する座標を記録する配列
51  particle_ax = []
52  particle_ay = []
53  particle_bx = []
54  particle_by = []
55
56  # 粒子を並進移動する
57  # 六角形格子を使用しているため、奇数行と偶数行では取り扱いが異なることに注意
58  def stream():
59      global nNE, nE, nSE, nSW, nW, nNW
60
61      nNE_save = np.copy(nNE)
62      nSE_save = np.copy(nSE)
63      nSW_save = np.copy(nSW)
64      nNW_save = np.copy(nNW)
65
66      nNE[2::2,:] = nNE_save[1:-1:2,:]
67      nNE[1::2,:] = np.roll(nNE_save[::2,:], 1, axis=1)
68      nE = np.roll(nE, 1, axis=1)
69      nSE[::2] = nSE_save[1::2,:]
70      nSE[1:-1:2,:] = np.roll(nSE_save[2::2], 1, axis=1)
71      nSW[::2,:] = np.roll(nSW_save[1::2,:], -1, axis=1)
72      nSW[1:-1:2,:] = nSW_save[2::2,:]
73      nW = np.roll(nW, -1, axis=1)
74      nNW[2::2,:] = np.roll(nNW_save[1:-1:2,:], -1, axis=1)
75      nNW[1::2,:] = nNW_save[::2,:]
76
77  # 上下左右の端における粒子の跳ね返り
78  def bounce():
79      global nNE, nE, nSE, nSW, nW, nNW
```

```
80
81      nSE[height-1,:] = nNE[height-1,:]
82      nNE[height-1,:] = 0
83      nNE[::2,0] = nNW[::2,0]
84      nNW[::2,0] = 0
85      nE[:,0] = nW[:,0]
86      nW[:,0] = 0
87      nNE[0,:] = nSE[0,:]
88      nSE[0,:] = 0
89      nSE[::2,0] = nSW[::2,0]
90      nSW[::2,0] = 0
91      nNW[0,:] = nSW[0,:]
92      nSW[0,:] = 0
93      nSW[1::2,width-1] = nSE[1::2,width-1]
94      nSE[1::2,width-1] = 0
95      nW[:,width-1] = nE[:,width-1]
96      nE[:,width-1] = 0
97      nSW[height-1,:] = nNW[height-1,:]
98      nNW[height-1,:] = 0
99      nNW[1::2,width-1] = nNE[1::2,width-1]
100     nNE[1::2,width-1] = 0
101
102 # 粒子どうしの衝突
103 def collide():
104     global nNE, nE, nSE, nSW, nW, nNW
105
106     for j in range(width):
107         for i in range(height):
108             # 格子点上に存在する粒子の速度の方向を、タプルに保存する
109             tupple = (nNE[i,j], nE[i,j], nSE[i,j], nSW[i,j], nW[i,j
                    ], nNW[i,j])
110             if tupple == (1,0,0,1,0,0):
111                 nNE[i,j] = nSW[i,j] = 0
112                 rd = random.random()
113                 if rd < 0.5:
114                     nE[i,j] = nW[i,j] = 1
115                 else:
116                     nSE[i,j] = nNW[i,j] = 1
117             elif tupple == (0,1,0,0,1,0):
118                 nE[i,j] = nW[i,j] = 0
119                 rd = random.random()
120                 if rd < 0.5:
```

```
121                     nNE[i,j] = nSW[i,j] = 1
122                 else:
123                     nSE[i,j] = nNW[i,j] = 1
124             elif tupple == (0,0,1,0,0,1):
125                 nSE[i,j] = nNW[i,j] = 0
126                 rd = random.random()
127                 if rd < 0.5:
128                     nNE[i,j] = nSW[i,j] = 1
129                 else:
130                     nE[i,j] = nW[i,j] = 1
131             elif tupple == (1,0,1,0,1,0):
132                 nNE[i,j] = nSE[i,j] = nW[i,j] = 0
133                 nE[i,j] = nSW[i,j] = nNW[i,j] = 1
134             elif tupple == (0,1,0,1,0,1):
135                 nE[i,j] = nSW[i,j] = nNW[i,j] = 0
136                 nNE[i,j] = nSE[i,j] = nW[i,j] = 1
137             elif tupple == (1,1,0,1,0,0) or tupple == (1,0,0,1,1,0)
                    :
138                 nNE[i,j] = nSW[i,j] = 0
139                 nSE[i,j] = nNW[i,j] = 1
140             elif tupple == (1,0,1,1,0,0) or tupple == (1,0,0,1,0,1)
                    :
141                 nNE[i,j] = nSW[i,j] = 0
142                 nE[i,j] = nW[i,j] = 1
143             elif tupple == (1,1,0,0,1,0) or tupple == (0,1,0,1,1,0)
                    :
144                 nE[i,j] = nW[i,j] = 0
145                 nSE[i,j] = nNW[i,j] = 1
146             elif tupple == (0,1,1,0,1,0) or tupple == (0,1,0,0,1,1)
                    :
147                 nE[i,j] = nW[i,j] = 0
148                 nNE[i,j] = nSW[i,j] = 1
149             elif tupple == (1,0,1,0,0,1) or tupple == (0,0,1,1,0,1)
                    :
150                 nSE[i,j] = nNW[i,j] = 0
151                 nE[i,j] = nW[i,j] = 1
152             elif tupple == (0,1,1,0,0,1) or tupple == (0,0,1,0,1,1)
                    :
153                 nSE[i,j] = nNW[i,j] = 0
154                 nNE[i,j] = nSW[i,j] = 1
155             elif tupple == (1,1,0,1,1,0):
156                 nSE[i,j] = nNW[i,j] = 1
```

```
157                     rd = random.random()
158                     if rd < 0.5:
159                         nNE[i,j] = nSW[i,j] = 0
160                     else:
161                         nE[i,j] = nW[i,j] = 0
162                 elif tupple == (0,1,1,0,1,1):
163                     nNE[i,j] = nSW[i,j] = 1
164                     rd = random.random()
165                     if rd < 0.5:
166                         nE[i,j] = nW[i,j] = 0
167                     else:
168                         nSE[i,j] = nNW[i,j] = 0
169                 elif tupple == (1,0,1,1,0,1):
170                     nE[i,j] = nW[i,j] = 1
171                     rd = random.random()
172                     if rd < 0.5:
173                         nNE[i,j] = nSW[i,j] = 0
174                     else:
175                         nSE[i,j] = nNW[i,j] = 0
176
177 # 粒子が存在する格子点を記録する
178 def setParticle():
179     global particle_ax, particle_ay, particle_bx, particle_by
180
181     particle_ax.clear()
182     particle_ay.clear()
183     particle_bx.clear()
184     particle_by.clear()
185
186     for j in range(width):
187         for i in range(0, height, 2):
188             sum = nNE[i,j] + nE[i,j] + nSE[i,j] + nSW[i,j] + nW[i,j
                    ] + nNW[i,j]
189             if sum > 0:
190                 particle_ax.append(j+0.5)
191                 particle_ay.append(i)
192         for i in range(1, height, 2):
193             sum = nNE[i,j] + nE[i,j] + nSE[i,j] + nSW[i,j] + nW[i,j
                    ] + nNW[i,j]
194             if sum > 0:
195                 particle_bx.append(j)
196                 particle_by.append(i)
```

```
197
198
199 fig , ax = plt.subplots(figsize=(5,5))
200 artists = []
201
202 # シミュレーション
203 for n in range(N):
204     stream()
205     bounce()
206     collide()
207     setParticle()
208     if n % 1 == 0:
209         artist_a = ax.scatter(particle_ax , particle_ay , s=0.2, c='k
            ')
210         artist_b = ax.scatter(particle_bx , particle_by , s=0.2, c='k
            ')
211         artists.append([artist_a , artist_b])
212
213 anim = ArtistAnimation(fig , artists , interval=50)
214 anim
215 #anim.save("FHP.mp4")
```

3.5.2 LBM によるシミュレーションのプログラム

```
1
2 %%time
3
4 # アニメーション
5 %matplotlib nbagg
6
7 import sys
8 import time
9 import numpy as np
10 import matplotlib
11 import matplotlib.pyplot as plt
12 from matplotlib import animation, rc
13 rc('animation', html='jshtml')
14
15
16 # 格子の次元
17 height = 80
18 width = 200
```

```
19
20 # 粘性
21 viscosity = 0.04
22
23 # 単一緩和時間
24 tau = 3*viscosity + 0.5
25 omega = 1 / tau
26
27 # 初期流速
28 u0 = 0.1
29
30 # 障害物の形( 1: 直線, 2: 円)
31 barrier_type = 2
32
33 # 平衡分布関数の重み係数 w_i
34 four9ths = 4.0/9.0    # i = 0
35 one9th   = 1.0/9.0    # i = 1,2,3,4
36 one36th  = 1.0/36.0   # i = 5,6,7,8
37
38 # 平衡分布関数の式に従って初期化
39 n0 = four9ths * (np.ones((height,width)) - 1.5*u0**2)
40 nN = one9th * (np.ones((height,width)) - 1.5*u0**2)
41 nS = one9th * (np.ones((height,width)) - 1.5*u0**2)
42 nE = one9th * (np.ones((height,width)) + 3*u0 + 4.5*u0**2 - 1.5*u0
       **2)
43 nW = one9th * (np.ones((height,width)) - 3*u0 + 4.5*u0**2 - 1.5*u0
       **2)
44 nNE = one36th * (np.ones((height,width)) + 3*u0 + 4.5*u0**2 - 1.5*
       u0**2)
45 nSE = one36th * (np.ones((height,width)) + 3*u0 + 4.5*u0**2 - 1.5*
       u0**2)
46 nNW = one36th * (np.ones((height,width)) - 3*u0 + 4.5*u0**2 - 1.5*
       u0**2)
47 nSW = one36th * (np.ones((height,width)) - 3*u0 + 4.5*u0**2 - 1.5*
       u0**2)
48
49 # 密度ρ
50 rho = n0 + nN + nS + nE + nW + nNE + nSE + nNW + nSW
51
52 # 流速の x 成分、 y 成分 u = (u_x, u_y)
53 ux = (nE + nNE + nSE - nW - nNW - nSW) / rho
54 uy = (nN + nNE + nNW - nS - nSE - nSW) / rho
```

```
55
56 # 障害物 ( true なら障害物あり)
57 barrier = np.zeros((height,width), bool)
58
59 # 障害物の位置 (横方向)
60 barrier_place = width / 5
61
62 # 障害物の大きさ (直線の長さ・円の半径)
63 barrier_size = height / (3 * barrier_type)
64
65 if barrier_type == 1:
66   barrier[height / 2 - barrier_size / 2 : height / 2 + barrier_size
       / 2, barrier_place] = True
67 if barrier_type == 2:
68   ii, jj = np.meshgrid(np.arange(width), np.arange(height))
69   d2 = np.square(ii - barrier_place) + np.square(jj - height/2)
70   barrier[d2<barrier_size*barrier_size] = True
71
72 # レイノルズ数
73 print("Re=", (barrier_size * 2 * u0) / ((1/6) * (2 * tau - 1)))
74
75 # 障害物の周囲
76 barrierN = np.roll(barrier,  1, axis=0)
77 barrierS = np.roll(barrier, -1, axis=0)
78 barrierE = np.roll(barrier,  1, axis=1)
79 barrierW = np.roll(barrier, -1, axis=1)
80 barrierNE = np.roll(barrierN,  1, axis=1)
81 barrierNW = np.roll(barrierN, -1, axis=1)
82 barrierSE = np.roll(barrierS,  1, axis=1)
83 barrierSW = np.roll(barrierS, -1, axis=1)
84
85 def stream():
86     global nN, nS, nE, nW, nNE, nNW, nSE, nSW
87
88     # Streaming step (並進移動)
89     nN  = np.roll(nN,   1, axis=0)
90     nNE = np.roll(nNE,  1, axis=0)
91     nNW = np.roll(nNW,  1, axis=0)
92     nS  = np.roll(nS,  -1, axis=0)
93     nSE = np.roll(nSE, -1, axis=0)
94     nSW = np.roll(nSW, -1, axis=0)
95     nE  = np.roll(nE,   1, axis=1)
```

```
96      nNE = np.roll(nNE,   1, axis=1)
97      nSE = np.roll(nSE,   1, axis=1)
98      nW  = np.roll(nW,   -1, axis=1)
99      nNW = np.roll(nNW, -1, axis=1)
100     nSW = np.roll(nSW, -1, axis=1)
101
102     # Bounce step （障害物による跳ね返り）
103     nN[barrierN] = nS[barrier]
104     nS[barrierS] = nN[barrier]
105     nE[barrierE] = nW[barrier]
106     nW[barrierW] = nE[barrier]
107     nNE[barrierNE] = nSW[barrier]
108     nNW[barrierNW] = nSE[barrier]
109     nSE[barrierSE] = nNW[barrier]
110     nSW[barrierSW] = nNE[barrier]
111
112 def collide():
113     global rho, ux, uy, n0, nN, nS, nE, nW, nNE, nNW, nSE, nSW
114     rho = n0 + nN + nS + nE + nW + nNE + nSE + nNW + nSW
115     ux = (nE + nNE + nSE - nW - nNW - nSW) / rho
116     uy = (nN + nNE + nNW - nS - nSE - nSW) / rho
117     ux2 = ux * ux
118     uy2 = uy * uy
119     u2 = ux2 + uy2
120     omu215 = 1 - 1.5*u2
121     uxuy = ux * uy
122
123     n0 = (1-omega)*n0 + omega * four9ths * rho * omu215
124     nN = (1-omega)*nN + omega * one9th * rho * (omu215 + 3*uy +
            4.5*uy2)
125     nS = (1-omega)*nS + omega * one9th * rho * (omu215 - 3*uy +
            4.5*uy2)
126     nE = (1-omega)*nE + omega * one9th * rho * (omu215 + 3*ux +
            4.5*ux2)
127     nW = (1-omega)*nW + omega * one9th * rho * (omu215 - 3*ux +
            4.5*ux2)
128     nNE = (1-omega)*nNE + omega * one36th * rho * (omu215 + 3*(ux+
            uy) + 4.5*(u2+2*uxuy))
129     nNW = (1-omega)*nNW + omega * one36th * rho * (omu215 + 3*(-ux+
            uy) + 4.5*(u2-2*uxuy))
130     nSE = (1-omega)*nSE + omega * one36th * rho * (omu215 + 3*(ux-
            uy) + 4.5*(u2-2*uxuy))
```

```
131     nSW = (1-omega)*nSW + omega * one36th * rho * (omu215 + 3*(-ux-
            uy) + 4.5*(u2+2*uxuy))
132
133     nE[:,0] = one9th * (1 + 3*u0 + 4.5*u0**2 - 1.5*u0**2)
134     nW[:,0] = one9th * (1 - 3*u0 + 4.5*u0**2 - 1.5*u0**2)
135     nNE[:,0] = one36th * (1 + 3*u0 + 4.5*u0**2 - 1.5*u0**2)
136     nSE[:,0] = one36th * (1 + 3*u0 + 4.5*u0**2 - 1.5*u0**2)
137     nNW[:,0] = one36th * (1 - 3*u0 + 4.5*u0**2 - 1.5*u0**2)
138     nSW[:,0] = one36th * (1 - 3*u0 + 4.5*u0**2 - 1.5*u0**2)
139
140 def rot(ux, uy):
141     return np.roll(uy,-1,axis=1) - np.roll(uy,1,axis=1) - np.roll(
            ux,-1,axis=0) + np.roll(ux,1,axis=0)
142
143 # グラフの設定
144 theFig = plt.figure(figsize=(8,3))
145 fluidImage = plt.imshow(rot(ux, uy), origin='lower', norm=plt.
        Normalize(-.1,.1),
146                         cmap=plt.get_cmap('jet'), interpolation='
                            none')
147
148 bImageArray = np.zeros((height, width, 4), np.uint8)
149 bImageArray[barrier,3] = 255
150 barrierImage = plt.imshow(bImageArray, origin='lower',
        interpolation='none')
151
152 # シミュレーションを行う関数
153 startTime = time.perf_counter()
154 def nextFrame(arg):
155     global startTime
156     if (arg%100 == 0) and (arg > 0):
157       endTime = time.perf_counter()
158       print("%1.1f" % (100/(endTime-startTime)), 'frames per second
              ')
159       startTime = endTime
160     for step in range(50):
161         stream()
162         collide()
163     fluidImage.set_array(rot(ux, uy))
164     return (fluidImage, barrierImage)
165
166 # アニメーションの出力
```

```
167 print("Now generating animation. Please wait.")
168 animate = animation.FuncAnimation(theFig, nextFrame, interval=20,
        blit=True, frames=range(2048))
169 animate
```

第 4 章　量子コンピュータ

4.1　量子コンピュータ

4.1.1　背景

本プロジェクトでは，前セクションにて紹介された格子ボルツマン法 (Lattice Boltzmann Method, LBM)，薄膜の方程式に対して 2 つの量子アルゴリズムに基づく手法を適用した．具体的には，格子ボルツマン法の実装には量子ウォーク (Quantum Walk,QW) を，薄膜の方程式に対しては量子レザバーコンピューティング (Quantum Reservoir Computing,QRC) を用いた．

4.1.2　量子レザバーコンピューティング

量子コンピュータ上で偏微分方程式を扱う手法としては，ハミルトニアンシミュレーションや変分量子回路 (Variational Quantum Algorithm,VQA) を用いた機械学習アプローチが提案されている．しかし，このレビュー [12] の中でも言われている通り，本プロジェクトで対象とする薄膜の方程式のような非線形偏微分方程式を扱う場合に，量子力学の基本原理であるユニタリ性に整合しないことや，非線形項の実装に伴う計算コストの増大といった問題が発生する．そのため完全にシミュレーションするのは難しいというのが現状である．そこで，本プロジェクトでは偏微分方程式を直接シミュレーションするのではなく，量子回路そのものが非線形ダイナミクスを表現する能力を有しているのかを検証することに注目した．具体的には，QRC[13] の枠組みを使用する．QRC では，量子回路をリザバー層として利用する．本プロジェクトでは，量子回路がターゲットとなる非線形ダイナミクスをどの程度忠実に模倣できるかという点に焦点を当てて検証した．

4.1.3　量子ウォーク

一方，LBM に対してはアルゴリズムの構造の類似性に着目し量子ウォークを用いた．LBM の計算プロセスは，衝突ステップと並進ステップによって構成されている．ここで，QW におけるコイン演算子による内部状態の混合は LBM における衝突ステップに，シフト演算子による移動は並進ステップにそれぞれ対応して考えることができる．本プロジェクトでは，この構造の類似性を利用して，量子ウォークの時間発展そのものを流体とみなして利用するアルゴリズムを構築した．

4.2　量子レザバーコンピューティングによる非線形ダイナミクスの再現

本セクションでは，薄膜の方程式の非線形ダイナミクスを学習，再現するために構築した QRC の手法について記述する．

4.2.1 基本概念

リザバーコンピューティングは，リカレントニューラルネットワークの一種であり，中間層 (リザバー) の結合重みを固定したまま，入力信号を高次元空間へ写像し，出力の際に読み出し層の重みのみを最適化するというものである．[14]

量子リザバーコンピューティングは，このリザバー層として，量子回路のダイナミクスを用いるものである．本プロジェクトでは，量子回路が持つ複雑性と高次元性を利用して非線形偏微分方程式のダイナミクスを表現する能力があるのか評価する．

4.2.2 量子リザバーコンピューティング

量子回路が．古典的なニューラルネットワーク比較して強力なレザバーとして機能する背景には量子力学特有の性質によるものがある．

ヒルベルト空間の指数関数的増加

古典的なレザバーにおいて，ノードの数 N のネットワークは N 次元の状態空間しか持たない．一方，量子レザバーにおいては，n 個の量子ビットからなる系は 2^n 次元のヒルベルト空間 $\mathcal{H}$ によって記述される．量子系の状態 $|\psi\rangle$ は，基底状態 $|i\rangle (i = 0, \ldots, 2^n - 1)$ の線形結合として表される．

$$|\psi\rangle = \sum_{i=0}^{2^n-1} c_i |i\rangle , \quad \sum |c_i|^2 = 1 \tag{4.1}$$

ここで，c_i は複素確率振幅である．量子ビット数 n に対して状態空間の次元は指数関数的に増大するため，わずか数十量子ビットであっても，古典コンピュータでは扱いきれないほどの高次元特徴空間へ入力データを写像することが可能となる．

角度エンコーディングによる非線形写像

線形なシュレーディンガー方程式に従う量子系において，入力データに対する非線形性はエンコーディングのプロセスによって生まれる．本手法では，入力データ $\mathbf{u}(t) = [u_1, u_2, \ldots, u_n]^T$ を，各量子ビットの回転角に対応させる角度エンコーディングを採用した．初期状態 $|0\rangle^{\otimes n}$ に対し，入力層のユニタリゲート $\hat{U}_{in}(\mathbf{u})$ は以下のように定義される．

$$U_{in}(\mathbf{u}) = \bigotimes_{j=1}^{n} R_y(u_j) \tag{4.2}$$

ここで，$R_y(\theta)$ はパウリ $\hat{Y}$ 行列による回転演算子である．この操作により，各量子ビットの状態は以下のように変換される．

$$|\psi_j\rangle = \cos\left(\frac{u_j}{2}\right)|0\rangle + \sin\left(\frac{u_j}{2}\right)|1\rangle \tag{4.3}$$

ここで，回転演算子の定義は $\hat{R}_y(\theta) = \exp\Big(-i\theta\hat{Y}/2\Big)$ に従う．このように，入力値 u_j は $\cos, \sin$ という非線形関数によって量子状態の振幅へと変換される．さらに，回路内に CNOT ゲートなどのエンタングルメント操作が含まれる場合，系全体の波動関数は個々の量子ビットのテンソル積と

なるため，振幅の項には $\cos(u_i/2)\sin(u_j/2)\ldots$ などの入力変数同士の積が現れる．この特性により，量子レザバーは入力データに対して非線形性を生むことができる．

4.2.3 実装

本プロジェクトでは，シミュレーション環境における量子レザバーの能力を評価する為に，量子回路の出力として測定値の平均ではなく，状態ベクトルの全振幅を直接特徴量として利用するアプローチを採用した．具体的な流れは以下の通りである．

データ生成と前処理

学習のターゲットとなるデータとして，薄膜の方程式の数値解を用いた．古典的なソルバーを用いて，空間グリッド数 $N = 700$, 時間 $T = 500$ における薄膜の高さ分布 $h(x,t)$ を生成した．このデータを量子回路の入力次元に合わせて $N_q = 64$ にダウンサンプリングした．

そして，高さデータを角度エンコーディングに対応させるため，以下の式を用いて回転角 $u \in [0, 2\pi]$ への変換を行った．時刻 t における生の入力ベクトル成分を $h_{j,raw}(t)$，その最大値・最小値をそれぞれ $h_{\max}, h_{\min}$ とするとき，エンコードされる角度データ $u_j(t)$ は以下で与えられる．

$$u_j(t) = 2\pi \times \frac{h_{j,raw}(t) - h_{\min}}{h_{\max} - h_{\min}} \tag{4.4}$$

この処理により，データの大小関係を保ちつつ，非線形性を最大限に活用することが可能となる．

リザバー層

入力データを拡散，混合させるために，エンコード後の状態に対して固定パラメータを持つ回転ゲート $\hat{R}_z(\phi)$（ϕ はランダムに初期化）と，隣接ビット間に CNOT ゲートを適用した．この操作を 3 回繰り返すことで，非線形性と複雑性を量子状態に付与する．

データの読出しと Ridge 回帰

量子回路による時間発展度の状態 $|\Psi(t)\rangle$ から情報を取り出すため，状態ベクトルの全振幅を用いた読み出し層を構成した．n 量子ビット系の状態ベクトルは 2^n 次元の複素ベクトル $\mathbf{c} \in C^{2^n}$ である．本実装では，この実部と虚部をそれぞれ独立した特徴量として連結し，合計 2×2^n 次元の特徴ベクトル $\mathbf{x}(t)$ を生成した．

$$\mathbf{x}(t) = \begin{bmatrix} \mathrm{Re}(\mathbf{c}) \\ \mathrm{Im}(\mathbf{c}) \end{bmatrix} \tag{4.5}$$

最後に，この特徴ベクトルを用いて薄膜の時間発展を学習させる．本プロジェクトでは，予測誤差の蓄積による評価のブレを排除し，量子回路の表現力を評価するため，1 ステップ先予測を用いた．特徴量 $\mathbf{x}(t)$ から次の時刻の状態 $\mathbf{y}(t + \Delta t)$ を予測する重み行列 W_{out} を学習する．

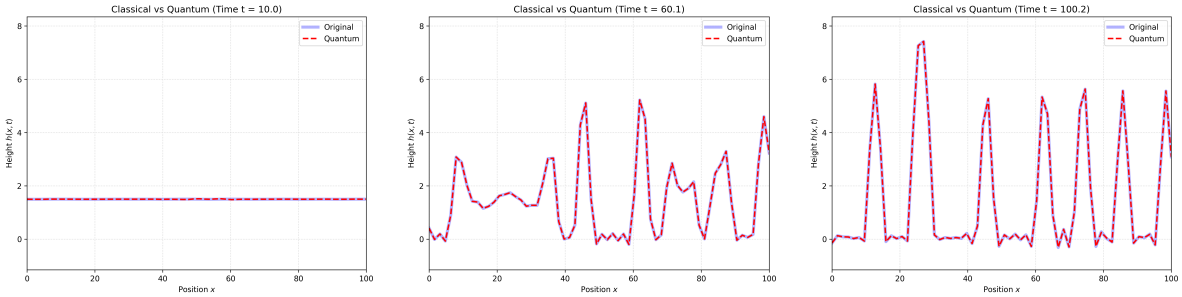
学習アルゴリズムには，Ridge 回帰を用いた．

4.3 結果

4.3.1 予測の精度

構築した量子レザバーを用いて，薄膜の方程式のダイナミクスの学習を行った結果を示す．図は，テストデータに対する 1 ステップ先予測の結果である．青線は古典ソルバーによる正解デー

タ，赤線は量子レザバーによる予測値を示している．



(a) $(t = 10)$　(b) $(t = 60)$　(c) $(t = 100)$

図 4.1　量子リザバーによる薄膜ダイナミクスの予測結果．青実線は古典数値解（正解），赤破線は量子回路による予測値を表す．

結果として，量子レザバーは，薄膜の方程式のダイナミクスを高い精度で追随していることが確認できる．これは．わずか 6 量子ビットという小規模な回路であっても偏微分方程式の複雑なダイナミクスを近似可能であることを示している．

4.4　考察

本プロジェクトでは, 線形なユニタリ発展に従う量子回路を用いて非線形微分方程式の挙動を再現することに成功した．一見矛盾しているこの結果について，要因を考察する

まず，シュレディンガー方程式は線形であり，時間発展演算子 $\hat{U}$ も線形である．

$$\hat{U}(\alpha\,|\psi\rangle + \beta\,|\phi\rangle) = \alpha\hat{U}\,|\psi\rangle + \beta\hat{U}\,|\phi\rangle \tag{4.6}$$

この式が示すように，量子回路のダイナミクスには非線形性は含まれていない．すなわち，量子状態そのものが非線形な変形をしているわけではない．

では，なぜ非線形な薄膜のダイナミクスの再現が可能であったかというと，前節で導入した角度エンコーディングに要因がある．入力データ u を非線形関数に変換して量子状態の振幅に埋め込むことで，線形な空間であるヒルベルト空間内に，データの非線形な特徴を射影している．言い換えると，本手法における量子回路は，非線形な演算を行っているのではなく，入力段階で非線形に変換された状態を，高次元空間で線形に回転や拡散をしているということになる．

しかし，物理シミュレーションとして考えると，問題点が存在する．QRC があくまでもダイナミクスの外見，入出力関係を数学的な関数として近似しているのに過ぎず，流体現象の物理的なメカニズムを量子回路上で実装しているわけではないという点である．従って，より物理的に忠実なシミュレーションを行うには，ほかのアプローチが必要である．

一方で，少ない量子ビットで薄膜のダイナミクスを表現できたことは，量子回路が持つ高い表現力と，情報を圧縮する能力の高さを示している．

4.4.1　付録 (QRC コード)

本セクションで使用した Python コードの主要部分を以下に示す．

Listing 4.1　QRC for Thin Film Dynamics

```
1 import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from scipy.signal import resample
from sklearn.linear_model import Ridge
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from numpy.fft import rfft, irfft


# 1. Physics Simulation (Thin Film Equation)
L, T, N = 100.0, 500.0, 700
dx = L/N
x = np.linspace(0, L, N, endpoint=False)
k = 2 * np.pi * np.fft.rfftfreq(N, d=dx)

def spectral_filter(k, cutoff=0.8):
    return np.exp(-(k / (cutoff * np.max(k)))**8)

def diff(h,k): return irfft((1.0j * k) * rfft(h) * spectral_filter(k))
def diff2(h,k): return irfft((1.0j * k)**2 * rfft(h) * spectral_filter(k))
def diff3(h,k): return irfft((1.0j * k)**3 * rfft(h) * spectral_filter(k))

def eq(t, h, k, beta, S, h_min, delta):
    phi1 = 1/(h**3 + 0.05)
    phi2 = -S * np.exp(delta * (1 - h/h_min))
    Q = h**2/3 * (3*beta + h)
    inner = -diff3(h, k) + diff(phi1, k) + diff(phi2, k)
    flux = Q * inner
    return diff(flux, k)

# Parameters and Initial Condition
h_min, beta, S, delta = 1.0e-3, 0.1, 0.01, 0.1
np.random.seed(0)
h0 = 1.5*np.ones(len(x)) - 0.01*np.random.rand(len(x))

# Solver Execution (Radau method)
sol = solve_ivp(eq, (0, T), h0, method='Radau', dense_output=True,
                args=(k, beta, S, h_min, delta))

```

```
# Sampling
t_eval = np.linspace(0, T, 600)
data_high_res = sol.sol(t_eval).T

# Downsampling for Quantum Input
N_Q = 64
data_h = resample(data_high_res, N_Q, axis=1)
x_low = np.linspace(0, L, N_Q)

# 2. Quantum Reservoir Computing (QRC)
def quantum_reservoir(data_input, n_qubits=6, depth=3):
    n_samples, n_features = data_input.shape
    # Angle Encoding
    d_min, d_max = np.min(data_input), np.max(data_input)
    scaled = 2 * np.pi * (data_input - d_min) / (d_max - d_min + 1e
        -6)

    features = []
    np.random.seed(42)
    rand_params = np.random.rand(depth, n_qubits) * 2 * np.pi

    for i in range(n_samples):
        qc = QuantumCircuit(n_qubits)
        current_in = scaled[i]

        # Reservoir Dynamics
        for d in range(depth):
            for q in range(n_qubits):
                idx = (q + d*n_qubits) % n_features
                qc.ry(current_in[idx], q)
                qc.rz(rand_params[d, q], q)
            for q in range(n_qubits-1): qc.cx(q, q+1)
            qc.cx(n_qubits-1, 0)

        # Readout
        sv = Statevector(qc).data
        features.append(np.concatenate([sv.real, sv.imag]))

    return np.array(features)

X_res = quantum_reservoir(data_h)

```

```
82
83 # 3. Training and Prediction
84 split = len(data_h) // 2
85 X_train = X_res[:split-1]
86 Y_train = data_h[1:split] # One-step ahead target
87
88 # Ridge Regression
89 reg = Ridge(alpha=1e-5)
90 reg.fit(X_train, Y_train)
91 Y_pred = reg.predict(X_res)
92
93 # 4. Visualization
94
95 target_times = [10.0, 60.0, 100.0] # Times to visualize
96
97 for t_target in target_times:
98     idx = (np.abs(t_eval - t_target)).argmin()
99     pred_idx = idx - 1
100
101     if pred_idx >= 0 and pred_idx < len(Y_pred):
102         plt.figure(figsize=(6, 4))
103         plt.plot(x_low, data_h[idx], 'b-', lw=3, alpha=0.4, label='
              Original')
104         plt.plot(x_low, Y_pred[pred_idx], 'r--', lw=2, label='
              Quantum')
105         plt.title(f"Time t = {t_eval[idx]:.1f}")
106         plt.xlabel("Position x")
107         plt.ylabel("Height h")
108         plt.legend()
109         plt.grid(True, linestyle=':')
110         # Save figure for the paper
111         plt.savefig(f"snapshot_t{t_eval[idx]:05.1f}.png", dpi=300)
112         plt.close()
```

4.5 量子ウォークによる流体のダイナミクスのシミュレーション

4.5.1 背景と先行研究

流体のシミュレーション

量子コンピュータを用いて流体現象をシミュレーションするには大きく 2 つのアプローチが用いられる．第一のアプローチは，流体の方程式を離散化し線形連立方程式に帰着させ，HHL アルゴリズムやハミルトニアンシミュレーションを用いて解く手法である．量子力学は線形な理論である

ため，この手法は線形方程式に対しては強力である．しかし，ナビエストークス方程式のような非線形な方程式を扱うためには，カーレマン線形化などの数理的手法を用いて，対象を無理やり線形に近似や変換する必要がある．この線形化のプロセスは，アルゴリズムの実装の難易度を上げる要因となっており，直感に基づいた柔軟な実装を困難にしている．

第二のアプローチは，微視的な粒子の挙動を量子系で模倣し，マクロな流体挙動を再現する手法である．このアプローチは，古典計算における LBM や LGA と同じ考えに基づいている．このアプローチの利点は，物理現象そのものを量子回路にマップするという点である．すなわち，ゲートの設計により直接物理系の現象を操作することができ，直感的な拡張が可能である．

量子ウォークによる流体の再現

第 2 のアプローチの中でも，量子ウォーク (Quantum Walk,QW) はその連続極限が Dirac 方程式，シュレディンガー方程式といった波動方程式に帰着するため，流体シミュレーションの有力な候補とされている．しかし，標準的な QW は，ユニタリ性 (可逆性) と線形性という量子力学の制約に縛られているため，古典流体にはある以下の 2 つの要素が欠けている．

1. 粘性: エネルギーが熱として散逸して起こる過程．ユニタリ発展する閉じた系では表現できない．
2. 非線形: 流速が自身を運ぶ移流項の効果．線形な系では表現できない．

これらを解決しないと QW を用いてナビエストークス方程式が支配する古典流体の挙動を再現することはできない．

先行研究

本プロジェクトで導入する散逸と非線形性の解決策については，それぞれ独立した先行研究が存在する．まず，散逸に関して開放量子系の枠組みを用いた研究が挙げられる．標準的な QW に Ancilla(補助) 量子ビットを導入し，相互作用させた後にその Ancilla を測定あるいはトレースアウトすることで，系に不可逆なデコヒーレンスを引き起こす手法である．[15] による Open Quantum Walk では，この操作によってユニタリな波動方程式ではなく，拡散方程式に近い挙動が得られることが示されている．

次に，非線形性の導入に関しては，粒子の存在確率（密度）に応じてコイン演算子を動的に変化させる手法が提案されている．鹿野らは，測定に基づくフィードフォワード制御を用いることで，非線形量子ウォークを実現できることを示した [16].

また，光 Kerr 効果のような物理的な非線形性を利用するアプローチもある．Anglés-Castillo らは，光の偏光状態をコインに見立てて,Kerr 媒質中での非線形偏光回転を利用することで，コインの回転角がその場所の光の強度 (粒子密度) に依存して変化するモデルを提唱した [17]．具体的には，コインの回転角 $\theta_{t,x}$ を以下の式 (4.7) のように状態依存の形式で記述する．

$$\theta_{t,x} = \theta_0 + \alpha |u_{t,x}||d_{t,x}| \sin \delta_{t,x} \tag{4.7}$$

ここで，θ_0 は定数項，α は非線形パラメータ,$|u_{t,x}||d_{t,x}|$ はそれぞれコイン状態上向き成分，下向き成分の振幅の絶対値，$\delta_{t,x}$ は両成分の位相の差を表す．この式により，粒子密度が高いほど回転角が変化し，強い相互作用が生じることになる．

彼らは，このモデルの連続極限において非線形 Dirac 方程式が導出されることを示し，数値計算によってソリトンの形成を確認している [17]．これは，非線形性が適切に導入されれば，QW 上で

移流項によって生じる構造をシミュレーション可能であることを示している．

しかし，既存の研究では散逸による拡散もしくは非線形性の導入のいずれか一方に焦点を当てたものが多い．ナビエストークス方程式が支配する古典流体をシミュレートするにはこれらの 2 つの要素を同時に量子アルゴリズムとして統合する必要がある．

4.5.2 量子ウォークの基礎と連続極限

本セクションでは，まず QW の基礎的な理論と性質を記述する．次に，QW のダイナミクスを生成消滅演算子を用いて記述し，その連続極限をとることで系の支配方程式が Dirac 方程式に帰着することを示す．これにより，離散的な格子上の移動が，流体の移流とみなすことの物理的な妥当性を明らかにする．

量子ウォークのアルゴリズム

量子ウォークは，古典的なランダムウォーク (Classical Random Walk,CRW) を量子力学的に拡張したモデルである．粒子は，位置空間 $\mathcal{H}_p$ とコイン空間 $\mathcal{H}_c$ のテンソル積空間 $\mathcal{H} = \mathcal{H}_p \otimes \mathcal{H}_c$ 上に存在する．

1 ステップの時間発展は以下の，コイン演算子 $\hat{C}$ とシフト演算子 $\hat{S}$ の積であるユニタリ演算子 $\hat{U}$ によって記述される．

$$|\Psi(t+1)\rangle = \hat{U}\,|\Psi(t)\rangle = \hat{S}\cdot(\hat{I}_p \otimes \hat{C})\,|\Psi(t)\rangle \tag{4.8}$$

- コイン演算:Collision 各格子点において粒子の内部状態を混合させる操作．流体における粒子の衝突に対応する．代表的なものとしてアダマールコインやグローバーコインが用いられる．
- シフト演算:Streaming コインの状態に応じて，粒子を隣接する格子点へ移動させる操作．流体における並進（移流）に対応する．

QW の特徴的な性質として，量子干渉がある．CRW が確率の和として拡散的に広がるのに対し，QW は振幅の和として干渉し合う．この干渉により，波束は崩れずに形を保ったまま広がる性質を持つ．この波として直進する性質は流体現象を記述するのに適している．

4.5.3 量子ウォークと場の理論

本セクションでは，量子ウォークを格子上の場の理論として定式化し，その連続極限において系が Dirac 方程式に帰着する過程を記述する．

格子上のスピノル場の記述

量子ウォークの状態は，格子点 x 上の 2 成分複素ベクトル $\Psi(x,t)$ によって記述される．

$$\Psi(x,t) = \begin{pmatrix} \psi_R(x,t) \\ \psi_L(x,t) \end{pmatrix} \tag{4.9}$$

この $\Psi(x,t)$ は時空上に分布するスピノル場 (Spinor Field) に相当する．

シミュレーションにおける 1 ステップの時間発展は，以下の差分方程式で定義される．

$$\Psi(x, t+\Delta t) = \hat{S}\cdot\hat{C}\Psi(x,t) \tag{4.10}$$

ここで $\hat{C}$ は成分間の混合を表すコイン演算子，$\hat{S}$ はシフト演算子である．

まず，コイン演算子 $\hat{C}$ について記述する．ここでは以下の形式を持つユニタリ行列を使う．

$$\hat{C}(\theta) = \begin{pmatrix} \cos\theta & i\sin\theta \\ i\sin\theta & \cos\theta \end{pmatrix} = \cos\theta \cdot \hat{I} + i\sin\theta \cdot \hat{\sigma}_x \tag{4.11}$$

ここで θ は混合の強さを表すパラメータである．混合が十分に弱いと仮定し，1 次近似を行うと，コイン演算子は以下のように表される．

$$\hat{C} \approx \begin{pmatrix} 1 & i\epsilon \\ i\epsilon & 1 \end{pmatrix} = \hat{I} + i\epsilon\hat{\sigma}_x \tag{4.12}$$

これは場が自分自身と局所的に反応する相互作用項に対応する．

次に，シフト演算子 $\hat{S}$ を定義する．本セクションでは，演算子 $\hat{S}$ の作用を，場の成分（ψ_R, ψ_L）に応じて空間座標を移動させる操作として以下のように定義する．

$$\hat{S}\begin{pmatrix} \psi_R(x) \\ \psi_L(x) \end{pmatrix} = \begin{pmatrix} \psi_R(x-\Delta x) \\ \psi_L(x+\Delta x) \end{pmatrix} \tag{4.13}$$

この定義は，上段（右成分）を正の方向へ，下段（左成分）を負の方向へ移流させることに対応する．

以上の定義を組み合わせると，1 ステップの時間発展は成分ごとに以下の式となる．

$$\begin{cases} \psi_R(x, t+\Delta t) \approx \psi_R(x-\Delta x, t) + i\epsilon\psi_L(x-\Delta x, t) \\ \psi_L(x, t+\Delta t) \approx \psi_L(x+\Delta x, t) + i\epsilon\psi_R(x+\Delta x, t) \end{cases} \tag{4.14}$$

ここで，第 2 項に着目すると，この項には微小パラメータ ϵ が掛けられている，位置のズレによる補正（$\epsilon \cdot \Delta x \partial_x \psi$）は微小量となり無視できる．したがって，相互作用項の空間座標を $x \pm \Delta x \approx x$ と近似すると，以下の方程式が得られる．

$$\begin{cases} \psi_R(x, t+\Delta t) \approx \psi_R(x-\Delta x, t) + i\epsilon\psi_L(x, t) \\ \psi_L(x, t+\Delta t) \approx \psi_L(x+\Delta x, t) + i\epsilon\psi_R(x, t) \end{cases} \tag{4.15}$$

この式において，右辺第 1 項は並進を，第 2 項は成分間の衝突を表している．

連続極限

格子間隔 $\Delta t, \Delta x$ を極限まで小さくすると，どう振る舞うか見ていく．

テイラー展開による線形化　時間ステップ Δt ，格子間隔 Δx を微小量とみなし，1 次のテイラー展開を行う．

左辺:

$$\psi_{R/L}(x, t+\Delta t) \approx \psi_{R/L}(x, t) + \Delta t \frac{\partial \psi_{R/L}}{\partial t} \tag{4.16}$$

右辺:

$$\psi_{R/L}(x \mp \Delta x, t) \approx \psi_{R/L}(x, t) \mp \Delta x \frac{\partial \psi_{R/L}}{\partial x} \tag{4.17}$$

これらを式 (4.15) に代入する．

ψ_R の展開:

$$\psi_R + \Delta t \frac{\partial \psi_R}{\partial t} = \left(\psi_R - \Delta x \frac{\partial \psi_R}{\partial x}\right) + i\epsilon\psi_L \tag{4.18}$$

両辺にある 0 次の項 $\psi_R(x,t)$ は相殺して消える．整理すると以下の方程式が得られる．

$$\Delta t\frac{\partial \psi_R}{\partial t} = -\Delta x\frac{\partial \psi_R}{\partial x} + i\epsilon\psi_L \tag{4.19}$$

ψ_L の展開:

$$\psi_L + \Delta t\frac{\partial \psi_L}{\partial t} = \left(\psi_L + \Delta x\frac{\partial \psi_L}{\partial x}\right) + i\epsilon\psi_R \tag{4.20}$$

同様に $\psi_L(x,t)$ を相殺して整理する．

$$\Delta t\frac{\partial \psi_L}{\partial t} = \Delta x\frac{\partial \psi_L}{\partial x} + i\epsilon\psi_R \tag{4.21}$$

行列形式への変換　得られた式 (4.19) と式 (4.21) を行列形式にまとめる．

(1) 時間発展方程式への変形

まず，左辺を時間微分の形 $\frac{\partial \Psi}{\partial t}$ にするため，両辺を時間ステップ Δt で割る．同時に，空間微分の係数として，格子上の移動速度 $c = \Delta x/\Delta t$ を定義する．

$$\frac{\partial}{\partial t}\begin{pmatrix}\psi_R\\ \psi_L\end{pmatrix} = \begin{pmatrix}-c\frac{\partial}{\partial x} & 0\\ 0 & c\frac{\partial}{\partial x}\end{pmatrix}\begin{pmatrix}\psi_R\\ \psi_L\end{pmatrix} + \frac{i\epsilon}{\Delta t}\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}\psi_R\\ \psi_L\end{pmatrix} \tag{4.22}$$

(2) 量子化

この数式を量子力学として扱うためには，位置 x と運動量 p が正準交換関係を満たす必要がある．

$$[x,\hat{p}] = x\hat{p} - \hat{p}x = i\hbar \tag{4.23}$$

空間微分を運動量演算子 $\hat{p} = -i\hbar\partial_x$ と対応づける．これに合わせて，時間微分もエネルギー演算子 $\hat{E} = i\hbar\partial_t$ と対応づけ，両辺に $i\hbar$ をかける．

- 時間微分 $\to$ エネルギー:$i\hbar\frac{\partial}{\partial t} \longrightarrow \hat{E}$
- 空間微分 $\to$ 運動量:$-i\hbar\frac{\partial}{\partial x} \longrightarrow \hat{p}$

実際に $i\hbar$ を掛けて項を整理すると，方程式は以下の形となる．

$$i\hbar\frac{\partial}{\partial t}\Psi = \underbrace{c\begin{pmatrix}-i\hbar\frac{\partial}{\partial x} & 0\\ 0 & i\hbar\frac{\partial}{\partial x}\end{pmatrix}}_{\text{運動項 } (c\cdot\hat{p})}\Psi + \underbrace{i\hbar\frac{i\epsilon}{\Delta t}\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}}_{\text{相互作用項}}\Psi \tag{4.24}$$

(3) パウリ行列と質量の定義

各項を整理する．右辺第 1 項（運動項）は，運動量演算子 $\hat{p}$ とパウリ行列 $\hat{\sigma}_z = \mathrm{diag}(1,-1)$ を用いることで，$c\hat{\sigma}_z\hat{p}$ と簡潔に記述できる．

次に，右辺第 2 項（相互作用項）の係数に着目する．

$$\text{係数} = i\hbar\frac{i\epsilon}{\Delta t} = -\frac{\hbar\epsilon}{\Delta t} \tag{4.25}$$

ここで，ϵ(コインの回転角) は負の値を仮定し，質量が正になるとする．この項はエネルギーの次元を持ち，かつ成分（カイラリティ）を混合させる役割を持つ．相対論的量子力学において，カイラリティを反転させるエネルギーこそが質量の定義である．そこで，この係数の大きさを mc^2 と定義する．

$$mc^2 \equiv -\frac{\hbar\epsilon}{\Delta t} \tag{4.26}$$

これにより，パラメータ ϵ が，質量 m になった．最終的に，パウリ行列 $\hat{\sigma}_x$ を用いて記述すると，以下の Dirac 方程式が得られる．

$$i\hbar\frac{\partial\Psi}{\partial t} = \left(c\hat{\sigma}_z\hat{p} + mc^2\hat{\sigma}_x\right)\Psi = \left(-i\hbar c\hat{\sigma}_z\frac{\partial}{\partial x} + mc^2\hat{\sigma}_x\right)\Psi \tag{4.27}$$

Step 3: 物理的解釈　この結果は，QW の各要素が，連続極限において以下のように物理量へ変換されたことを示している．

- 差分（$\pm\Delta x$） $\longrightarrow$ 運動量演算子（$-i\hbar\partial_x$）
- 右移動と左移動の区別 $\longrightarrow$ 進行方向を表す行列（$\hat{\sigma}_z$）
- コイン回転角（ϵ） $\longrightarrow$ 質量エネルギー（mc^2）

流体の方程式との関係

最後に，導出された Dirac 方程式が，流体力学の基礎方程式とどのような関係にあるかを確認する．

古典流体力学において，密度 ρ と流束 j は以下の連続の式を満たす．

$$\frac{\partial\rho}{\partial t} + \frac{\partial j}{\partial x} = 0 \tag{4.28}$$

QW のモデルがこの関係式を満たしているか確認する．

まず，QW おける密度 $\rho(x,t)$ を，各成分の振幅の二乗の和として定義する．

$$\rho(x,t) = |\psi_R(x,t)|^2 + |\psi_L(x,t)|^2 \tag{4.29}$$

これは量子力学では確率密度に対応するが，流体モデルとしては質量密度と解釈される．

この密度 ρ の時間変化 $\frac{\partial\rho}{\partial t}$ を，Dirac 方程式を用いて計算する．具体的な成分計算を行うと，以下の 2 つに分けられる．

1. 質量項（コイン演算）の寄与　右辺に含まれる質量項は，複素共役をとって和をとるとプラスとマイナスで相殺し合う．

$$\left(\frac{\partial\rho}{\partial t}\right)_{\text{mass}} = 0 \tag{4.30}$$

これは，コインを振って成分を混ぜ合わせても，その地点にある総量は変わらないという意味である．

2. 運動項（シフト演算）の寄与　次に，運動項の寄与を整理する．

$$\begin{aligned}\left(\frac{\partial\rho}{\partial t}\right)_{\text{kin}} &= -c\frac{\partial}{\partial x}(|\psi_R|^2) + c\frac{\partial}{\partial x}(|\psi_L|^2) \\ &= -\frac{\partial}{\partial x}\left[c(|\psi_R|^2 - |\psi_L|^2)\right]\end{aligned} \tag{4.31}$$

ここで，右辺の括弧内は右への流れと左への流れの差であり，これを流束カレント j と定義できる．

$$j(x,t) = c(|\psi_R(x,t)|^2 - |\psi_L(x,t)|^2) \tag{4.32}$$

結論　以上の結果をまとめると，密度と流束の間には以下の関係が成り立つことが確認できる．

$$\frac{\partial \rho}{\partial t} = -\frac{\partial j}{\partial x} \quad \Longleftrightarrow \quad \frac{\partial \rho}{\partial t} + \frac{\partial j}{\partial x} = 0 \tag{4.33}$$

この式は，流体力学における連続の式そのものである．これにより，QW よって記述されるダイナミクスは，ミクロな量子力学的振る舞いを持ちながらも，マクロには流体として性質を持つことを示した．

4.6　粘性と非線形性の導入

本セクションでは，前セクションで定義した QW のモデルに対し，古典流体が持つ粘性と非線形性を導入する為に拡張したモデルを構築する．

4.6.1　散逸の導入による粘性の再現

情報を捨てることで粘性を生む

ナビエストークス方程式において，粘性項 $\nu\nabla^2\mathbf{v}$ は，流体の運動量が拡散してエネルギーが熱として失われる過程 (不可逆) を表している．しかし，標準的な QW はユニタリ発展 (可逆) に支配されているためエネルギーは保存され，粘性のない流体のようなふるまいをする．

これを粘性のある流体へと変化させるために，不可逆性を導入する必要がある．本プロジェクトでは，この過程を量子回路上で再現する為に，Ancilla ビットを導入する手法を用いる．

1. 環境の用意: ウォーカー（粒子）のほかに，熱浴となる Ancilla ビットを用意する．
2. 衝突: ウォーカーとアンシラを相互作用させる．これは，粒子が熱浴の分子と衝突し，互いの状態が相関を持つことに対応する．
3. 情報の破棄: 相互作用の後，Ancilla ビットの状態を測定せず，意図的に無視する．

この無視するという操作により，系が持っていた情報の一部が Ancilla とともに系の外に捨てられる．結果として，純粋な波としての性質が部分的に破壊され，散逸が現れる．

情報，エントロピー，エネルギーの関係　情報の破棄が粘性と等価だといえる事について，補足する．情報理論において，系の状態に対する不確かさは，シャノンエントロピーによって定義される．Ancilla ビットをトレースアウトする (相互作用の結果を無視する) ことは，系の正確な状態がわからなくなることを意味し，これは情報エントロピーが増大するということである．

いっぽう，流体において粘性とは水分子同士の摩擦抵抗でうまれる．摩擦によって運動エネルギーが熱エネルギーに変われば，系全体の熱力学的なエントロピーは増大する．

つまり，アルゴリズム上で情報を捨てるという操作は，物理現象としての粘性と同じくエントロピーを増大させる，不可逆な変化を起こす役割を持つ．

密度演算子と量子チャネル

系の散逸を記述するため，開放量子系の枠組みを用いる．なお，本セクションにおける密度演算子，量子チャネルの定式化は Nielsen and Chuang[18] に基づいている．詳細に関しては，[18] の第 8 章を参照されたい．以下では，本手法で必要な部分のみ説明する．

密度演算子　通常，閉じた系の純粋状態はベクトル $|\psi\rangle$ で記述される．一方，散逸を伴う開放系の状態 (混合状態) は以下の密度演算子 ρ を用いる必要がある．

$$\rho = \sum_i p_i \left|\psi_i\right\rangle \left\langle\psi_i\right| \tag{4.34}$$

ここで，行列の各成分には以下の物理的意味がある．

- **対角項 (ρ_{ii}):** 粒子の存在確率（流体の密度分布）．
- **非対角項 (ρ_{ij}):** 波の干渉能力（流体の直進性）．

量子チャネル　開放系における時間発展は，ユニタリ変換ではなく量子チャネルとして記述される．これは，CPTP 写像として定義され，その作用は Kraus 表現を用いて以下のように書かれる．

$$\mathcal{E}(\rho) = \sum_k \hat{M}_k \rho \hat{M}_k^\dagger \tag{4.35}$$

ここで $\hat{M}_k$ は Kraus 演算子であり，完全性条件 $\sum_k \hat{M}_k^\dagger \hat{M}_k = \hat{I}$ を満たす．これを用いることで，保存則を満たして開放系のダイナミクスを記述することができる．

4.6.2　Ancilla による位相緩和

流体の粘性を再現するための具体的な散逸モデルを構築する．流体における粘性はミクロには粒子間の衝突によって生じる．これを再現するために，位置の情報を変えずに波の干渉のみを部分的に破壊する操作を用いる．そこで，Ancilla ビットを用いた位相緩和チャネルを用いる．

部分相互作用とトレース

1 ステップの時間発展を以下のプロセスとして定式化する．

Step1: 相互作用　時刻 t における系の状態 $\rho(t)$ に対して，初期状態 $|0\rangle_a$ の Ancilla ビットを加える．系と Ancilla ビットの間に散逸強度 p $(0 \leq p \leq 1)$ のパラメータを持つ以下のユニタリオペレータ $\hat{U}_{int}$ を適用する．

$$\hat{U}_{int}(|\psi\rangle \otimes |0\rangle_a) = \sqrt{1-p}\,|\psi\rangle \otimes |0\rangle_a + \sqrt{p}(\hat{\sigma}_z |\psi\rangle) \otimes |1\rangle_a \tag{4.36}$$

この操作は，確率 p で系に位相フリップ $\hat{\sigma}_z$ を作用させ，フリップしたか否かを Ancilla ビットに転写することに対応する．ここで，$\hat{\sigma}_z$ を用いるのは，粒子の居場所はそのままで，波の符号のみをランダムに反転させるためである．この操作は，流体のミクロな粒子間の衝突を表現する．以上により，系と環境の間には量子もつれが形成される．

Step2: 情報の破棄　相互作用後，Ancilla ビットの状態を観測せずに破棄する．これは，Ancilla ビットがとりうるすべての状態について，それぞれの状態をとる確率を足し合わせることに相当する．この，観測されない自由度についての足し合わせは，全系の密度行列に対して，Ancilla の自由度についての部分トレースをとる操作として定義できる．

$$\rho(t+\Delta t) = \mathrm{Tr}_a \left[\hat{U}_{int}(\rho(t) \otimes |0\rangle \langle 0|_a)\hat{U}_{int}^\dagger\right] \tag{4.37}$$

この計算を実行すると，以下の Kraus 演算子 $\{\hat{M}_0, \hat{M}_1\}$ によって記述される時間発展方程式が得られる．

$$\rho(t+\Delta t) = (1-p)\rho(t) + p\hat{\sigma}_z \rho(t) \hat{\sigma}_z \tag{4.38}$$

ここで，Kraus 演算子は以下のように定義される．

- $\hat{M}_0 = \sqrt{1-p}\hat{I}$：散逸なし（恒等変換）
- $\hat{M}_1 = \sqrt{p}\hat{\sigma}_z$：散逸あり（位相フリップ）

物理的解釈

前セクションに基づき，密度演算子の各成分がどのように変化するかを考える．$\hat{\sigma}_z$ の性質（対角成分は不変，非対角成分は符号反転）により，以下の結果が得られる．

- **対角項の保存:** 対角成分 ρ_{ii} は粒子の存在確率を表す．$\hat{\sigma}_z$ は位相を反転させるのみで確率振幅の絶対値を変えないため，

$$\rho_{ii} \to (1-p)\rho_{ii} + p(\hat{\sigma}_z \rho_{ii} \hat{\sigma}_z) = \rho_{ii} \tag{4.39}$$

  となり，値は保存される．これは，流体の質量や運動量そのものを減衰させるのではなく，保存則を満たしたダイナミクスであることを意味する．
- **非対角項の減衰:** 異なるコイン状態間の非対角成分 ρ_{ij} $(i \neq j)$ は，波の干渉能力を表す．$\hat{\sigma}_z$ の作用により，

$$\rho_{ij} \to (1-p)\rho_{ij} + p(-\rho_{ij}) = (1-2p)\rho_{ij} \tag{4.40}$$

  となり，係数 $(1-2p)$ で減衰する．これは，散逸強度 p に応じて，系が本来持っていた波として直進する能力が削がれることを意味する．

以上のミクロな変化がマクロなスケールではどのような変化をもたらすか考察する．散逸パラメータ p が 0 の場合，粘性のない流体に対応する．一方，$p > 0$ とし位相緩和を導入すると，干渉項の減衰により直進性が失われ，拡散的な挙動となる．

以上の操作は物理的には粒子が確率的に衝突し運動の方向が乱されるプロセスを再現したものである．実際の流体において，粘性が生じるミクロなプロセスは，この分子間の衝突における運動量の拡散にある．よって，本モデルにおける位相緩和プロセスは，単なる減衰ではなく，流体の粘性の発生の機構を模倣しているといえる．結果として，p を操作することで粘性のある流体をシミュレーションすることと同じとなる．

衝突モデルの拡張：外力項の導入

前項まで述べた Ancilla ビットによる散逸過程は，物理的には系が「低温の環境（$|0\rangle_a$）」と衝突し，エネルギーを失うプロセスと解釈できる．この解釈を拡張すると，全く逆のプロセスとして，系へのエネルギー注入（駆動）も同一の枠組みで記述可能である．

もし環境として「励起状態（$|1\rangle_a$）」にある Ancilla ビットを用意し，系と衝突させれば，エネルギーは環境から系へと流入する．

$$\text{Dissipation (Viscosity): } \rho \leftarrow \mathrm{Tr}_a[U(\rho \otimes |0\rangle\langle 0|)U^\dagger] \quad \text{(Loss)} \tag{4.41}$$

$$\text{Pump (External Force): } \rho \leftarrow \mathrm{Tr}_a[U(\rho \otimes |1\rangle\langle 1|)U^\dagger] \quad \text{(Gain)} \tag{4.42}$$

このエネルギー注入プロセスは，流体力学において流体を加速させる外力項に対応する．したがって，本モデルにおける Ancilla ビットとの相互作用は，環境の状態を制御することで，粘性項 $\nu\nabla^2\mathbf{v}$ と外力項 $\mathbf{F}$ の両方を量子アルゴリズム上で実装する機構となっている．

粘性量子ウォークのモデル

本セクションの話をまとめ，粘性を導入した QW のアルゴリズムを定義する．

標準的な QW がユニタリ演算子 $\hat{U}$ のみによる可逆な発展であったのに対し，提案モデルはユニタリ過程と散逸過程の 2 段階で構成される．1 ステップの時間発展 $\rho(t) \to \rho(t+\Delta t)$ のアルゴリズムは以下の通りである．

$$\rho(t+\Delta t) = \mathcal{E}_{dissipate}\left(\mathcal{U}_{unitary}(\rho(t))\right) \tag{4.43}$$

ここで，各ステップは具体的に以下の演算を行う．

1. **ユニタリ過程 $\mathcal{U}$:** 標準的なコイン・シフト操作により，粒子の移動を行う．

$$\rho'(t) = \hat{S}\hat{C}\rho(t)\hat{C}^{\dagger}\hat{S}^{\dagger} \tag{4.44}$$

2. **散逸過程 $\mathcal{E}$:** 更新された状態 $\rho'(t)$ に対し，Ancilla ビットを用いた位相緩和を行う．

$$\rho(t+\Delta t) = (1-p)\rho'(t) + p(\hat{I}_x \otimes \hat{\sigma}_z)\rho'(t)(\hat{I}_x \otimes \hat{\sigma}_z) \tag{4.45}$$

このモデルにより，QW に粘性を導入できた．次セクションでは，このモデルにさらに非線形性を導入し，より古典流体に近づけることを目指す．

4.6.3 非線形性の導入

前セクションで，粘性を導入することはできた．しかし，ナビエストークス方程式における移流項 $(\mathbf{v}\cdot\nabla)\mathbf{v}$ に由来する非線形性はまだ導入されていない．量子力学は線形なシュレーディンガー方程式に従うため，このような非線形効果は表れない．そこで，光学的，流体力学のアナロジーに基づいてコイン演算子を拡張することで非線形性を導入する．

光 Kerr 効果と Madelung 変換

本モデルの構築に当たり，非線形光学における光 Kerr 効果と量子力学の流体力学的記述である Madelung 変換に着目する．

Madelung 変換　量子力学的な波動関数と古典流体のアナロジーは,Madelung 変換によって与えられる．波動関数 $\psi(\mathbf{r},t)$ を，密度 ρ と位相 S を用いて以下のように極形式で記述する．

$$\psi(\mathbf{r},t) = \sqrt{\rho(\mathbf{r},t)}\exp\left(iS(\mathbf{r},t)\right) \tag{4.46}$$

ここで，ρ と位相 S は，波動関数から以下のように定義される．

$$\rho(\mathbf{r},t) = |\psi(\mathbf{r},t)|^2 \tag{4.47}$$

$$S(\mathbf{r},t) = \arg\left(\psi(\mathbf{r},t)\right) \tag{4.48}$$

ここで，ρ は流体の密度に対応する．この表現をシュレーディンガー方程式に代入して整理すると，位相 S の空間勾配が流体の流速 $\mathbf{v}$ に対応することが導かれる．

$$\mathbf{v}(\mathbf{r},t) \propto \nabla S(\mathbf{r},t) \tag{4.49}$$

さらに，この関係式を用いることで，シュレーディンガー方程式は形式的に以下の流体の方程式（オイラー方程式）と等価になることが知られている．

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla(V + Q) \tag{4.50}$$

この式は，波動関数の位相分布を変化させることは流体の速度場を変化させることと等価であるということを意味する．

光 Kerr 効果　非線形光学において，光の強度が媒質の屈折率を変化させる Kerr 効果が知られている．光の強度（粒子密度）を $I \propto \rho$ とすると，屈折率 n は以下のように変化する．

$$n(\rho) = n_0 + n_2\rho \tag{4.51}$$

屈折率の変化は，波の位相速度の変化，すなわち位相 S の変化をもたらす．以上の光 Kerr 効果と Madelung 変換を用いることで，以下が成立する．

1. 密度 ρ が局所的な屈折率（位相 S）を変化させる（カー効果）．
2. 位相 S の空間的な変化は，流速 $\mathbf{v}$ の勾配に対応する（Madelung 変換）．
3. したがって，密度 ρ の分布が，流速場 $\mathbf{v}$ に流れを生み出す．

このプロセスは，流体の密度分布が原因で速度場を変化させる，流体方程式の非線形項の役割を持っている．

密度依存型演算子

上記に基づき，コイン演算子のパラメータを密度に依存させるモデルを導入する．[17] ではカー効果に着目し，コイン演算子の回転角をコイン状態や光強度に依存させることで QW に非線形性を導入できる事を示した．この着想を，流体力学とのアナロジーに基づいて実装するため，各ステップにおいて粒子密度に応じた位相変調を起こすモデルを用いる．

具体的には，非線形位相演算子 $\hat{N}$ を導入する．まず，位置 $\mathbf{x}$ における局所的な密度 $d(\mathbf{x}, t)$ は，密度行列 $\rho(t)$ の対角成分の和として得られる．

$$d(\mathbf{x}, t) = \sum_c \langle \mathbf{x}, c|\rho(t)|\mathbf{x}, c\rangle \tag{4.52}$$

ここで c は内部状態（コイン状態）のインデックスである．

この密度 $d(\mathbf{x}, t)$ を用いて，非線形位相シフト演算子 $\hat{N}(\rho)$ を以下のように定義する．

$$\hat{N}(\rho) = \sum_{\mathbf{x}} \exp\left(ig \cdot d(\mathbf{x}, t)\right) |\mathbf{x}\rangle \langle \mathbf{x}| \otimes \hat{I}_c \tag{4.53}$$

ここで，g は非線形性の強さを決定する定数である．この操作は，物理的には密度に比例して位相をシフトさせることに相当する，

$$\mathcal{N}_{nonlinear}(\rho) = \hat{N}(\rho)\rho\hat{N}^\dagger(\rho) \tag{4.54}$$

このモデルにより，粒子の密度が高い場所では局所的に位相が回転し，Madelung 変換における関係式 $\mathbf{v} \propto \nabla S$ を通じて，流速場に勾配が生じる．結果として，流体が自分自身の密度分布によって流れを変える, 移流項 $(\mathbf{v} \cdot \nabla)\mathbf{v}$ に見られるような非線形なダイナミクスが再現される．

非線形粘性量子ウォークのモデル

以上の話をまとめ，粘性と非線形性の両方を統合した量子ウォークのアルゴリズムを定義する．モデルの 1 ステップの時間発展 $\rho(t) \to \rho(t+\Delta t)$ は，以下の順序で行われる．

$$\rho(t+\Delta t) = \mathcal{S}_{shift}\left(\mathcal{E}_{visc}\left(\mathcal{C}_{coin}\left(\mathcal{N}_{nonlinear}(\rho(t))\right)\right)\right) \tag{4.55}$$

ここで，各ステップは具体的に以下の演算を行う．

1. **Step 1: 非線形過程 ($\mathcal{N}_{nonlinear}$)** 密度分布に基づき，自己位相変調を行う．

$$\rho'(t) = \hat{N}(\rho)\rho(t)\hat{N}^{\dagger}(\rho) \tag{4.56}$$

2. **Step 2: コイン過程 ($\mathcal{C}_{coin}$)** 標準的なコイン演算子 $\hat{C}$ により，内部状態を混合する．

$$\rho''(t) = (\hat{I}_p \otimes \hat{C})\rho'(t)(\hat{I}_p \otimes \hat{C})^{\dagger} \tag{4.57}$$

3. **Step 3: 粘性散逸過程 ($\mathcal{E}_{visc}$)** 前節で定義した Ancilla ビットによる位相緩和を行う．

$$\rho'''(t) = (1-p)\rho''(t) + p(\hat{I}_p \otimes \hat{\sigma}_z)\rho''(t)(\hat{I}_p \otimes \hat{\sigma}_z) \tag{4.58}$$

4. **Step 4: シフト過程 ($\mathcal{S}_{shift}$)** 最後に，コイン状態に応じて隣接格子点へ移動する．

$$\rho(t+\Delta t) = \hat{S}\rho'''(t)\hat{S}^{\dagger} \tag{4.59}$$

これにより，QW に非線形性と粘性を導入することができた．

4.7 実装と数値シミュレーション

本セクションでは，提案したアルゴリズムの実装と，それを用いた数値シミュレーションの結果について述べる．ここでは，ゲート型量子コンピュータで動作するアルゴリズムを構築したが，その有用性を評価するためには，アルゴリズムそのものが適切に動作するか確認する必要がある．そこで，本実装では構築した量子回路の挙動を行列計算ベースの古典シミュレータを用いた．これにより，量子ビット数やハードウェアの制限を受けずにアルゴリズムが理想的に動作したのかを確認できる．まずは．量子回路の実装コードを示し，続いて実際に検証に用いた行列計算の手法について述べる．

4.7.1 量子回路の実装コード

本アルゴリズムは，Qiskit ライブラリを用いて実装した．

Listing 4.2 D2Q9 Quantum Walk

```
1 import numpy as np
2 from qiskit import QuantumCircuit, QuantumRegister,
      ClassicalRegister
3 from qiskit.circuit.library import UnitaryGate, RZGate, RYGate
4
5 class ViscousD2Q9Circuit:
6     """
```

```
7       1. Source Term:External energy injection
8       2. Nonlinearity: Fluid density dependent phase modulation
9       3. Collision:Relaxation processes with parameterized coin
            operators
10      4. Forcing & Viscosity: Non-unitary energy transfer using an
            ancillary system
11      5. Streaming: Moving particles with state-dependent shift
            operators
12      6. Boundary: Implementing an absorbing barrier using quantum
            oracles and measurements
13      """
14      def init(self, grid_bits_x, grid_bits_y):
15          self.nx = grid_bits_x
16          self.ny = grid_bits_y
17          self.dim_x = 2**self.nx
18          self.dim_y = 2**self.ny
19
20
21          self.qr_x = QuantumRegister(self.nx, 'x')       # location
                coordinates X
22          self.qr_y = QuantumRegister(self.ny, 'y')       # location
                coordinates Y
23          self.qr_c = QuantumRegister(4, 'c_d2q9')        # Coin
24          self.qr_a = QuantumRegister(1, 'env_ancilla')  #
                Environmental Ancilla
25
26          self.cr_geom = ClassicalRegister(1, 'geom_flag')
27
28          self.qc = QuantumCircuit(self.qr_x, self.qr_y, self.qr_c,
                self.qr_a, self.cr_geom)
29
30      def add_increment_gate(self, target_reg, inverse=False):
31          n = len(target_reg)
32          sub_qc = QuantumCircuit(n)
33          if not inverse:
34              for i in range(n - 1, 0, -1):
35                  sub_qc.mcx(list(range(i)), i)
36              sub_qc.x(0)
37              gate = sub_qc.to_gate(label='+1')
38          else:
39              sub_qc.x(0)
40              for i in range(1, n):
```

```
41                 sub_qc.mcx(list(range(i)), i)
42             gate = sub_qc.to_gate(label='-1')
43         return gate
44
45     def apply_source(self, inlet_coords, strength):
46
47         if not inlet_coords: return
48
49         self.qc.x(self.qr_a) # Activate Ancilla
50
51         for (x, y) in inlet_coords:
52             # Flip 0-bits to 1 to target specific (x, y)
53             for i in range(self.ny):
54                 if not ((y >> i) & 1): self.qc.x(self.qr_y[i])
55             for i in range(self.nx):
56                 if not ((x >> i) & 1): self.qc.x(self.qr_x[i])
57
58             # Apply Controlled-RY (All-1 control)
59             num_ctrls = self.ny + self.nx + 1
60             gate = RYGate(strength * np.pi).control(num_ctrls)
61             ctrl_qubits = list(self.qr_y) + list(self.qr_x) + [self
                   .qr_a[0]]
62             self.qc.append(gate, ctrl_qubits + [self.qr_c[0]])
63
64             # Uncompute bit flips
65             for i in range(self.nx):
66                 if not ((x >> i) & 1): self.qc.x(self.qr_x[i])
67             for i in range(self.ny):
68                 if not ((y >> i) & 1): self.qc.x(self.qr_y[i])
69
70         self.qc.reset(self.qr_a)
71
72     def nonlinear_modulation(self, density_map, g):
73
74         for x in range(self.dim_x):
75             for y in range(self.dim_y):
76                 rho = density_map[x, y]
77                 theta = g * rho
78                 if abs(theta) < 1e-6: continue
79
80                 # Specify the control state
81                 ctrl_val = (x << self.ny) + y
```

```
82                  # Apply phase shift only to particles at specified
                        coordinates
83                  gate = RZGate(theta).control(self.nx + self.ny,
                        ctrl_state=ctrl_val)
84                  self.qc.append(gate, list(self.qr_y) + list(self.
                        qr_x) + [self.qr_c[0]])
85
86      def coin_mixing(self, theta):
87          N = 9
88          if theta is None:
89              # Grover coin
90              D = np.full((N, N), 2.0/N) - np.eye(N)
91              label = 'Grover Coin'
92          else:
93              # Generalized coin with adjustable spreading strength
94              s = np.ones((N, 1)) / np.sqrt(N)
95              P = s @ s.T
96              factor = 1.0 - np.exp(1j * theta)
97              D = np.eye(N) - factor * P
98              label = f'Coin({theta:.2f})'
99
100         U = np.eye(16, dtype=complex); U[:N, :N] = D
101         self.qc.append(UnitaryGate(U, label=label), self.qr_c)
102
103     def pump_and_viscosity(self, pump_strength, visc_strength):
104
105         # 1. Pump : Energy transfer from Ancilla (|1>) to Coin
106         if pump_strength > 1e-6:
107             self.qc.x(self.qr_a)
108             self.qc.cry(pump_strength * np.pi, self.qr_a[0], self.
                        qr_c[0])
109             self.qc.reset(self.qr_a)
110
111         # 2. Viscosity : Transfer energy from coin to Ancilla ->
                Reset and discard outside the system
112         if visc_strength > 1e-6:
113             self.qc.cry(visc_strength * np.pi, self.qr_c[0], self.
                        qr_a[0])
114             self.qc.reset(self.qr_a)
115
116     def shift_advection(self):
117
```

```
118         inc_x = self._add_increment_gate(self.qr_x, inverse=False)
119         dec_x = self._add_increment_gate(self.qr_x, inverse=True)
120         inc_y = self._add_increment_gate(self.qr_y, inverse=False)
121         dec_y = self._add_increment_gate(self.qr_y, inverse=True)
122 
123         # c=1:Right, 2:Up, 3:Left, 4:Down, 5:UR, 6:UL, 7:DL, 8:DR
124         ctrl_map = [
125             (inc_x, 1), (inc_y, 2), (dec_x, 3), (dec_y, 4),
126             (inc_x, 5), (inc_y, 5), (dec_x, 6), (inc_y, 6),
127             (dec_x, 7), (dec_y, 7), (inc_x, 8), (dec_y, 8)
128         ]
129 
130         for gate, state in ctrl_map:
131 
132             target_qubits = list(self.qr_c) + list(self.qr_x) if
                    gate in [inc_x, dec_x] \
133                                 else list(self.qr_c) + list(self.qr_y)
134             self.qc.append(gate.control(4, ctrl_state=state),
                    target_qubits)
135 
136     def boudary_condition(self):
137 
138         def flag_boundary(reg, val_bits, target):
139             for i, b in enumerate(val_bits):
140                 if b == '0': self.qc.x(reg[i])
141             self.qc.mcx(list(reg), target)
142             for i, b in enumerate(val_bits):
143                 if b == '0': self.qc.x(reg[i])
144 
145         flag_boundary(self.qr_x, '0'*self.nx, self.qr_a[0])
146         flag_boundary(self.qr_x, '1'*self.nx, self.qr_a[0])
147         flag_boundary(self.qr_y, '0'*self.ny, self.qr_a[0])
148         flag_boundary(self.qr_y, '1'*self.ny, self.qr_a[0])
149 
150         self.qc.measure(self.qr_a, self.cr_geom)
151         with self.qc.if_test((self.cr_geom, 1)):
152              self.qc.reset(self.qr_c)
153              self.qc.reset(self.qr_x)
154              self.qc.reset(self.qr_y)
155         self.qc.reset(self.qr_a)
156 
157     def build_step(self, density_map, nonlinear_g=2.0, coin_theta=
```

```
        None,
158                     inlet_coords=[], pump_strength=0.1,
                            visc_strength=0.05):
159 #Build a quantum circuit for one time step
160         self.qc.data = []
161         self.apply_source(inlet_coords, 0.2)
162         self.nonlinear_modulation(density_map, nonlinear_g)
163         self.coin_mixing(coin_theta)
164         self.pump_and_viscosity(pump_strength, visc_strength)
165         self.shift_advection()
166         self.boudary_condition()
167         return self.qc
```

4.7.2 コードの解説

上記に示したコードは，QW のシミュレーションの 1 ステップを量子回路として定義したものである．本セクションでは，このコードがどのようにして非線形性，粘性，境界条件などを量子ビットの操作に変換しているのかをプログラムに沿って記述する．

1. 量子ビットの準備 (init)

量子回路には，役割の異なる 4 種類の量子ビットのグループ（レジスタ）を用意した．

- qr_x,qr_y: 粒子の位置を表す．
- qr_c: 粒子の進行方向を表す．LBM の D2Q9 モデルに従って，9 個の状態を区別する為に 4 量子ビットを割り当てた．
- qr_a: 環境 (熱浴) を表す Ancilla ビット．

2. エネルギー注入 (apply_source)

シミュレーションにおいて，特定の位置にエネルギーを注入する操作である．系への一方的なエネルギー流入を実現する為に，Ancilla ビットのリセットによる非ユニタリ操作として実装した．標準的な多重制御ゲートは，制御ビットがすべて $|1\rangle$ の状態に対して作用するように定義されている．だが，指定したい座標は，$|0\rangle$ を含む場合がある．そこで，以下の工夫を行った．

1. ターゲットの指定: 位置レジスタ（$|x\rangle, |y\rangle$）を制御ビットとして使用する．エネルギー注入をしたい座標のビットが $|0\rangle$ の場合は X ゲートで反転させる．
2. エネルギーの注入: コインレジスタをターゲットとして，Ancilla ビットと連動した回転ゲートをかけ，その地点における状態の振幅を増幅させる．
3. 処理: 反転させたビットを再度反転ゲートで元にもどす．また，使用した Ancilla ビットをリセットする．

3. 密度分布による相互作用 (nonlinear_modulation)

各地点の密度マップを読み込み，密度 ρ に係数 をかけた角度 $theta$ だけ位相ゲートを回転させている．ここで，係数の g は非線形性の強さを調整するパラメータである．

4. 粒子間の衝突と散乱 (coin_mixing)

それぞれ独立した波として振る舞っていた粒子同士を衝突させ，進行方向を混ぜ合わせる．

- コイン演算子の役割: コインレジスタに対し，各方向の振幅を混合するユニタリオペレータ (コインオペレータ) を適用する．
- 拡散の強さの調整: 引数の θ によって，混ぜ合わせ具合を調整する．θ が指定されない場合，Grover コインと呼ばれる演算子が適用される．これは入射方向にかかわらず全方向に均等に確率を散らす操作で，最も拡散率が強い．

5. 散逸と外力の再現 (pump_and_viscosity)

Ancilla ビットを用いて散逸と外力を再現した．

- 散逸: まず、基底状態 $|0\rangle_a$ にある Ancilla ビットとコインレジスタに対し，制御回転ゲートを適用してをエンタングルさせる。これにより、系のエネルギーの一部が環境へ移動する。その後、Ancilla ビットをリセットして初期化することで、移動したエネルギーを系外へ排出し、不可逆な粘性散逸を実現する。
- 外力: 環境ビットに反転ゲートを適用して励起状態 $|1\rangle_a$ を用意する。このエネルギーを持った環境と系に対して制御回転ゲートを適用し、エンタングルさせることで、環境から系へのエネルギー注入（流入）を行い、外力項を表現する。その後，Ancilla ビットをリセットする．

6. 粒子の移動 (shift_advection)

コインの状態に従って，粒子を隣接する格子点へ移動させる．この操作では，コインレジスタを制御ビット，位置レジスタをターゲットビットとする条件付き加算ゲートを適用している．この操作により，コインと位置の間にエンタングルメントが生まれる．

7. 境界条件 (boudary_condition)

境界条件として，吸収境界条件を実装した．これは，領域の端のみではなく，空間内に配置された障害物に対しても同様に適用できる．

- 境界への到達判定: まず，粒子の位置が境界座標に一致しているかを判断する．位置レジスタを制御ビット，Ancilla ビットをターゲットとする多重制御ゲートを構成し，粒子が壁の位置に在る場合のみ，Ancilla ビットを $|1\rangle$ に反転させる．
- 吸い出し: 次にこの Ancilla ビットを即座に測定する．もし測定結果が $|1\rangle$ 出会った場合，粒子の状態をリセットして強制的に初期化する．

行列計算に基づく古典検証

前セクションで構築した量子回路アルゴリズムの妥当性を検証するため，Python を用いた行列シミュレーションを行った．量子力学において，有限次元の量子状態はヒルベルト空間上のベクトルであり，量子ゲートはその空間上のユニタリ行列として記述される．従って，量子コンピュータを使わなくてもこれらのベクトルと行列の演算を古典コンピュータ上で厳密に計算すれば，ノイズのない理想的な量子アルゴリズムの検証が可能である．

検証用シミュレーションコード

本プロジェクトで実際に使用したコード示す．

Listing 4.3 D2Q9 Quantum Walk(matrix)

```
 1   import numpy as np
 2 from scipy import sparse
 3
 4 class Config:
 5     # Grid
 6     GRID_Y = 100
 7     GRID_X = 300
 8     STEPS  = 20000
 9
10     # Parameters
11     VISCOSITY_DECAY = 0.95
12     NONLINEAR_G     = 20.0
13     PUMP_STRENGTH   = 0.1
14
15     # 3Injection Conditions
16     INJECTION_AMP   = 0.02
17     NOISE_LEVEL     = 0.0
18     INJECTION_WIDTH = 30
19     DIFFUSION_THETA = np.pi / 5.0
20
21     # Numerical Stabilization
22     MAX_ENERGY_CAP  = 500
23
24 DIM_COIN = 9
25
26 #barrier
27 BARRIER_MASK = np.ones((Config.GRID_Y, Config.GRID_X))
28 cx, cy, r = int(Config.GRID_X/5), int(Config.GRID_Y/2), int(Config.
       GRID_Y/7)
29 y_grid, x_grid = np.ogrid[:Config.GRID_Y, :Config.GRID_X]
30 BARRIER_MASK[(x_grid - cx)**2 + (y_grid - cy)**2 < r**2] = 0.0
```

```
31
32 # D2Q9 Shift Rules
33 SHIFT_RULES = [
34     (0, 0), (0, 1), (-1, 0), (0, -1), (1, 0),
35     (-1, 1), (-1, -1), (1, -1), (1, 1)
36 ]
37
38 def get_coin_op(theta):
39     #Construct Coin Operator
40     s = np.ones((DIM_COIN, 1)) / np.sqrt(DIM_COIN)
41     P = s @ s.T
42     return np.eye(DIM_COIN) - (1.0 - np.exp(1j * theta)) * P
43
44 def get_pump_op(strength):
45     Construct Pump Operator+
46     bias = np.ones(DIM_COIN)
47
48     # Gain for Right-going components
49     bias[1] = 1.0 + strength
50     bias[5] = 1.0 + strength * 0.7
51     bias[8] = 1.0 + strength * 0.7
52
53     # Loss for Left-going components
54     bias[3] = 1.0 - strength
55     bias[6] = 1.0 - strength * 0.7
56     bias[7] = 1.0 - strength * 0.7
57     return np.diag(bias)
58
59 def run_simulation():
60     #Time Evolution
61     U_coin = get_coin_op(Config.DIFFUSION_THETA)
62     U_pump = get_pump_op(Config.PUMP_STRENGTH)
63     decay  = Config.VISCOSITY_DECAY
64     g_val  = Config.NONLINEAR_G
65
66     # Initialize State Vector
67     psi = np.zeros((Config.GRID_Y, Config.GRID_X, DIM_COIN), dtype=
           complex)
68
69     # Setup Injection Source
70     mid = Config.GRID_Y // 2
71     y_coords = np.arange(Config.GRID_Y)
```

```
72      sigma = Config.INJECTION_WIDTH / 2.0
73      profile = np.exp(-((y_coords - mid)**2) / (2 * sigma**2))
74
75      injection = np.zeros_like(psi)
76      injection[:, 0, 1] = profile * Config.INJECTION_AMP * 0.6 #
            Right
77      injection[:, 0, 5] = profile * Config.INJECTION_AMP * 0.2 # Up-
            Right
78      injection[:, 0, 8] = profile * Config.INJECTION_AMP * 0.2 #
            Down-Right
79
80      mask = BARRIER_MASK[:, :, None]
81      history = []
82
83      print(f"Simulating {Config.STEPS} steps...")
84
85      for t in range(Config.STEPS):
86          # 1. Source Injection
87          psi += injection
88
89          # 2. Nonlinear Phase Modulation
90          if abs(g_val) > 1e-5:
91              density = np.sum(np.abs(psi)**2, axis=2, keepdims=True)
92              psi *= np.exp(1j * g_val * density)
93
94          # 3. Coin Mixing & Pump
95          psi = np.dot(psi, U_coin.T)
96          psi = np.dot(psi, U_pump.T)
97          psi *= decay # Viscosity
98          psi *= mask  # Boundary
99
100         # 4. Streaming (Shift)
101         new_psi = np.zeros_like(psi)
102         for c in range(DIM_COIN):
103             dy, dx = SHIFT_RULES[c]
104             new_psi[:, :, c] = np.roll(psi[:, :, c], shift=(dy, dx)
                    , axis=(0, 1))
105         psi = new_psi
106
107         # 5. Boundary cleanup
108         psi[:, 0, :] = 0; psi[:, -1, :] = 0
109         psi[0, :, :] = 0; psi[-1, :, :] = 0
```

```
110
111         # Record History (Density)
112         if t % 2 == 0:
113             history.append(np.sum(np.abs(psi)**2, axis=2))
114
115     return history
```

量子回路実装との対応

上記のシミュレーションコードは，前セクションの量子回路とほぼ同じ挙動を示すが，以下の近似と安定化処理を行っている．

- ゲート操作と行列計算の対応: 量子回路におけるコイン演算やシフトはそれぞれ行列計算におけるユニタリ行列の積，配列のインデックス操作として実装している．
- 障害物 (境界条件) の実装: 量子回路において障害物，境界条件は特定座標の反転，マーキングによって実装されるが，行列シミュレーションでは空間マスク配列との要素ごとの積として実装した．障害物領域を 0 として毎ステップ掛け算することで，障害物内部に粒子が侵入することを阻止している．
- 非ユニタリ操作の処理: 量子回路における散逸は，Ancilla ビットへのエネルギー移動とリセットによって実装されたが，これは系の波動関数のノルムが減っていることと同じである．そのため，行列シミュレーションでは状態ベクトル全体に 1 未満の実数係数をかけることで，エネルギーの減衰過程を実装した．同様に，外力についても増幅係数の掛け算として実装した．
- 数値的な発散防止: 量子回路におけるエネルギー注入は，回転ゲートによる確率振幅の遷移として実装されるため，その値は発散することはない．一方，行列計算ではこれを単純な実数係数の掛け算で実装している．そのため，発散してしまう可能性がある．これを防ぐためにエネルギーの上限を設けた．

4.8 シミュレーション結果と考察

前セクションで構築したシミュレーション環境を用い，円柱障害物周りの挙動を解析した．

密度分布の可視化

図 4.2 に十分に時間がたった後での密度分布 ($|\psi(x,y)|^2$) を示す．上段は生の計算結果，下段は，ガウシアンフィルタ ($\sigma = 2.0$) を適用したものである．図より，左端から流入した粒子が円柱によって散乱され，背後に後流を形成している様子が確認できる．

考察

本プロジェクトにおいて，QW を構成するパラメータとナビエストークス方程式における流体の物理量との厳密な対応付は，課題として残っている．しかし，シミュレーションによって可視化された現象を流体現象と紐づけて考察することは可能である．4.2 の結果には，以下の流体的な構造が現れている．

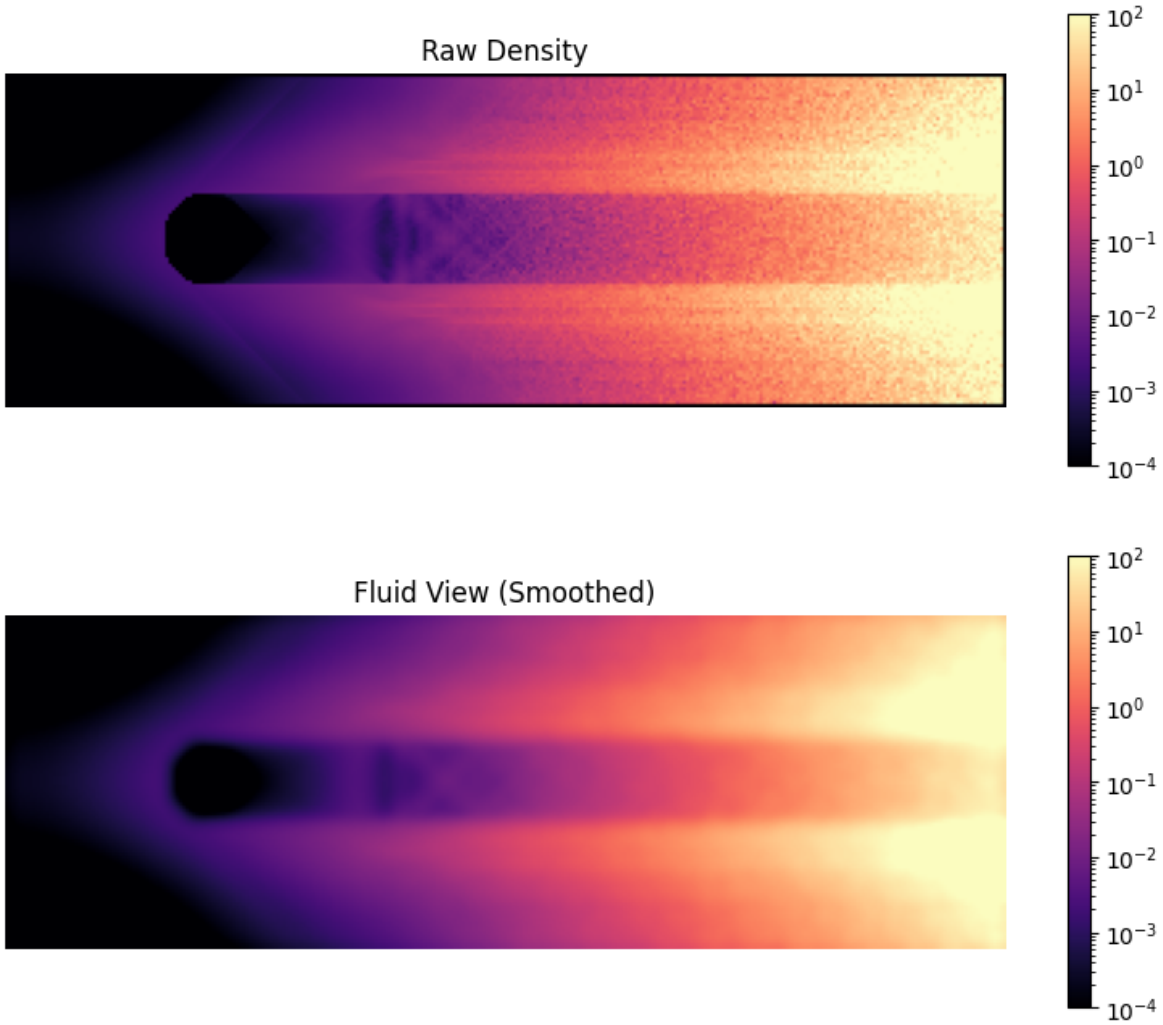


図 4.2　Simulation result of quantum walk-based fluid dynamics ($T = 20000$). The upper panel shows the raw density distribution. The lower panel shows the smoothed fluid view.

1. 左右対称性と流れの安定性: 円柱のに対し，巨視的には左右対称に流れが保たれている．カルマン渦の発生に伴うような対称性の破れは見られないことから，安定した層流状態にあると考えられる．
2. 後流の形成: 円柱の直後に密度が著しく低下した領域がのびている様子が確認できる．この領域が時間発展後に一定の形状を保っているため，本シミュレーションが粘性の強い安定した流体の挙動を再現していると考えられる．

また，流れの中に，格子状の干渉縞を確認することができる．これは，本シミュレーションが波動関数の重ね合わせに基づいていることによって生じる現象だと考えられる．本シミュレーションにおいて，明確なカルマン渦の観察には至らなかったが，層流の挙動を再現することに成功した．これは，提案アルゴリズムが流体現象を再現していることを示している．また，量子干渉縞を確認できたことより，量子力学的な性質も保持していると考えらる．

4.9 QW についてのまとめ

本セクションでは，拡張した QW について，シミュレーションした結果と課題について述べる．

結果

本プロジェクトの主要な成果は以下の主に以下の 3 つである．

1. 非ユニタリ，流体回路の設計: 標準的な QW に対し，Ancilla ビット用いた散逸過程と自己位相変調による非線形過程を組み込むことで，流体現象を模倣する量子回路を具体的に構築した．これは可逆な量子力学系を用いて，不可逆性を有するマクロな流体現象をシミュレーションするための枠組みを構築するものである．
2. 層流と後流の再現: 円柱障害物周りのシミュレーションにおいて，安定した層流の後流の形成を確認した．この結果は．提案アルゴリズムが，流体のダイナミクスを正しくシミュレーション出来ていることを示す．

課題

本プロジェクトで得られた結果に基づき，現状の課題について考える．

パラメータ調整の困難

本シミュレーションにおいて，カルマン渦や乱流への遷移が観測されずに，層流にとどまった要因は，アルゴリズムに含まれるパラメータの多さとその調整の難易度にある．提案手法では，相互に影響し合う多数の自由度を持っており，これらが流体の挙動を決定している．この多数のパラメータの組み合わせの中から，層流から乱流へと遷移する特定の条件を見つけ出すには，手作業では限界があり，今回の検証の中ではその領域まで探索しきれなかったことが課題として残る．

拡張モデルにおける連続極限の数学的導出

理論面に最大の課題は，連続極限の導出である．標準的な QW が Dirac 方程式に帰着することを示した．しかし，本プロジェクトで拡張した粘性項や非線形項を含むモデルが連続極限において厳密に流体に関する方程式に帰着するかどうかは，未完了である．現在は，現象論的な考えに基づいて実装を行っており，この拡張 QW から流体方程式の導出を完成させることが，本手法における物理的な妥当性を示すうえで重要な課題である．

（※文責: 大場英輝）

第 5 章　まとめ

5.1　各流体シミュレーションのまとめ

本プロジェクトでは，流体を量子コンピュータ上で扱うという課題に対して，古典，量子の 2 つの視点から検証を行った．各パートで得られた知見を統合して，総括を記述する．

dewetting のシミュレーション

dewetting の方程式の方程式に関しては，古典ソルバーによるデータ生成と，量子レザバーコンピューティングによる学習を行った．

- 古典パート: Dewetting の支配方程式を数値計算で解いた。その際は、数値的発散を防ぐため自由エネルギーの形状を変えて計算を行った。結果、dewetting に見られるの膜の破裂と穴の形成を確認することができた。
- 量子パート:QRC を用いたアプローチでは，古典パートが生成したデータを用いて学習を行い，量子回路が非線形ダイナミクスをどの程度模倣できるかを検証した．結果として，少数の量子ビットを用いた回路であっても，複雑な挙動を再現できることが確認された．

カルマン渦のシミュレーション

流体のシミュレーションに関しては, 格子ボルツマン法と拡張した量子ウォークの実装を行った．

- 古典パート: 古典コンピュータを用いた数値流体力学の一例として LBM を導入し，これを用いて流体のシミュレーションを試みた．その結果，LBM によってカルマン渦のシミュレーションを行うことができた．また，レイノルズ数の変化によって渦の挙動が変化することも確認された．
- 量子パート:LBM の計算プロセスとのアルゴリズム的な対応関係に着目した量子回路を設計し，シミュレーションを行った．その結果，得られた密度分布において，古典流体に見られる典型的な挙動を確認した．これにより，提案した量子アルゴリズムが，流体力学的な現象を再現できていることが示された．

5.2　結言

本プロジェクトを通して，古典計算と量子計算は，現時点では相互補完的な関係にあることが再確認できた．新しい量子アルゴリズムを提案するにあたって，そのアルゴリズムの妥当性を評価するために，確立された古典シミュレーションによる答え合わせが必要不可欠である．本プロジェクトにおいては，量子アルゴリズムの提案の段階までしか至らなかった．今後は，今回得られた結果を基に，古典解との定量的な比較を行うことが課題となる．

参考文献

[1] J.S.Rowlinson and B.Widom (2013)*Molecular Theory of Capillarity*, Dover Publications, pp.26-28.

[2] S.A.Safran, 好村滋行訳 (2001) **コロイドの物理学**, 吉岡書店.

[3] Pierre-Gilles de Gennes, Françoise Brochard-Wyart and David Quéré, 奥村剛訳 (2003) **表面張力の物理学 : しずく, あわ, みずたま, さざなみの世界**, 吉岡書店.

[4] R.V.Craster and O.K.Matar (2009) *Dynamics and stability of thin liquid films*, Reviews of Modern Physics, Vol.81.

[5] 高橋和孝, 西森秀稔 (2017) **相転移・臨界現象とくりこみ群**, 丸善出版.

[6] 田崎晴明 (2023) **熱力学 = 現代的な視点から**, 培風館.

[7] 瀬田剛. 格子ボルツマン法. 森北出版, 2021.

[8] Robert Cahalan. NASA, 1999.

[9] 蔦原道久, 高田尚樹, 片岡武. 格子気体法・格子ボルツマン法ー新しい数値流体力学の手法ー. コロナ社, 1999.

[10] Taro Imamura, Kojiro Suzuki, Takashi Nakamura, Masahiro Yoshida, Masahiro Fukuda. Numerical Analysis of the Flow around a Cylinder at High Reynolds Number using the Lattice Boltzmann Method. JSCFD, 2001.

[11] Schroeder D. Fluid Dynamics Simulation. https://physics.weber.edu/schroeder/fluids/.

[12] Cesar A. Amaral, Vinícius L. Oliveira, Juan P. L. C. Salazar, and Eduardo I. Duzzioni. Quantum machine learning and quantum-inspired methods applied to computational fluid dynamics: a short review, 2025.

[13] Kohei Nakajima, Keisuke Fujii, Makoto Negoro, Kosuke Mitarai, and Masahiro Kitagawa. Boosting computational power through spatial multiplexing in quantum reservoir computing. *Physical Review Applied*, 11(3), March 2019.

[14] 田中 剛平. リザバーコンピューティングの概念と最近の動向. **電子情報通信学会誌**, 102(2):108–113, 2019.

[15] S. Attal, F. Petruccione, C. Sabot, and I. Sinayskiy. Open quantum random walks. *Journal of Statistical Physics*, 147(4):832–852, May 2012.

[16] Yutaka Shikano, Tatsuaki Wada, and Junsei Horikawa. Discrete-time quantum walk with feed-forward quantum coin. 4(1):4427, 2014.

[17] Andreu Anglés-Castillo, Armando Pérez, and Eugenio Roldán. Solitons in a photonic nonlinear quantum walk: lessons from the continuum, 2023.

[18] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.