2016 年度桂田研究室卒業レポート

スペクトル法を用いた偏微分方程式の解法

明治大学　総合数理学部　現象数理学科
大平　明日河

2017 年 2 月 15 日

目 次

第1章　はじめに **3**
1.1　目的 3
1.2　レポートの流れ 3

第2章　スペクトル法とは何か **4**
2.1　スペクトル法 4
2.2　スペクトル法の長所と短所 5

第3章　Fourier 級数・変換と直交性 **7**
3.1　Fourier 級数 7
3.2　直交性 7
3.2.1　直交系 7
3.2.2　内積・ノルム 8
3.3　Fourier 変換 9
3.3.1　Fourier 変換と反転公式 9
3.4　畳み込み 11
3.4.1　畳み込みとは 11
3.5　線形移流方程式 15
3.5.1　1次元線形移流方程式とは 15
3.5.2　スペクトル法を用いて解く 16
3.6　非線形移流方程式 17
3.6.1　非線形移流方程式とは 17
3.6.2　方程式の説明 17
3.6.3　スペクトル法を用いて解く 18
3.7　数値的安定性 19
3.7.1　数値的安定性とは 19
3.7.2　前進 Euler 法の安定性 19
3.7.3　硬い方程式 20
3.7.4　後退 Euler 法 21

第 4 章　線形・非線形移流方程式のプログラム **23**
4.1 FFT 23
4.2 線形移流方程式の数値計算 23
4.2.1 実数値 DFT 23
4.2.2 前進 Euler 法によるプログラム 25
4.2.3 後退 Euler 法によるプログラム 27
4.3 非線形移流方程式の数値計算 30
4.3.1 変換法 30
4.3.2 変換法なしのプログラム 32
4.3.3 変換法ありのプログラム 35
4.3.4 変換法のあり・なしの比較 40

第 5 章　まとめ **41**

第1章　はじめに

1.1　目的

金星の大気の循環は今までの観測事実から高度 70 km 付近で緯度－６０°～６０°の広範囲に渡って約　100 m/s の東風が吹いていることが分かっている。金星の自転速度というのは 1.5m/s であるので約 60 倍の速度で金星の自転方向に風が吹いている。この自転を超える速度で吹く風をスーパーローテーションと呼ぶ。金星の他にもタイタンと呼ばれる土星の第六衛星でも確認されている。しかし、確認されているのがこの二つの星のみであるので現在、確認されている星の大気循環の中では非常に珍しい現象である。また、この現象は未だにどのように起こるのか詳しく分かっていない。この現象が生じる原因については、昼夜間対流に着目する仮説、重力波に着目する仮説、子午面循環に着目す仮説という 3 つの仮説がある。この仮説を元に現在でもこの現象の原因究明の研究は進められている。そして、私の目的として自身で仮説をたて、数値計算によって、再現実験を現象の解明を修士課程での最終目的としている。

そのためにスーパーローテーションや大気循環モデルの数値計算でよく用いられるスペクトル法を前段階として理解することが必要である。

よって本研究の目的は、スペクトル法の数値計算の原理を石岡（2004）に沿って解説し、それに加えて、Fourier 変換との関係を解説していく。そして、単純な問題である 1 次元線形移流方程式、少し複雑な問題である 1 次元非線形移流方程式スペクトル法を用いて数値計算を行った。

今回扱う非線形移流方程式を Navier-Stokes 方程式の変形であり、大気の運動方程式の中にも含まれている。大気循環モデルの要素の一つをなり得る方程式を実際にスペクトル法によって数値計算の原理を学ぶことは、意義があることと考えられる。

1.2　レポートの流れ

第2章　スペクトル法とは何か

2.1　スペクトル法

スペクトル法とは、偏微分方程式の解 u を固有関数も用い展開しその固有関数に付く展開係数と呼ばれる係数を満足するような常微分方程式を導きそれを解くことによって解 u を求める方法である。

正式名称　ガラーキンスペクトル法順を追ってガラーキンスペクトル法について詳しく解説する。$J-1$ 個の独立な関数 $\varphi_j(x)$ ($j=1,2,\cdots,J-1$) を用意して、

$$u(x,t)=\sum_{j=1}^{J-1} a_j(t)\varphi_j(x)$$

と離散化することを考える。ここで展開関数と呼ばれる $\varphi_j(x)$ は境界条件に対応して、$\varphi_j(0)=\varphi_j(1)=0$ ($j=1,2,\cdots,,J-1$) を満たすように選んでおくものとする。ここの級数を用いて、以下の方法で $a_j(x)$ を満たす常微分方程式を求めていく。

方法　選点法 $J-1$ 個の分点 x_j を定め、元の偏微分方程式がこの分点上で成立していることを求める。例として熱方程式 $(\partial u(x,t))/\partial t=\ \kappa\ (\partial^2 u(x,t))/\partial x^2$ で考える。級数を代入すると、

$$\sum_{k=1}^{J-1}\frac{da_k(t)}{dt}\varphi_k(x_j)=\kappa\sum_{k=1}^{J-1}a_k(t)\left(\frac{\partial^2\varphi_k(x)}{\partial x^2}\right)_{x=x_j}\qquad(j=1,2,\cdots,J-1)$$

上記のように偏微分方程式が展開される。これは、分点 x_j ($j=1,2,\cdots,J-1$) で上記の式を見たいしていることを表している。しかし、未だに展開関数 $\varphi_j(x)$ を選ぶときの任意性が残っている。ここで用いられる展開関数 φ_j の選び方の方法をガラーキンスペクトル法と呼ぶ。

方法　ガラーキンスペクトル法展開関数 φ_j として境界条件を満たすように直交関数系を用いる方法をガラーキンスペクトル法と呼ぶ（一般にスペクトル法と呼ぶ)。例として同様の熱方程式を用いると、展開関数は境界条件に合わせて $\sin(j\pi x)$

を用いる。[1]よって、

$$u(x,t)=\sum_{j=1}^{J-1}a_j(t)\sin(j\pi x)$$

と展開する。すると、熱方程式は、

$$\sum_{j=1}^{J-1}\frac{da_j(t)}{dt}\sin(j\pi x)=-\kappa\sum_{j=1}^{J-1}a_j(j\pi)^2\sin(j\pi x)$$

となる。直交関数の性質から、全ての $\sin(j\pi x)$ は一次独立なので、任意の $\sin(j\pi x)$ に対して、

$$\frac{da_j(t)}{dt}=-\kappa a_j(j\pi)^2 \quad (m=1,2,\cdots,,J-1)$$

という常微分方程式を得る。初期条件は、Fourier 変換を用いることで得られる。よって、係数を満足するような常微分方程式が得られた。これがスペクトル法である。

2.2 スペクトル法の長所と短所

ここでは、差分法とスペクトル法の違いに沿って長所と短所をまとめる。長所として

- 通常、展開関数として滑らかな関数系を使うので偏微分を差分近似することに伴う数値的分散性[2]がない。また、展開関数系として適切なものを使えば、展開の収束が非常に速い[3]。よって、差分法に比べて高精度の解が得られる。

[1]展開関数と呼ばれる関数 φ_j は固有関数になっており、解 u を固有関数で展開している。ここで固有関数は、L を何らかの演算子とすると、偏微分方程式が

$$\varphi_j(\partial u(x,t))/\partial t=L[u(x,t)]$$

という形でかける時、今スペクトル法で解くために、

$$u=\sum_{j=1}^{J-1}a_j(t)\varphi_j(x)$$

上記のように級数展開する。この時、関数 φ_j は直交関数で演算子 L の固有関数となり、β_j を固有値とすると、

$$L[\varphi_j(x)]=\beta_j\varphi_j(x)$$

を満たしている。つまり熱方程式の時、演算子である $\partial^2/(\partial x^2)$ に対する固有関数は、$\sin(j\pi x)$ となっているので $\sin(j\pi x)$ を用いた。

[2]数値解法によって得られる近似値が数値解法に依存してしまい、本来のパターンが崩れてしまうこと。

[3]近似解が厳密解に近くのが速いということ。

- 展開関数系として、偏微分方程式に含まれる演算子の固有関数系を使うことができれば、離散化として得られる常微分方程式が非常に簡単になる。

短所として

- 境界条件や領域の形が複雑な場合は展開関数系が簡単に更生できない。
- 係数 $a_j(t)$ と分点上の物理量が直接に対応していないので、対応付けのために変換コストがかかる。
- 非線形問題を扱う場合には特別な空が必要になる。

以上の長所と短所について以降で考察していく。

第3章　Fourier級数・変換と直交性

3.1　Fourier級数

$f : \mathbb{R} \to \mathbb{C}$ は周期関数で滑らかな関数とする。f の Fourier 係数

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)dx \quad (n = 0, 1, 2, \dots), \quad b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx \quad (n = 1, 2, \dots)$$

と定めると、f の Fourier 級数は、

$$a_0/2 + \sum_{n=1}^{\infty}(a_n\cos(nx) + b_n\sin(nx)) \quad (x \in \mathbb{R})$$

これが基本的なFourier級数展開である。ここでEulerの公式を用いると複素Fourier級数という形になる。($e^{i\theta} = \cos\theta + i\sin\theta$) 同様の条件で、

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx$$

と定めると、f の Fourier 級数は、

$$\sum_{n=-\infty}^{\infty} c_n e^{inx} = \lim_{N\to\infty}\sum_{n=-N}^{N} c_n e^{inx}$$

これが複素 Fourier 級数展開である。

3.2　直交性

3.2.1　直交系

相違なる関数 $_{j,k}$ 同士の積の積分が 0 という性質を持つ関数 $_j$ は直交系という。例えば、関数系 $\{e^{ijx}\}_{j\in\mathbb{Z}}$ なら、

となり、関数系 $\{e^{ijx}\}_{j\in\mathbb{Z}}$ は直交系である。他にも関数系 $\{\cos(nx)\}_n \cup \sin(nx)_{n\in\mathbb{N}}$ も直交系である。

3.2.2 内積・ノルム

関数の範囲を $P_2\pi\{f|f:\mathbb{R}\to C$ 区分的に滑らかで、周期 2π の周期関数とするとこれは、加法とスカラー倍を自然に定義すれば、C 上のベクトル空間（線形空間）になる。$f,g\in P_2\pi$ に対して、f と g との内積 (f,g) を

$$(f,g)=\int_{-\pi}^{\pi}f(x)(g(x))dx$$

定める。内積は次の 1〜3 の公理を満たす。1　任意の $f_1\in X$ に対して $(f,f)\geq 0.$ 等号が成り立つためには、$f=0$ が必要十分条件. 2　任意の $f,g\in X$ に対して $(g,f)=(f,g)$ 3　任意の $f_1,f_2,g\in X,\lambda_1,\lambda_2\in\mathbb{C}$ に対して $(\lambda_1 f_1+\lambda_2 f_2,g)=\lambda_1(f_1,g)+\lambda_2(f_2,g)$. f のノルムを

$$\|f\|:=\sqrt{(f,f)}$$

で定める。ノルムは次の 1〜3 の公理を満たす。（ノルムの公理）

1. 任意の $f\in X$ に対して $\|f\|\geq 0.$ 等号が成り立つためには、$f=0$ ０ が必要十分条件.
2. 任意の $f\in X,\lambda\in C$ に対して,$\|\lambda f\|=|\lambda|\|f\|$.
3. 任意の $f,g\in X$ に対して、$\|f+g\|\leq\|f\|+\|g\|$ ここで述べた内積を使うと、直交系、正規直交系は以下のようにかける。X を C または $\mathbb{R}$ 上の内積空間、ϕ_n を X の要素の集合とする。

- ϕ_n が X の直交系であるとは、次の ２ 条件 1、2 が成り立つことをいう。２ ϕ_n が X の正規直交系であるとは、

$$(\phi_m,\phi_n)=\delta_m n$$

$$\delta_{mn}=\begin{cases}1 & (m=n)\\ 0 & (m\neq n)\end{cases}$$

が成り立つことを言う。また、直交系から正規直交系を作ることができる。（正規化）$\{\phi_n\}_n$ が内積空間 X の直交系である時、

$$\psi_n=\frac{1}{\|\phi_n\|}\phi_n$$

とおくと、$\{\psi_n\}_n$ は X の正規直交系となる。

3.3 Fourier 変換

3.3.1 Fourier 変換と反転公式

一般に関数 f に対して、f の Fourier 変換 $TheFourierTransformoff$ と呼ばれる関数 $\hat{f} = Ff$ を

$$f(\xi) = Ff(\xi)\frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)e^{-i\xi x}dx \quad (\xi \in \mathbb{R}\ )$$

定義する。また、写像 $Ff \to f$ のことも Fourier 変換と呼ぶ.(Fourier transform) F は適当な仮定のもとで、次が成り立つ。

$$f(x) = \int_{-\infty}^{\infty} f(x)e^{i\xi x}d\xi \quad (((x \in \mathbb{R})))$$

これは一般に、関数 g に対する共役な Fourier 変換と呼ばれ、関数 $g = F^{*}g$ を

$$g(x) = F^{*}g(x) = \int_{-\infty}^{\infty} g(\xi)e^{i\xi}xd\xi \quad ((x \in \mathbb{R}))$$

で定義する。また、写像 $F^{*} : g \to g$ 自身も共役な Fourier 変換と呼ぶ。これを用いること次のようにかける。

$$F^{*}(Ff) = f\hat{F}(F^{*}g) = g$$

これが Fourier の反転公式である。3.3.2 Fourier 級数と Fourier 変換ここで Fourier 級数と Fourier 変換を見比べる。周期 2 π の関数 $f : \mathbb{R} \to \mathbb{C}$ に対して、f の Fourier 係数 $c_{n n\in\mathbb{Z}}$ を

$$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx(n \in \mathbb{Z})$$

と定めたが、これは f の Fourier 変換 $\hat{f}(\xi) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)e^{-i\xi x}dx\ (\xi \in \mathbb{R})$ に相当している。また、適当な条件のもとで Fourier 級数展開

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}$$

が成り立ち、これが Fourier 反転公 $f(x) = 1/\sqrt{2}\pi\int_{-\pi}^{\pi} \hat{f}(x)e^{i\xi x}dx((x \in \mathbb{R}))$ に相当する。3.4 離散 Fourier 変換（DFT）離散 Fourier 変換は、Fourier 係数・

Fourier 級数展開の離散化と言える。周期 2 π（周期 T なら $nx \to 2n\pi x/T$ と書き換える）で滑らかな関数とする。

$$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx(n \in \mathbb{Z})$$

($[-\pi, \pi]$ => $[0, 2\pi]$ へと変更しても同じ値になるので可能) とおくと、$[0, 2\pi]$ を N 等分（等分点を $x_j = jh(h = 2\pi/N), (j = 0, 1, 2, \ldots, N)$）での値を $f_j = f(x_j)$ を用いて、C_n の近似値 C_n は、台形則（注意１）を用いると、

$$\hat{f}_n = \frac{1}{2\pi}h\sum_{j=0} j^{N-1} f(x_j)e^{-inx_j}$$

$$x_j = jh, h = 2\pi/N$$

であるので、

$$e^{ih} = e^{2\pi i/N} = \omega$$

（注意２）とすれば、$f(x)$ これを離散 Fourier 係数と呼ぶ。

$\boldsymbol{f} = \begin{pmatrix} f_0 \\ \vdots \\ f_{N-1} \end{pmatrix} \in \mathbb{C}^N$ に対して、離散 Fourier 係数の式 $\boldsymbol{C} = \begin{pmatrix} C_0 \\ \vdots \\ C_{N-1} \end{pmatrix} \in \mathbb{C}^N$ を f の離散 Fourier 変換と呼ぶ。そして、

$$W = \frac{1}{N}\left(\omega^{-(n-1)(j-1)}\right) = \frac{1}{N}\begin{pmatrix} \omega^0 & \omega^0 & \cdot & \omega^0 \\ \omega^0 & \omega^1 & \cdots & \omega^{-1 \cdot N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^0 & \omega^{-N-1} & \cdots & \omega^{-(N-1)(N-1)} \end{pmatrix}$$

とおく時、$f = \begin{pmatrix} C_0 \\ \vdots \\ C_{N-1} \end{pmatrix}$ に対して、

$$\boldsymbol{f} = \begin{pmatrix} f_0 \\ \vdots \\ f_{N-1} \end{pmatrix}, \quad \mathcal{C} = \begin{pmatrix} C_0 \\ \vdots \\ C_{N-1} \end{pmatrix}$$

$C_n=\frac{1}{N}\sum_{j=0}^{N-1} f_j\omega^{-nj}$ $(n=0,1,\cdots,N-1)$ が成り立ち、W は正則（注意３）で逆行列は、

$$W^{-1}=\frac{1}{N}\left(\omega^{-(n-1)(j-1)}\right)=\frac{1}{N}\begin{pmatrix}\omega^0 & \omega^0 & \cdot & \omega^0\\ \omega^0 & \omega^1 & \cdots & \omega^{-1\cdot(N-1)}\\ \vdots & \vdots & \ddots & \vdots\\ \omega^0 & \omega^{-(N-1)} & \cdots & \omega^{-(N-1)(N-1)}\end{pmatrix}$$

で表される。つまり、離散 Fourier 逆変換は、

$$W^{-1}\mathcal{C}=\boldsymbol{f}$$

となる。

3.4 畳み込み

3.4.1 畳み込みとは

関数 f,g に対して、f*g を f と g の畳み込み、合成積、convolution と呼ぶ。畳み込みは以下の性質を持つ。

(i) $f*g=g*f$

(ii) $(f*g)*h=f*(g*h)$

(iii) $f_1+f_2)*g=f_1*g+f_2*g$

(iv) $(c*f)*g=c*(f*g)$ $(c\in\mathbb{C})$

(v) $[f\neq 0\wedge\ f*g=f*h]\rightarrow g=h$

次に述べることが重要である。

1 畳み込みの Fourier 変換は、Fourier 変換の積に移る。F[f*g]=定数*Ff*Fg. （定数が何になるかは、畳み込みや Fourier 変換の流儀で異なる。）

2 畳み込みには、単位元の役割をする“ デルタ ”δ がある。$f*\delta=f$. （実変数の関数に対しては、δ はディラックのデルタ関数（デルタ超関数)、また、離散信号に対しては、単位インパルス $\delta=\{\delta_{n0}\}_{n\in\mathbb{Z}}=\{\cdots,0,0,1,0,0,\cdots\}$ である。）

3 デルタ δ の Fourier 変換は定数関数 1 である。また、定数関数 1 の Fourier 変換は δ 。$F\delta=$ 定数 $*1$, $F1=$ 定数 $*\delta$

形式的な定義

$f, g : \mathbb{R} \to \mathbb{C}$ に対して、$f * g : \mathbb{R} \to \mathbb{C}$ を

$$f * g(x) = \int_{-\infty}^{\infty} f(x-y)g(y)dy \quad (x \in \mathbb{R})$$

により定める。周期 2π の周期関数 $f, g : \mathbb{R} \to \mathbb{C}$ に対して、$f * g : \mathbb{R} \to \mathbb{C}$ を

$$f * g(x) = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x-y)g(y)dy \quad (x \in \mathbb{R})$$

により定める。$f, g : Z \to \mathbb{C}$ に対して、$f * g : Z \to \mathbb{C}$ を

$$f * g(n) := \sum_{k=-\infty}^{\infty} f(n-k)g(k) \quad (n \in \mathbb{Z})$$

により定める。周期 N の $f, g : Z \to \mathbb{C}$ に対して、$f * g : Z \to \mathbb{C}$ を

$$f * g(n) := \sum_{k=0}^{N-1} f(n-k)g(k)(n \in \mathbb{Z})$$

により定める。

畳み込みの Fourier 変換は Fourier 変換の積

1 普通の関数の Fourier 変換

f を $\mathbb{R}$ を定義域とする関数 $f : \mathbb{R} \to \mathbb{C}$ とするとき、f の Fourier 変換 $\mathscr{F} f$ は、

$$Ff(\xi) := 1/\sqrt{2}\pi \int_{-\infty}^{\infty} f(x)e^{-i\xi x}dx \quad (\xi \in \mathbb{R})$$

で定義される。$Ff : \mathbb{R} \to \mathbb{C}$ である。このとき、畳み込み $f * g$ について

$$F[f * g](\xi) = \sqrt{2}\pi FfFg$$

上記の等式が成り立つ。

2 周期関数の Fourier 変換

$f : \mathbb{R} \to \mathbb{C}$ を周期 2π の関数とするとき、$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx(n \in \mathbb{Z})$ を f の Fourier 係数と定義したが、これを（周期関数）f の ”Fourier 変換 ” と呼ぶことにして記号 Ff で表すことにする。すなわち

$$Ff(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)e^{-inx}dx(n \in \mathbb{Z})$$

$Ff : Z \to \mathbb{C}$ このとき、畳み込み $f * g$ について

$$F[f * g](\xi) = FfFg$$

上記の等式が成り立つ。

3 周期関数の Fourier 変換

$N \in \mathbb{N}$ に対して、$\omega := e^{2\pi i/N}$ とおく。周期 N の周期数列 $f_{j(}j \in \mathbb{Z})$ に対して、

$$C_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j \omega^{-nj} (n \in \mathbb{Z})$$

を $f_{j(}j \in \mathbb{Z})$ の離散 Fourier 係数と定義したが、これを周期数列の ” Fourier 変換 ” と呼ぶことにする。$f : Z \to \mathbb{C}$ を周期 N の関数（周期 N の周期数列）とするとき、

$$Ff(n) = \frac{1}{N}\sum_{j=0}^{N-1} f_j \omega^{-nj} (n \in \mathbb{Z})$$

$$Ff : Z \to \mathbb{C}$$

で定まる Ff を（周期数列）f の ” Fourier 変換 ” と呼ぶ。周期は N である。このとき、畳み込み f*g について $F[f * g](\xi) = NFfFg$ 上記の等式が成り立つ。

4 数列の Fourier 変換（離散時間 Fourier 変換）

数列 $x_{nn\in\mathbb{Z}}$ に対して、

$$X(\omega) = \sum_{n=-\infty}^{\infty} \quad x_n e^{-in\omega}$$

$$(\omega \in \mathbb{R})$$

を $x_{nn\in\mathbb{Z}}$ の離散時間 Fourier と定義したが、これを数列の ” Fourier 変換 ” と呼ぶことにする。$f : Z \to \mathbb{C}$ に対して、

$$Ff(\xi) := \sum_{n=-\infty}^{\infty} f(n) e^{-in\xi} (\xi \in \mathbb{R})$$

で定まる Ff を数列 f の ” Fourier 変換 ” と呼ぶ。このとき、畳み込み $f * g$ について

$$F[f * g](\xi) = FfFg$$

上記の等式が成り立つ。

（注意）１　台形則 $I=\int_a^b F(t)dt$ に対して、$\{t_j\}_{j=0}^N$ を [a,b] の N 等分点とすると I の近似値を T_N

$$T_N=\sum_{j=0}^{N}(F(t_(j-1))+F(t_j))/2=h'(F(t_0)/2+F(t_j1)+\cdots+F(t_{N-1})+F(t_N)/2)(h'=(b-a)/N)$$

ここで周期は b-a なので $F(t_0)=F(a)=F(b)=F(t_N)$ なので、

$$T_N=h'\sum_{j=0}^{N-1}F(t_j)$$

が導かれる。２　１の N 乗根の性質 $N\in\mathbb{N}$ に対して、

$$\omega=e^{2\pi i/N}$$

とおくとき、次の (1),(2) が成り立つ。

1 ω は原始 N 乗根、すなわち

(i) $1\le m\le N-1\to\omega^m\ne1$

(ii) $\omega^N=1$

を満たす。

2 $\forall m\in\mathbb{Z}$ に対して、次式が成り立つ。

$$\sum_{j=0}^{N-1}?\omega^m j=\blacksquare\,(N(m\equiv0(modN))@0(else))$$

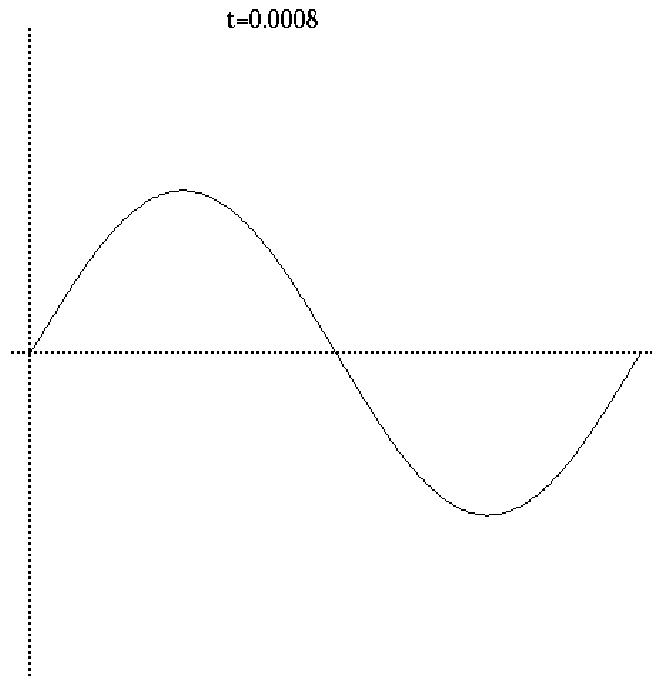
証明 (1) は明らか (2)　$m\equiv0(modN)$ の時、$\omega^m=1$ であるから、任意の j に対して、$\omega^m j=1$　ゆえに、

$$\sum_{j=0}^{N-1}?\omega^m j=\sum_{j=0}^{N-1}?1=N$$

そうでない時、$\omega^m\ne1$ であるから等比数列の和の公式より

$$\sum_{j=0}^{N-1}?\omega^m j=\sum_{j=0}^{N-1}(\omega^m)^j=(1-(\omega^m)^N)/(1-\omega^m)?=(1-(\omega^N)^m)/(1-\omega^m)=0$$

３　これは実際に計算すると $WW^{-1}=I$ となり単位行列になるので W^{-1} は正則であり W も正則である。Fourier 級数を用いたスペクトル法・ここでは周期境界条件の問題に Fourier 級数を用いたスペクトル法を適用する



■

図 3.1: 説明を書く

3.5　線形移流方程式

3.5.1　1 次元線形移流方程式とは

$$\partial u/\partial t + \partial u/\partial x = 0$$

初期条件　$u(x,0) = f(x)$.

境界条件　$u(0,t) = u(2\pi,t)$. 周期境界条件 x,t に対する 1 階の線形偏微分方程式を考える。4.1.2 方程式の説明この方程式は速度（相対速度）1 で伝播することを表す波動方程式である。この方程式は一般的に厳密解が

$$u(x,t) = f(x-t).$$

となることが知られている。これは、時刻 $t > 0$ における u が初期値 f(x) の形を保ちながら t だけ並行移動した形になることを表している。·· ここには図が入る

初期条件と時間発展

3.5.2 スペクトル法を用いて解く

$u(x,t)$ を Fourier 級数を用いて以下のように展開する。

$$u(x,t) = \sum_{k=-N}^{N} (u_k)(t)e^{ikx}$$

N は数値解の解像度（誤差の程度）を決める定数 : 切断波数また展開の自由度（複素数は実部と虚部の 2 つの自由度を持つと数えることにする）は展開係数の自由度（$2N+1$）× 2 だけあるように見えるが、$u(x,t)$ を実数とすると、

$$u?_(- k)^* = u_k$$

の制約がかかる。そのため、自由度はその半分の $2N+1$ になっている。（ここで使われる*は複素共役を表している。）$u(x,t) = \sum_{k=-N}^{N} ??(u_k)(t)e^{ikx}?$ を用いると、偏微分方程式は、

$$du(x,t)/dt = \sum_{k=-N}^{N} ??(d(u_k)?)/dt(t)e^{ikx}?, (\partial u(x,t))/\partial x = \sum_{k=-N}^{N} ??(u_k)?(t)*i*k*e^{ikx}?$$

$$\sum_{k=-N}^{N} ??(d(u_k)?)/dt(t)e^{ikx}? = -\sum_{k=-N}^{N} ??(u_k)?(t) * i * k * e^{ikx}?$$

$$\sum_{k=-N}^{N} ??(d(u_k)?)/dt(t) + ik?(u_k)?(t)e?^i kx? = 0$$

e^{ikx} は直交系なので、この式を満たすためには、係数部分が 0 でなければならないので、ここから以下の常微分方程式が得られる。

$$(d(u_k)?)/dt(t) + ik(u_k)?(t) = 0, (k \in -N, \ldots, N)$$

初期値は、偏微分方程式の初期条件から $u(x,0) = f(x)$ を Fourier 変換することで、

$$(u_k)?(0) = \frac{1}{2\pi} \int_0^{2\pi} f(x)e^{ikx}dx?, (k \in -N, \ldots, N)$$

上記のように得られる。この常微分方程式を解くと、

$$log(u_k)?(t) = ikt + C \text{（任意定数：}C\text{）}$$

$$(u_k)?(t) = Ce^{-ikt}$$

$$(u_k)?(0) = C = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{ikx}dx?, (k \in -N, \ldots, N)$$

よって

$$(u_k)?(t) = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{ikx}dx?e^{-ikt}(k \in -N, \ldots, N)$$

これをはじめの級数に代入すると

$$u(x,t) = \sum_{k=-N}^{N} ??(u_k)?(t)e^{ikx}? = \sum_{k=-N}^{N} ?\frac{1}{2\pi}\int_0^{2\pi} f(x)e^{ikx}dx?e^{-ikt}e^{ikx} = \sum_{k=-N}^{N} ??\frac{1}{2\pi}\int_0^{2\pi} f(x)e^{ikx}dx?e?(i$$

よってスペクトル法によって解が得られた。このスペクトル法によって得られた解は、厳密解と同じ性質を持つ。

3.6 非線形移流方程式

3.6.1 非線形移流方程式とは

一般には非粘性 Burgers 方程式と呼ばれている。

$$\partial u/\partial t + u\partial u/\partial x = 0$$

$$\text{初期条件} \quad u(x,0) = f(x).$$

$$\text{境界条件} \quad u(0,t) = u(2\pi,t).\text{周期境界条件}$$

x, t に対する 1 階の線形偏微分方程式を考える。

3.6.2 方程式の説明

流体運動を支配する方程式に Navier-Stokes の方程式（流体運動を支配する偏微分方程式）がある。そんな Navier-Stokes の方程式を簡単にしたものが Burgers 方程式である。具体的に一次元 Navier-Stokes の方程式は、

$$\partial u/\partial t + u\partial u/\partial x = -1/\rho\,\partial p/\partial x + \nu(\partial^2 u)/(\partial x^2)$$

（ρ (>0) : 流体の密度、$u(>0)$: 流速、$p(>0)$: 圧力、$\nu(>0)$: 粘性係数）と記述できる。この方程式の圧力項と呼ばれる $-1/\rho\,\partial p/\partial x$ を省略したのが Burgers 方程式である。つまり

$$\partial u/\partial t + u\partial u/\partial x = \nu(\partial^2 u)/(\partial x^2)$$

また、今回扱う非粘性 Burgers 方程式は名前の通り粘性がないので上の方程式の粘性係数を 0 と考える。つまり

$$\partial u/\partial t + u\partial u/\partial x = \nu(\partial^2 u)/(\partial x^2)$$

これが非粘性 Burgers 方程式である。非粘性 Burgers 方程式は、Navier-Stokes 方程式の変形であることがわかった。また、非粘性 Burgers 方程式は解析的に考察してみると、n 階微分可能な初期関数に対して、有限時間で方程式の形が崩れてしまい、微分不可能になることが証明できる（今回はこれについての証明は行わない)。ある初期関数を考えると、最も振幅の高い部分が右側にずれていく。

非粘性 Burgers 方程式のイメージ

3.6.3 スペクトル法を用いて解く

はじめに $u(x,t)$ を Fourier 級数を用いて級数展開すると、

$$u(x,t) = \sum_{k=-\infty}^{\infty} (u_k)?(t)e^{ikx}?$$

上記の形で展開される。これを偏微分方程式に代入すると、

$$u(x,t)+u\partial u(x,t)/\partial x = \sum_{k=-\infty}^{\infty} ??((?du?_k)?(t))/dte^{ikx} + \sum_{k=-\infty}^{\infty} ??(u_k)?(t)e^{ikx} \sum_{l=-\infty}^{\infty} (u_l)?(t)?ile?^i kx? = 0 -$$

クロネッカーの $\delta_(m+l,k)$(注意 1) を使うと次のようにかける。

$$\sum_{k=-\infty}^{\infty} ((?du?_k)?(t))/dte^{ikx}? + \sum_{k=-\infty}^{\infty} ? \sum_{l=-\infty}^{\infty} ? \sum_{m=-\infty}^{\infty} il(u_l)?(t)(u_m)?(t)\delta_{m+l,k}e^{ikx}? = 0$$

今回は数値計算を行うので級数を有限区間 $k \in [-N, N]$ にして、計算を行うので上記の級数は、有限項にすると $\delta_(m+l,k)$ の性質から・の中の級数は、$m+l=k$ を満たすものしか残らない。つまり、$[\cdot]$ の中の級数は 1 つになり範囲は $\max\{-N, -N+k\}?l?\min\{N, N+k\}$ となり、

$$\sum_{l=-\infty}^{\infty} ? \sum_{m=-\infty}^{\infty} il(u_l)?(t)(u_m)?(t)\delta_{m+l,k}? \to \sum_{l=-\max\{-N,-N+k\}}^{\min\{N,N+k\}} ??il(u_l)?(t)(u_(k-l))?(t)?$$

これを使うことで、

$$\sum_{k=-N}^{N} ?((?du?_k)?(t))/dt + \sum_{l=-\max\{-N,-N+k\}}^{(\min\{N,N+k\}} ??il(u_l)?(t)(u_(k-l))?(t)?e^{ikx} = 0$$

よってe^{ikx}は固有関数であり一次独立なので、上記の式から次の常微分方程式が得られる。（初期値は初期条件をFourier変換することで得られる。）$({}_{?}(@)^{\forall})k \in [-N, N]$に対して、

■ $$(((?du?_k)?(t))/dt + \sum_{l=-\max\{-N,-N+k\}}^{\min\{N,N+k\}} ??il(u_l)?(t)(u_(k-l))?(t)?@u?_k(0) = 1/\sqrt{2}\int_{-\pi}^{\pi} f(x)?e^{-ikx}dx)$$

この常微分方程式は少なくとも数値的には解ける。（注意２）

3.7 数値的安定性

3.7.1 数値的安定性とは

長時間解を追跡したい場合があるとき（$t \to \infty$となるとき）、刻み幅hを固定して時間を進めたとしても「変なこと」が起こらないようにしたい。

3.7.2 前進Euler法の安定性

1次元常微分方程式

$$dx/dt = \lambda x$$

を考える。

前進Euler法

前進Euler法とは、dx/dtを

$$dx/dt = (x(t+h) - x(t))/h$$

$$x(0) = x_0$$

上記のように前進差分近似する。すると、一次元常微分方程式は、(

$$x(t+h) - x(t))/h = \lambda x$$

$$x(t+h) = x(t) + h\lambda x$$

この式を用いて数値的に解くのが前進 Euler 法である。5.2.2 安定性ステップ幅 h が一定であるとすると、$t = nh$ の時の x の値は、

$$x(t) = x(nh) = (1 + h\lambda)x((n-1)h) = (1 + h\lambda)^2 x((n-2)h) =? = (1 + h\lambda)^n x_0$$

となる。初期値 x_0 に摂度 δ_x が加わった場合、得られる近似値 $x?(t)$ は

$$x?(t) = x?(nh) = (1 + h\lambda)^n ?(x?_0 + \delta_x)$$

となるから、その差は、

$$x?(t) - x(t) = (1 + h\lambda)^n \delta_x$$

で与えられる。この値 $t \to \infty$ すなわち $n \to \infty$ の時に有界におさまる時、その数値計算法は安定であるという。従って、$|1 + h\lambda| < 1$ の時は、$n \to \infty$ では $x?(t) - x(t) \to 0$ となるからこの時前進 Euler 法は安定である。逆に $|1 + h\lambda| > 1$ の時は不安定となる。ここから、安定と不安定の境界から、前進 Euler 法が安定である条件が導かれ、

$$|1 + h\lambda|?1$$

を得る。λ が実数の場合は、

$$-2?\lambda?0$$

となり、時間刻み幅 h が

$$h? - 2/\lambda, (\lambda < 0)$$

の時、安定になることがわかる。このように安定になる条件がある時, 数値解法は条件付き安定（conditionally stable）であるという。つまり、前進 Euler 法は条件付き安定である。

3.7.3 硬い方程式

微分方程式には硬い方程式と呼ばれる方程式があり、これはタイムスケールが違う値が同じ方程式内にあると、「具合が悪いこと」が起きる。

定義

ここでは、硬い方程式の 2 つの定義を紹介する。定義 1 ある安定な微分方程式が解くべき全区間に比較して極めて小さい時定数（注意２）を持つ、指数関数的に減衰する解を 1 つの特解として持つ時、その方程式は硬い方程式であると呼ばれ

る。定義1　1つの問題の中に時定数が大きな部分と小さな部分がある時、その方程式は硬い方程式であると呼ばれる。5.3.2 硬い方程式に問題硬い方程式を数値解法で解く場合、数値的安定性の要請から小さな時定数に合わせて h を選ぶと、なかなか計算が進まない。つまり、複雑な方程式を高速で解きたい場合には、実用的でない。つまり、後退 Euler 法で解くことがしばしば進められる。subsection 後退 Euler 法の安定性 1 次元常微分方程式

$$dx/dt = \lambda x$$

を考える。

3.7.4 後退 Euler 法

後退 Euler 法とは、dx/dt を

$$dx/dt = (x(t) - x(t-h))/h$$

$$x(0) = x_0$$

上記のように後退差分近似する。すると、一次元常微分方程式は、

$$(x(t) - x(t-h))/h = \lambda x(t)$$

$$x(t) = x(t-h) + h\lambda x(t)$$

$$x(t) = 1/(1-h\lambda)x(t-h)$$

この式を用いて数値的に解くのが後退 Euler 法である。

安定性

$t = nh$ における値は

$$x(t) = x(nh) = (1/(1-h\lambda))^n x_0$$

となり、後退 Euler 法が安定になる条件は、

$$|1/(1-h\lambda)|^n?1$$

でなければならないが解が安定となる $Re(\lambda) < 0$ の場合は、任意の $h > 0$ に対して上の式が成り立つ。すなわち、微分方程式が安定（解を持つならば）であれば、後退 Euler 法は常に安定である。このような方法を無条件安定 (unconditionally stable) であるという。

後退 Euler 法の抱える問題

前進 Euler 法の数値的安定性と後退 Euler 法の数値的安定性の話から後退 Euler 法の法が優れているように感じる。しかし、後退 Euler 法には問題が 2 つある。方程式を解く必要がある。他の数値計算法に比べて制度があまり良くないについて、簡単な常微分方程式なら簡単に後退 Euler 法の式を導くことが簡単だが複雑な式になると導けないこともあるために使える式が限られている。についてそもそも Euler 法の制度は他の数値計算法と比べて制度が低い。

第4章　線形・非線形移流方程式のプログラム

4.1　FFT

FFTとは、高速Fourier変換の略称であり、離散 Fourier変換（以降からDFTと略す）を非常に効率よく計算するためのアルゴリズムである。手短にいうと、

(1)　周期関数の、1周期区間の等分割点上での関数値からなる数列に、その関数のFourier係数の近似値[1]を対応させる写像

(2) 周期関数のFourier係数に、その関数の1周期区間の等分割点上での関数値[2]からなる数列を対応させる写像

今回は、数値計算でFFTを使うために、ライブラリー「FFTPack」を用いて数値計算を行なった。

4.2　線形移流方程式の数値計算

4.2.1　実数値DFT

線形移流方程式を数値計算をするためにDFTを使い、数値計算を行なったのでここでは、実数値DFTの解説を行う。fが実数値関数

$$c_{+k} = \overline{c_k}$$

$$C_{+k} = \overline{C_k}$$

$$k \in \mathbb{Z}$$

[1]Fourier係数を定義式の定積分を台形則で近似したもの

[2]一般には無限級数になるFourier係数を、有限項までしか計算してない。といういう意味では近似値あるが、有限Fourier級数の計算としては、丸め誤差がない限り正確な値が得られる方法である。

これは、実数値関数の Fourier 係数の Hermite 対称性といい、冗長性（余分なものが現れること）が見られる。

復習

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}((a_k)\cos kx + b_k \sin kx)$$

と Fourier 級数展開される。ここで、

$$a_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos kxdx \quad (k = 0, 1, \cdots)$$

$$b_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin kxdx \quad (k = 0, 1, \cdots)$$

ここから

$$\begin{aligned} \hat{f}(k) &= \frac{1}{2}(a_k - \imath b_k) \quad (k \in \mathbb{N}), \\ \hat{f}(0) &= \frac{1}{2}a_0. \end{aligned} \tag{4.1}$$

a_k の近似値を A_k、b_k の近似値を B_k として、

台形則を使うと、

$$\begin{aligned} A_k &= \frac{1}{\pi}\sum_{j=0}^{N-1} f_j \cos k - \frac{2\pi}{N}j * \frac{2\pi}{N}, \\ &= \frac{2}{N}\sum_{j=o}^{N-1} f_j \cos\frac{2\pi kj}{N}. \\ B_k &= \frac{1}{\pi}\sum_{j=0}^{N-1} f_j \sin k - \frac{2\pi}{N}j * \frac{2\pi}{N}, \\ &= \frac{2}{N}\sum_{j=o}^{N-1} f_j \sin\frac{2\pi kj}{N} \end{aligned} \tag{4.2}$$

となる。ここで次の性質は保たれている。

1.
$$C_k = \frac{1}{2}(A_k - \imath B_k) \quad (k \in \mathbb{N}), \quad C_0 = \frac{1}{2}A_0$$

2.
$$k \equiv l \quad (\mod \mathbb{N}) \quad A_k = A_l, \quad B_k = B_l$$

3.

$$A_{N-k} = A_k, \quad B_{N-k} = -B_k, \quad N \text{ が偶数} \Rightarrow B_{\frac{N}{2}} = 0$$

N が偶数なのか奇数なのかで場合分けする。N が偶数の場合

(4.3)

写像： $(f_0, f_1, \cdots, f_{N-1}) \to (A_0, A_1, B_1, A_2, B_2, \cdots, A_{\frac{N}{2}-1}, B_{\frac{N}{2}-1}, A_{\frac{N}{2}},) \quad (B_{\frac{N}{2}} = 0)$

は $\mathbb{R}^N \to \mathbb{R}^N$ への全単射で逆写像は、

$$f_j = \frac{A_0}{2} + 2\sum_{k=1}^{\frac{N}{2}-1}(A_k \cos\frac{2\pi k}{N}j + B_k \sin\frac{2\pi k}{N}j) + A_{\frac{N}{2}}(-1)^j, \quad (j = 0, 1, 2, \cdots, N-1). \tag{4.4}$$

N が奇数の場合

$$\text{写像：} \quad (f_0, f_1, \cdots, f_{N-1}) \to (A_0, A_1, B_1, A_2, B_2, \cdots, A_{\frac{N-1}{2}}, B_{fracN-12}) \tag{4.5}$$

は $\mathbb{R}^N \to \mathbb{R}^N$ への全単射で逆写像は、

$$f_j = \frac{A_0}{2} + 2\sum_{k=1}^{\frac{N}{2}-1}(A_k \cos\frac{2\pi k}{N}j + B_k \sin\frac{2\pi k}{N}j), \quad (j = 0, 1, 2, \cdots, N-1). \tag{4.6}$$

で与えらる。これを実数値 DFT と呼ぶ。

4.2.2 前進 Euler 法によるプログラム

プログラム

```
/*
 * tstdzfft-f-euler.c -- test of dzffti(), dzfftf(), dzfftb() in FFTPACK
 *   How to compile: ccmg tstdzfft-f-euler-ad.c get_id.c -ldfftpack
 */

#include <stdio.h>
#include <math.h>
#include <glsc.h>
#include <dfftpack.h>

#define MAXN    4096
#define HMAXN   (MAXN/2)
```

```
double f(double);

int main(void)
{
    /* ユーザーが入力する値 */
    int N;
    double Tmax, tau;
    /* 定数 */
    double PI;
    /* プログラムの実行を通じて不変な値を持つ変数，関数宣言 */
    int halfN, maxnstep, getint();
    double h, f(), getdouble();
    /* 制御変数，作業用配列 */
    int j, k, nstep;
    double t;
    double r[MAXN+1], a[HMAXN+1], b[HMAXN+1], work[3*MAXN+15];
    double at[HMAXN+1], bt[HMAXN+1];

    /* 定数値の設定 */
    PI = 4.0 * atan(1.0);
    /* 分割数 N，追跡時間 Tmax，時間刻み幅 τ の決定 */
    printf("N=");    N    = getint();    if (N <= 0 || N > MAXN) return 0;
    printf("Tmax="); Tmax = getdouble();
    printf("τ=");   tau  = getdouble(); if (tau == 0.0) return 0;
    /* プログラムの実行を通じて不変な値を持つ変数 */
    halfN = N / 2; maxnstep = Tmax / tau; h = 2.0 * PI / N;
    if (maxnstep < 0) {
        fprintf(stderr, "Tmax, τの値を見直して下さい\n");
        return 0;
    }
    /* 初期値 */
    for (j = 0; j < N; j++) r[j] = f(j * h);
    /* N 項実 FFT の準備の後、順方向 DFT によって Fourier 係数を求める */
    dzffti(N, work);
    dzfftf(N, r, &a[0], a+1, b+1, work);
    /* Fourier 係数の最初の数項を表示 */
    printf("a[0]/2=%15f\n", a[0]);
    for (k = 1; k < 5; k++)
        printf("a[%d]  =%15f, b[%d]  =%15f\n", k, a[k], k, b[k]);
    at[0] = a[0];
    for (k = 1; k <= halfN; k++) {
    at[k] = a[k];
    bt[k] = b[k];
    }
    /* ウィンドウを開き、適当に座標を入れる */
    g_init("DZFFTF", 140.0, 140.0);
    g_device(G_BOTH);
    g_def_scale(0,
```

```
                -0.2, 6.48, -10.0, 10.0,
                10.0, 10.0, 120.0, 120.0);
    g_sel_scale(0);
    /* 座標軸を描く */
    g_def_line(0, G_BLACK, 2, G_LINE_DOTS);
    g_def_line(1, G_BLACK, 1, G_LINE_SOLID);
    g_sel_line(0);
    g_move(-0.2, 0.0); g_plot(6.48, 0.0);
    g_move(0.0, -10.0); g_plot(0.0, 10.0);
    g_sel_line(1);
    /* */
    for (nstep = 0; nstep <= maxnstep; nstep++) {
        t = nstep * tau;
        /* u(・,t) の Fourier 係数を求める */
        at[0] = a[0];
        for (k = 1; k <= halfN; k++) {
            at[k] = at[k] + tau * (- k * bt[k]);
            bt[k] = bt[k] + tau * ( k * at[k]);
        }
        /* 逆実 DFT により、関数値を求める */
        dzfftb(N, r, &at[0], at+1, bt+1, work);
        /* グラフを描く */
        r[N] = r[0];
        g_cls();
        g_move(0.0, r[0]);
        for (j = 1; j <= N; j++) g_plot(j * h, r[j]);
    }
    g_sleep(G_STOP);
    return 0;
}

double f(double x)
{
    /* a0/2+(a1 cos(x)+b1 sin(x))+...+(a3 cos(3x)+b3 sin(3x)) */
    int i;
    static double a[4] = {1.0, 2.0, 0.0, 4.0};
    static double b[4] = {0.0, 0.0, 3.0, 5.0};
    double result = a[0] / 2.0;
    for (i = 1; i < 4; i++)
        result += a[i] * cos(i * x) + b[i] * sin(i * x);
    return result;
}
```

4.2.3 後退 Euler 法によるプログラム

プログラム

```
/*
```

```
 * tstzfft.c -- test of zffti(), zfftf(), zfftb() in DFFTPACK
 *   How to compile: ccmg tstzfft-b-euler-ad2.c -ldfftpack
 */

#include <stdio.h>
#include <complex.h>
#include <math.h>
#include <glsc.h>
#include <dfftpack.h>

#define MAXN    (4096)

double f(double);

int main(void)
{
    /* ユーザーが入力する値 */
    int N, m;
    double Tmax, tau;
    /* 定数 */
    double PI;
    /* プログラムの実行を通じて不変な値を持つ変数，関数宣言 */
    int halfN, maxnstep, getint();
    double h, f(), getdouble();
    /* 制御変数，作業用配列 */
    int j, k, nstep;
    double t;
    double work[4*MAXN+15];
    double complex c[MAXN+1], ct[MAXN+1], u[MAXN+1];

    /* 定数値の設定 */
    PI = 4.0 * atan(1.0);
    /* 分割数 N，追跡時間 Tmax，時間刻み幅 τ の決定 */
    printf("m="); scanf("%d", &m); if (m <= 0 || m > MAXN) return 0;
    printf("Tmax="); scanf("%lf", &Tmax);
    printf("τ=");    scanf("%lf", &tau); if (tau == 0.0) return 0;
    /* プログラムの実行を通じて不変な値を持つ変数 */
    N = 2*m+1; halfN = N / 2; maxnstep = Tmax / tau; h = 2.0 * PI / N;
    if (maxnstep < 0) {
        fprintf(stderr, "Tmax, τの値を見直して下さい\n");
        return 0;
    }

    //N = 2*m+1;
    printf("%d\n", N);
    /* 初期値 (実数値なので c[j] の虚部は 0) */
    for (j = 0; j < N; j++) c[j] = f(j * h);
    /* N 項実 FFT の準備の後、順方向 DFT によって Fourier 係数を求める */
    zffti(N, work);
```

```
zfftf(N, c, work);
/* 「画像処理とフーリエ変換」の C_k を求めるには N で割る必要がある。 */
for (k = 0; k < N; k++)
c[k] /= N;
for(k = 0; k < N; k++) {
    ct[k] = c[k];
    }
/* Fourier 係数の最初と最後の数項を表示 */
printf("\n");
for (k = 0; k < 10; k++)
    printf("c[%d]  =(%15f, %15f)\n", k, creal(c[k]), cimag(c[k]));
for (k = N - 10; k < N; k++)
    printf("c[%d]  =(%15f, %15f)\n", k, creal(c[k]), cimag(c[k]));
/* ウィンドウを開き、適当に座標を入れる */
g_init("ZFFT", 140.0, 140.0);
g_device(G_BOTH);
g_def_scale(0,
    -0.2, 6.48, -10.0, 10.0,
    10.0, 10.0, 120.0, 120.0);
g_sel_scale(0);
/* 座標軸を描く */
g_def_line(0, G_BLACK, 2, G_LINE_DOTS);
g_def_line(1, G_BLACK, 1, G_LINE_SOLID);
g_sel_line(0);
g_move(-0.2, 0.0); g_plot(6.48, 0.0);
g_move(0.0, -10.0); g_plot(0.0, 10.0);
g_sel_line(1);
/* */
for (nstep = 0; nstep <= maxnstep; nstep++) {
    t = nstep * tau;
    /* u(・,t) の Fourier 係数を求める */
    for (k = 0; k <= m; k++) {
        double complex factor =  1 / (1 +  k * k * tau * tau);
        ct[k] = (1 - I * k * tau) * factor * ct[k];
    }
    for (k = -m; k < 0; k++) {
        double complex factor =  1/(1 +  k * k * tau * tau);
        ct[N + k] = (1 - I * k * tau ) * factor * ct[N + k] ;
        }

    for (k = 0; k < N; k++) {
        u[k] = ct[k];
    }
    /* 共役 DFT により、関数値を求める */
    zfftb(N, u, work);

    /* グラフを描く */
    u[N] = u[0];
    //if(nstep % 50==0) {
```

```
        g_cls();
        g_move(0.0, creal(u[0]));
        for (j = 1; j <= N; j++) g_plot(j * h, creal(u[j]));
            /* 座標軸を描く */
    g_def_line(0, G_BLACK, 2, G_LINE_DOTS);
    g_def_line(1, G_BLACK, 1, G_LINE_SOLID);
    g_sel_line(0);
    g_move(-0.2, 0.0); g_plot(6.48, 0.0);
    g_move(0.0, -10.0); g_plot(0.0, 10.0);
    g_sel_line(1);

            //g_sleep(G_STOP);
            //}
    }
    g_sleep(G_STOP);
    return 0;
}

double f(double x)
{
    /* a0/2+(a1 cos(x)+b1 sin(x))+...+(a3 cos(3x)+b3 sin(3x)) */
    int i;
    static double a[4] = {1.0, 2.0, 0.0, 4.0};
    static double b[4] = {0.0, 0.0, 3.0, 5.0};
    double result = a[0] / 2.0;
    for (i = 1; i < 4; i++)
        result += a[i] * cos(i * x) + b[i] * sin(i * x);
    return result;
}
```

4.3 非線形移流方程式の数値計算

4.3.1 変換法

スペクトル法で得られらた
常微分方程式

$$\frac{d\hat{u}_m}{dt} + \sum_{l=\max(-N,-N+m)}^{\min(N,N+m)} \imath l \hat{u}_{m-l}\hat{u}_l = 0 (m = -N, \cdots, -1, 0, 1, \cdots, N) \tag{4.7}$$

は数値計算するにあたって、コストが高くなっている。実際に計算量を見積もってみると、非線形移流方程式の左辺第 2 項の計算は各 m について $2N+1$ 個の総和が必要であり、また、l の総和 $2N+1$ と合わせて $O((2N+1)^2)$ となっている。

一方で差分法では評価に必要な計算コストは $O(2N+1)$ となっている。[3]
すなわち、このままでは、スペクトル法は、コストが高いため実用的ではない。

$$\Downarrow$$

ここで変換法を用いる計算コストを減らす。

考え方

非線形項と呼ばれる $u(x,t)\frac{\partial u(x,t)}{\partial x}$ を Fourier 級数（$u(x,t)=\sum_{k=-N}^{N}\hat{u}_k(t)e^{ikx}$）を用いることで、

$$\sum_{l=\max(-N,-N+m)}^{\min(N,N+m)} \imath l \hat{u}_{m-l}\hat{u}_l$$

$$(m=-N,\cdots,-1,0,1,\cdots,N)$$

が得られらたが、これは数列の畳み込みになっている。畳み込みの Fourier 変換は、２つの Fourier 変換の積であるので（$\mathscr{F}[f*g]=\mathscr{F}f\mathscr{F}g$）、それを逆 Fourier 変換すれば、畳み込みの計算ができる（$f*g=\mathscr{F}^{-1}[\mathscr{F}f\mathscr{F}g]$）。 つまり、非線形項を

$$\hat{u}_k(t) \Longrightarrow_{\text{逆 Fourier 変換}} u(x,t)$$

$$\imath k\hat{u}_k(t) \Longrightarrow_{\text{逆 Fourier 変換}} \frac{\partial u(x,t)}{\partial x}$$

を用いることで、

$$\hat{u}_k(t)\imath k\hat{u}_k(t) \Longrightarrow_{\text{逆 Fourier 変換}} u(x,t)\frac{\partial u(x,t)}{\partial x}$$

これが数列の畳み込みの計算に相当し、これをさらに Fourier 変換を行うことで、

$$u(x,t)\frac{\partial u(x,t)}{\partial x} \Longrightarrow_{\text{逆 Fourier 変換}} (\hat{u}\frac{\partial \hat{u}}{\partial x})_k(t)$$

よって、$u(x,t)\frac{\partial u(x,t)}{\partial x}$ の Fourier 係数が分かり、非線形移流方程式も

$$\frac{\partial u(x,t)}{\partial t}+u(x,t)\frac{\partial u(x,t)}{\partial x}=0 \Rightarrow \sum_{k=-N}^{N}\frac{d\hat{u}_k}{dt}(t)e^{\imath kx}+\sum_{k=-N}^{N}(\hat{u}\frac{\partial \hat{u}}{\partial x})_k(t)e^{\imath kx}=0$$

ここから、次の常微分方程式が得られる。

[3]中心差分法 ($\frac{\partial u}{\partial x}\simeq\frac{u_{j+1}-u_{j-1}}{2\Delta x}$) を使い、非線形移流方程式 $\frac{du_j}{dt}+u_j\frac{u_{j+1}-u_{j-1}}{2\Delta x}=0$ と差分化することで得られた式を用いて計算コストを評価した。

$$
\begin{aligned}
\frac{d\hat{u}_k(t)}{dt} &= -(\hat{u}\frac{\partial \hat{u}}{\partial x})_k(t), \quad (k \in (-N, \cdots, N)), \\
\hat{u}_k(0) &= \frac{1}{\sqrt{2\pi}} \int_{-\pi}^{\pi} f(x) e^{\imath kx} dx
\end{aligned} \tag{4.8}
$$

4.3.2 変換法なしのプログラム

```
/*
 * tstzfft.c -- test of zffti(), zfftf(), zfftb() in DFFTPACK
 *   How to compile: ccmg tstzfft-f-euler-ad.c get_id.c -ldfftpack
 */

#include <stdio.h>
#include <complex.h>
#include <math.h>
#include <glsc.h>
#include <dfftpack.h>

#define MAXN    (4096)

#define CT(k) ct[ (k>=0)?(k) :N+(k) ]
#define NEWCT(k) newct[ (k>=0)?(k) :N+(k) ]

double f(double);

double maxv(int n, double complex *x) {
    int k;
    double vmax = cabs(x[0]);
    for ( k = 1; k < n; k++){
        // printf("%g %g\n", vmax, cabs(x[k]));
        if(cabs(x[k]) > vmax){
            vmax = cabs(x[k]);
        }
    }
    return vmax;
}

double norm1(int n, double complex *x) {
    int k;
    double sum = cabs(x[0]);
    for (k = 1; k < n; k++)
         sum += cabs(x[k]);
    return sum;
}
```

```
int main(void)
{
    /* ユーザーが入力する値 */
    int N ;
    int m;
    double Tmax, tau;
    /* 定数 */
    double PI;
    /* プログラムの実行を通じて不変な値を持つ変数，関数宣言 */
    int halfN, maxnstep, getint();
    double h, f(), getdouble();
    /* 制御変数，作業用配列 */
    int j, k,l, nstep;
    double t;
    double work[4*MAXN+15];
    double complex c[MAXN+1], ct[MAXN+1], u[MAXN+1], newct[MAXN+1];
    char message[100];
    /* 定数値の設定 */
    PI = 4.0 * atan(1.0);
    /* 分割数 N，追跡時間 Tmax，時間刻み幅 τ の決定 */
    printf("m=");    m    = getint();    if (m <= 0 || m > MAXN) return 0;
    printf("Tmax="); Tmax = getdouble();
    printf("τ=");   tau  = getdouble(); if (tau == 0.0) return 0;
    /* プログラムの実行を通じて不変な値を持つ変数 */
    N = 2*m+1;
    halfN = N / 2; maxnstep = Tmax / tau; h = 2.0 * PI / N ;
    if (maxnstep < 0) {
        fprintf(stderr, "Tmax, τの値を見直して下さい\n");
        return 0;
    }

    /* 初期値 (実数値なので c[j] の虚部は 0) */
    for (j = 0; j < N; j++) c[j] = f(j * h);
    /* N 項実 FFT の準備の後、順方向 DFT によって Fourier 係数を求める */
    zffti(N, work);
    zfftf(N, c, work);
    /* 「画像処理とフーリエ変換」の C_k を求めるには N で割る必要がある。 */
    for (k = 0; k < N; k++)
    c[k] /= N;
    /* Fourier 係数の最初と最後の数項を表示 */
    printf("\n");
    printf ("N=%d", N);
    printf("\n");
    for (k = 0; k < 10; k++)
        printf("c[%d]  =(%15f, %15f)\n", k, creal(c[k]), cimag(c[k]));
    for (k = N - 10; k < N; k++)
    printf("c[%d]  =(%15f, %15f)\n", k, creal(c[k]), cimag(c[k]));
    for(k = 0; k <= N-1; k++) {
        ct[k] = c[k];
```

```
}
/*for (k =-m; k <=m; k++) {
    CT(k) = ct[m+k];
}*/
for (k = 0; k < 10; k++)
printf("CT[%d]  =(%15f, %15f)\n", k, creal(CT(k)), cimag(CT(k)));
for (k = -m; k < -m+10; k++)
printf("CT[%d]  =(%15f, %15f)\n", k, creal(CT(k)), cimag(CT(k)));
/* ウィンドウを開き、適当に座標を入れる */
g_init("ZFFT", 140.0, 140.0);
g_device(G_BOTH);
g_def_scale(0,
    -0.2, 6.48, -2.0, 2.0,
    10.0, 10.0, 120.0, 120.0);
g_sel_scale(0);
/* 座標軸を描く */
g_def_line(0, G_BLACK, 2, G_LINE_DOTS);
g_def_line(1, G_BLACK, 1, G_LINE_SOLID);
g_sel_line(0);
g_move(-0.2, 0.0); g_plot(6.48, 0.0);
g_move(0.0, -10.0); g_plot(0.0, 10.0);
g_sel_line(1);
/* */
for (nstep = 0; nstep <= maxnstep; nstep++) {
    t = nstep * tau;
    /* u(・,t) の Fourier 係数を求める */
   // ct[0] = c[0];
    /*for (k = 1; k <= halfN; k++) {
        double complex factor = cos( k * t ) - I * sin( k *t ); cexp( - k * I * t);
        ct[k] = c[k] * factor;
        ct[N - k] = c[ N - k ] * conj(factor);
    }*/
    for (k = 0; k < N; k++) {
        newct[k] = ct[k];
        }

    for (k = -m; k <= m; k++) {
        double complex factor = 0;
        if(k < 0){
        for (l = -m; l <= k + m; l++) {
             factor = factor + I * l * CT(l) * CT(k - l);
            }
        NEWCT(k) = CT(k) - tau * factor;
        }else{
        for (l = -m+k; l <= m ; l++) {
             factor = factor + I * l * CT(l) * CT(k - l);
            }
        NEWCT(k) = CT(k) - tau * factor;
            }
```

```
        }

        for (k = 0; k < N; k++) {
            ct[k] = newct[k];
            }
        for (k = 0; k < N; k++){
            u[k] = ct[k];
        }
        /* 共役 DFT により、関数値を求める */
        zfftb(N, u, work);
        /* グラフを描く */
        u[N] = u[0];
        g_cls();
        sprintf(message, "t=%g", t);
        g_move(0.0, creal(u[0]));
        for (j = 1; j <= N; j++) g_plot(j * h, creal(u[j]));
        g_text(50.0, 10.0, message);
        g_sel_line(0);
        g_move(-0.2, 0.0); g_plot(6.48, 0.0);
        g_move(0.0, -2.0); g_plot(0.0, 2.0);
        g_sel_line(1);
        /*u の最大値*/
        printf("1 norm of=%g\n", norm1(N-1, ct+1));
        //g_sleep(G_STOP);
    }
    g_sleep(G_STOP);
    return 0;
}

double f(double x)
{
    /* a0/2+(a1 cos(x)+b1 sin(x))+...+(a3 cos(3x)+b3 sin(3x)) */
    int i;
    // static double a[4] = {2.0, 0.0, 4.0/10, 0.0};
    // static double b[4] = {0.0, 5.0/10, 2.0/10, 0.0};
    static double a[4] = {0.0, 0.0, 0.0, 0.0};
    static double b[4] = {0.0, 1.0, 0.0, 0.0};
    double result = a[0] / 2.0;
    for (i = 1; i < 4; i++)
        result += a[i] * cos(i * x) + b[i] * sin(i * x);
    return result;
}
```

4.3.3 変換法ありのプログラム

```
/*
 * tstzfft.c -- test of zffti(), zfftf(), zfftb() in DFFTPACK
 *   How to compile: ccmg tstzfft-f-euler-ad-tr2.c get_id.c -ldfftpack
```

```
 */

#include <stdio.h>
#include <complex.h>
#include <math.h>
#include <glsc.h>
#include<stdlib.h>
#include<time.h>
#include <dfftpack.h>
#include<string.h>

#define MAXN    (4096)

double f(double);

#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>

void g_dump(char *fname, Display *display, Window wid);

int msleep(int ms)
{
    struct timeval timeout;
    timeout.tv_sec = ms / 1000;
    timeout.tv_usec = (ms % 1000) * 1000;
    if (select(0, (fd_set *) 0, (fd_set *) 0, (fd_set *) 0, &timeout) < 0) {
        perror("usleep");
        return -1;
    }
    return 0;
}

int main(void)
{
    //printf("%f",L);
    Display *display;
    Window window;
    char filename[256];

    /* ユーザーが入力する値 */
    int N, m;
    double Tmax, tau;
    /* 定数 */
    double PI;
    /* プログラムの実行を通じて不変な値を持つ変数, 関数宣言 */
    int halfN, maxnstep, getint();
    double h, f(), getdouble();
    /* 制御変数, 作業用配列 */
```

```
int j, k, nstep, skip;
double t;
char message[100];
double work[4*MAXN+15];
double complex c[MAXN+1], ct[MAXN+1], u[MAXN+1], du[MAXN+1], udu[MAXN+1];

/* 定数値の設定 */
PI = 4.0 * atan(1.0);
/* 分割数 N=2*m+1, 追跡時間 Tmax, 時間刻み幅 τ の決定 */
printf("m=");    m    = getint();    if (m <= 0 || m > MAXN) return 0;
printf("Tmax="); Tmax = getdouble();
printf("τ=");   tau  = getdouble(); if (tau == 0.0) return 0;
printf("skip=");  skip = getint();    if(skip >= Tmax/ tau) return 0;
/* プログラムの実行を通じて不変な値を持つ変数 */
N = 2*m + 1;
halfN = N / 2; maxnstep = Tmax / tau; h = 2.0 * PI / N;
if (maxnstep < 0) {
    fprintf(stderr, "Tmax, τ, skip のいずれかの値を見直して下さい\n");
    return 0;
}

printf("N=%d\n", N);

/* 初期値 (実数値なので c[j] の虚部は 0) */
for (j = 0; j < N; j++) u[j] = c[j] = f(j * h);
/* N 項実 FFT の準備の後、順方向 DFT によって Fourier 係数を求める */
zffti(N, work);
zfftf(N, c, work); //ここでは c はフーリエ係数
for (k = 0; k < N; k++)
    /*u[ k ]  =*/ c[ k ] = c[k] / N;
/* Fourier 係数の最初と最後の数項を表示 */
printf("\n");
for (k = 0; k < 10; k++)
    printf("c[%d]  =(%15f, %15f)\n", k, creal(c[k]), cimag(c[k]));
for (k = N - 10; k < N; k++)
    printf("c[%d]  =(%15f, %15f)\n", k, creal(c[k]), cimag(c[k]));
/* ウィンドウを開き、適当に座標を入れる */

display = g_get_display();
window = g_get_window();

g_init("ZFFT", 140.0, 140.0);
g_device(G_BOTH);
g_def_scale(0,
    -0.2, 6.48, -2.0, 2.0,
    10.0, 10.0, 120.0, 120.0);
g_sel_scale(0);
/* 座標軸を描く */
g_def_line(0, G_BLACK, 2, G_LINE_DOTS);
```

```
    g_def_line(1, G_BLACK, 1, G_LINE_SOLID);
    g_sel_line(0);
    g_move(-0.2, 0.0); g_plot(6.48, 0.0);
    g_move(0.0, -2.0); g_plot(0.0, 2.0);
    g_sel_line(1);
    /* */
    for (nstep = 0; nstep <= maxnstep; nstep++) {
        t = nstep * tau;

        /*ここで∂ u/∂ x=du[] とする*/
    for(k = -m; k < 0; k++) {

            du[N + k] = I * k * c[N + k]; //udu がフーリエ係数になった
        }
    for(k = 0; k <= m; k++) {
            du[k] = I * k * c[k];//udu がフーリエ係数になった
        }
    /*ここで u と du を逆変換*/
    //zfftb(N, u, work); //最初：ここでは u は関数値　２回目以降：関数値だった u をさら
に関数値に変換した (@1 がない時)   or しっかりと関数値になった (@1 がある時)
    zfftb(N, du, work); //最初：ここでは du は関数値　２回目以降：しっかり関数値になっ
てる
    /*分点である j について 0〜N-1 までの u*∂ u/∂ x を求める*/
    for(j = 0; j < N; j++) {
        udu[j] = u[j] * du[j];//最初：関数値　２回目以降：関数値
    }
    /*最後に u*∂ u/∂ x のフーリエ係数を求める*/
    zfftf(N, udu, work); //最初：du はフーリエ係数　２回目以降：フーリエ係数
        for (k = 0; k < N; k++) {
            udu[k] /= N;
        }
        /* u(・,t) の Fourier 係数を求める */
        for (k = 0; k <= m; k++) {
            c[k] = c[k] - tau* udu[k];
        }
        for (k = -m; k < 0; k++) {
            c[N + k] = c[ N + k ] - tau * udu[N + k];
        }

        for(k = 0; k <= N-1; k++){
            u[k] = c[k];//最初：ここでは u[] は c[] のフーリエ係数の値
        }
        /* 共役 DFT により、関数値を求める */
        zfftb(N, u, work);//最初：ここ u は関数値になった
        if(nstep % skip == 0){
        /*nstep が 50 回に 1 回描写される*/
        /* グラフを描く */
        u[N] = u[0];
        g_cls();
```

```
        sprintf(message, "t=%g", t);
        g_move(0.0, creal(u[0]));
        for (j = 1; j <= N; j++) g_plot(j * h, creal(u[j]));
        g_text(50.0, 10.0, message);
        g_sel_line(0);
        g_move(-0.2, 0.0); g_plot(6.48, 0.0);
        g_move(0.0, -2.0); g_plot(0.0, 2.0);
        g_sel_line(1);
        g_move(0.0, creal(u[0]));
            for (j = 1; j <= N; j++) g_plot(j * h, creal(u[j]));
            sprintf(filename,"anim%03d.png",k);
        g_dump(filename,display,window);
        }
        //g_sleep(G_STOP);
        //msleep(10);
        /*関数値になった u をフーリエ係数に戻す*/
        /*for(k = 0; k <= N-1; k++){
            u[k] = c[k];//最初：ここでは u[] は c[] のフーリエ係数の値-@1
        }*/

    }
    //保存するため
    //sprintf(filename,"anim%03d.jpg",k);
        //g_dump(filename,display,window);
    g_sleep(G_STOP);
    return 0;
}

double f(double x)
{
    /* a0/2+(a1 cos(x)+b1 sin(x))+...+(a3 cos(3x)+b3 sin(3x)) */
    int i;
    static double a[4] = {0.0, 0.0, 0.0, 0.0};
    static double b[4] = {0.0, 1.0, 0.0, 0.0};
    double result = a[0] / 2.0;
    for (i = 1; i < 4; i++)
        result += a[i] * cos(i * x) + b[i] * sin(i * x);
    return result;
}

//画像を保存する
void g_dump(char *fname, Display *display, Window wid)
{
    char command[256];

    sprintf(command, "import -silent -window %lu %s", wid, fname);
    system(command);
}
```

4.3.4　変換法のあり・なしの比較

今回行った数値計算では、精度には差がなかったが tau を大きくした時には、計算の速さに大きな差が出てきたため、確かに変換法による計算コストの削減が見られた。

第5章　まとめ

　スペクトル法は空間を有限個の端数で展開した。解 u に対して展開するとき滑らかな直交関数系を用いて行った。滑らかな関数系を用いているので空間の離散化の時の数値的分散性が起こらず、差分法に比べて高精度なかいが得られた。
1 次元線形移流方程式を解く時は Fourier 級数を用いて展開して、波数空間で時間 t に対して積分することで非常に簡単な方程式に帰着することができた。 1 次元非線形移流方程式を解く時には、非線形項をうまく評価する必要があった。 1 次元線形移流方程式と同じように非線形項を計算してしまうと波数空間では計算コストがかかってしまた。そこで変換法を用いることで非線形項のみを実空間で評価することで計算コストを $o\ (N^2) \rightarrow\ o\ (N \log_2 N)$ へと抑えることができ、ある程度の低コストで計算することができた。そして、数値計算を行う際には、高速 Fourier 変換を用いることでも計算コストを削減することができた。　今後の課題としては、球面調和関数を用いる場合の偏微分方程式を理解し、プログラムを作ることである。本レポートで扱った偏微分方程式は、線形・非線形移流方程式であり、Fourier 級数を用いたスペクトル法を理解することができた。スペクトル法を理解することにおいては、本レポートは非常に意義のあるものであったが実際に惑星気象現象を扱うためにも今後は球面調和関数を用いた場合も理解し、プログラムを作りたい。

関連図書

[1] 石岡, スペクトル法, 東京大学出版会 (2004).

[2] http://takeno.iee.niit.ac.jp/~foo/thesis/2000/nabe-2.pdf