Last modified: 2025 年 9 月 10 日

1　数値モデルを使うための数値計算法

1.1　誤差

数値計算は、計算機を用いて数値的な計算を行う。計算機は有限精度で数値を表現するため、計算結果には誤差が生じる。実際、数値シミュレーションは、「数値実験」と呼ばれることもある。これは、数値計算が実験と同様の位置づけにあることを示している。実験においては、測定器の精度や測定誤差が問題となるが、数値計算においては計算機の有限精度による誤差が問題となる。

まず、最初に誤差の定義を行う。ある量 x の測定値 a に見込まれる誤差を $\Delta a(>0)$ であるとする。このとき、x の値は $a \pm \Delta a$ の範囲にあると考える。Δa の値は、x の値がこの範囲にあると考えられる範囲である。

$$x \in (a - \Delta a) \tag{1}$$

もしくは、$x = a \pm \Delta a$ のように略記を用いて記述される。ここで、2 つの量 x と y を考える。

$$x = a \pm \Delta a,\ y = b \pm \Delta b \tag{2}$$

このとき、x と y の関数として計算される量 z は、量 c と誤差 Δc を持つ。

$$z = c \pm \Delta c \tag{3}$$

量 z は、観測された 2 つの量 x と y の関数 f として計算されるとすると、量 c と誤差 Δc は次のように表される。

$$c = f(a, b),\ \Delta c = |f_x(a, b)|\,\Delta a + |f_y(a, b)|\,\Delta b \tag{4}$$

数値計算で行う計算は、四則演算がほとんどである。故に、四則演算における誤差の伝播を考えることが重要である。四則演算における誤差の伝播は、以下のように表される。まず、加減算における誤差の伝播について考える。量 z は、x と y の加減算によって計算されるとする。

$$z = x \pm y \tag{5}$$

このとき、z の誤差 Δc は、x と y の誤差 Δa と Δb によって次のように表される。

$$\Delta c = \Delta a + \Delta b \tag{6}$$

負の符号が消えているが、これは量 Δc は誤差として含まれる量の最大値であるためである。次に、乗算と除算における誤差の伝播について考える。量 z は、x と y の乗算と除算によって計算されるとする。

$$z = x \cdot y,\ z = x/y \tag{7}$$

このとき、z の誤差 Δc は、真値 a 及び b、c と x と y の誤差 Δa と Δb によって次のように表される。

$$\left|\frac{\Delta c}{c}\right| = \left|\frac{\Delta a}{a}\right| + \left|\frac{\Delta b}{b}\right| \tag{8}$$

上記の式は、分母に真値があることから分かるように相対的な値であり、相対誤差と呼ばれる。一方、加減算における誤差の伝播は、絶対誤差と呼ばれる。

1.2 クイズ

Python を用いて、以下の計算を行い、その結果を出力せよ。

1. 1 を 1000 回足し合わせる計算
2. 0.1 を 1000 回足し合わせる計算

1.3 数の表現

整数型は、計算機のメモリ上で 2 進数として表現される。整数型の表現には、符号付きと符号なしの 2 種類がある。符号付き整数型は、正負の値を表現するために 1 ビットを符号ビットとして用いる。符号なし整数型は、正の値のみを表現するために符号ビットを用いない。整数型の表現には、2 の補数表現が用いられる。2 の補数表現は、正の値と負の値を同じビットパターンで表現することができる。2 の補数表現は、符号ビットを用いる符号付き整数型においても、符号ビットを用いない符号なし整数型においても用いられる。現在の計算機では、64 ビットの整数型が一般的である。64 ビットの整数型は、2^{64} 個の整数を表現することができる。10 進数で表現すると、およそ 19 桁の整数を表現することができる。構造図を図 1 に示す。

一方、実数型は、計算機のメモリ上で固定少数点数もしくは浮動小数点数として表現される。固定小数点は、整数型と同様に 2 進数として表現される。固定小数点は、整数型と同様に符号付きと符号なしの 2 種類がある。固定小数点は、整数型と同様に 2 の補数表現が用いられる。固定小数点は、小数点以下の桁数を固定して表現する (図 2)。固定小数点は、小数点以下の桁数を固定して表現するため、小数点以下の桁数を表現するためにビット数を増やす必要がある。マイコンなどの組み込みシステムなどで使用されることはあるが、一般的な数値計算で使用する計算機では固定小数点を用いることは少ない。

浮動小数点数は、指数部と仮数部から構成される。浮動小数点数の表現には、IEEE754 規格が用いられる。図 3 が IEEE754 規格のビット割り当てを表現している。IEEE754 規格は、32 ビットと 64 ビットの 2 種類がある。32 ビットの浮動小数点数は、単精度浮動小数点数と呼ばれ、64 ビットの浮動小数点数は、倍精度浮動小数点数と呼ばれる。単精度浮動小数点数は、32 ビットのメモリを用いて表現される。単精度浮動小数点数は、符号ビット、指数部、仮数部から構成される。符号ビットは、正負の値を表現するために 1 ビットを用いる。指数部は、浮動小数点数の桁数を表現するために 8 ビットを用いる。仮数部は、浮動小数点数の精度を表現するために 23 ビットを用いる。倍精度浮動小数点数は、64 ビットのメモリを用いて表現される。倍精度浮動小数点数は、符号ビット、指数部、仮数部から構成される。符号ビットは、正負の値を表現するために 1 ビットを用いる。指数部は、浮動小数点数の桁数を表現するために 11 ビットを用いる。仮数部は、浮動小数点数の精度を表現するために 52 ビットを用いる。指数部のビット数が単精度と倍精度では、異なるため、単精度と倍精度の浮動小数点数の表現範囲が異なる。

浮動小数点数など有限の桁数で実数を表現する際、表現できない桁は丸められる。この過程で発生する理論上の値との差を丸め誤差と呼ぶ。丸め誤差により、数値計算の結果は理論上の厳密解と異なる値になることがある。

浮動小数点数を用いた繰り返し足し算における丸め誤差の累積が生じることは先に説明したとおりである。図 4 では、青線は 0.04 を繰り返し足していった場合の数学的に正確な結果を示し、赤い破線は丸め誤差によって理論値から徐々に逸脱していく実際の計算値を表している。0.04 のような小数は二進数形式で正確に表現で

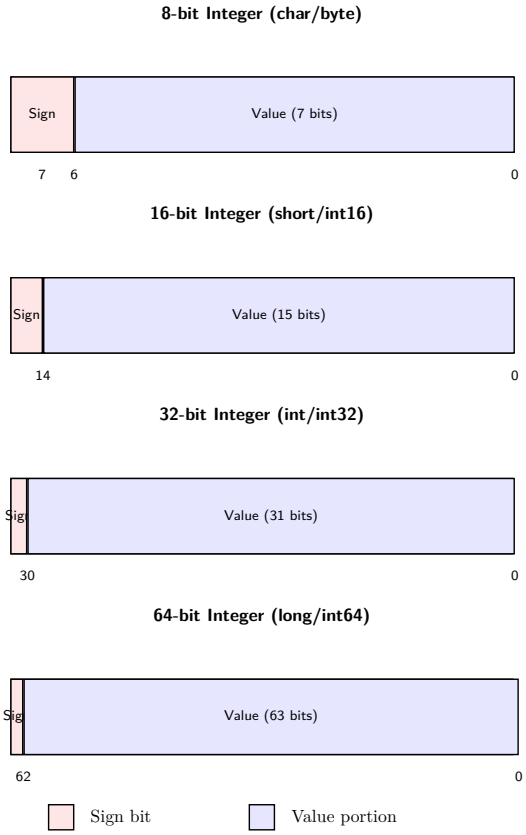

図 1　Integer data type structure.

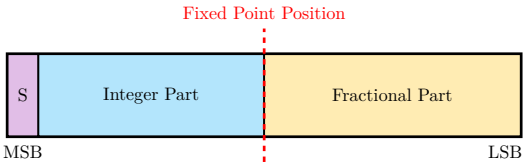

図 2　Fixed–point number structure.

きないため、各演算ごとに小さな丸め誤差が生じる。これらの誤差は相殺されるのではなく、系統的に蓄積される傾向にある。赤く塗られた領域は、足し算の回数が増えるにつれて真の値と計算値の間の差異が拡大していく様子を強調したものである。

1.4　桁落ちに気をつけよう

方程式 $f(x) =$ を解くことは、数値計算的には「$f(x)$ が十分 0 に近くなるような x を求める」ということである。誤っても、$f(x)$ が 0 になる x を求めることはできない。具体的に、次の方程式について考えるとする。

$$f(x) = x^3 - 15.70875x^2 + 61.69729x - 0.04844725 \tag{9}$$

この方程式の零点は以下の付近にある。

$$x \simeq 0.0007853982,\ 7.844763,\ 7.863201 \tag{10}$$

問題は、それぞれの零点の値を求めるとき、$f(x)$ の値をどこまで小さくすることで解くことができたのかということである。

あまり厳しい条件を設定しても、計算中に発生する丸め誤差があるため条件を満足させることはできない。ここでは、丸め誤差の知識を利用して、大まかな見当をつけることが重要である。

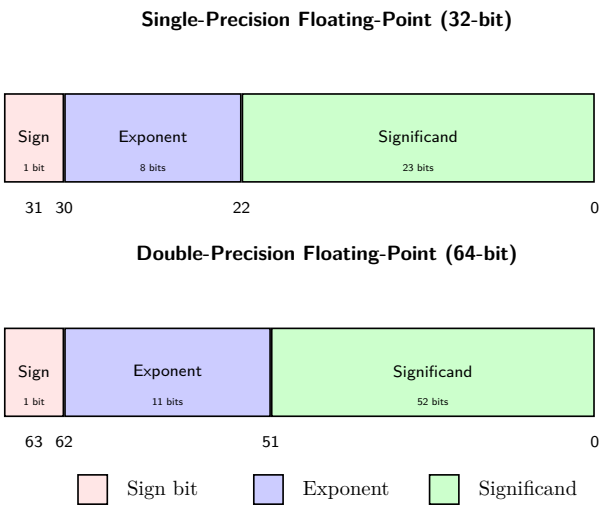

図 3　IEEE754 floating–point structure.

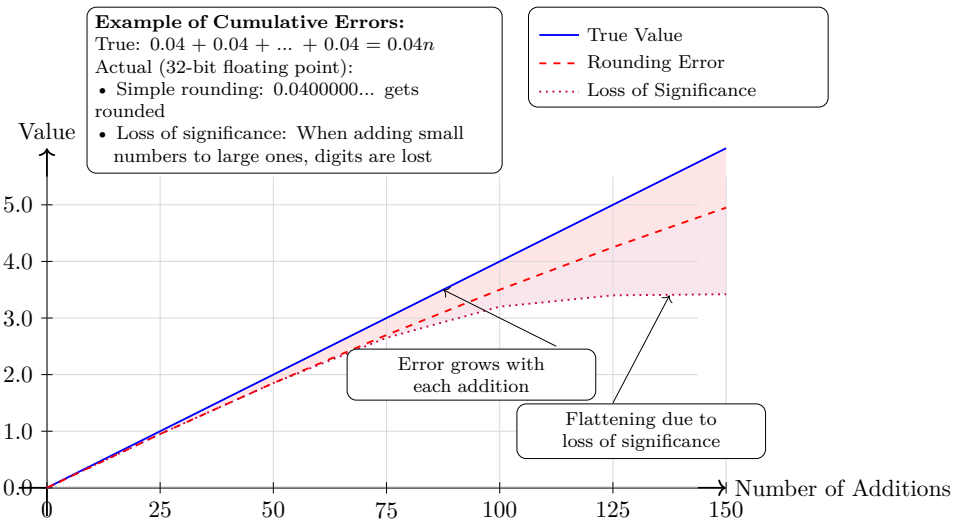

図 4　Accumulation of Rounding Errors in Repeated Addition.

$x \simeq 0.00078$ の付近について検討する。それぞれの項の値を計算すると、次のようになる。

$$x^3 \simeq 4.8 \times 10^{-10},\ -15.70875 \times x^2 \simeq -9.7 \times 10^{-6},\ 61.69729 \times x \simeq 0.048 \tag{11}$$

ここで、計算を単精度浮動小数点数の条件で行うと、10 進 7 桁程度の有効数字で計算（四捨五入）される。このとき、各項で 10^{-16} 及び 10^{-11}、10^{-8} 程度の誤差が生じる。従って、$|f(x)|$ が 10^{-8} のオーダーになったならば、それ以上細かく x の値を調整しても、それに伴う $f(x)$ の値の変化には意味がない。

1.5 境界値問題

境界値問題とは、微分方程式の解を求める際に、境界条件が与えられている問題である。例えば、ある 1 次元空間を考える。この空間において、距離 x における関数 $f(x)$ が次の微分方程式を満たすとする（図 5)。

$$\frac{d^2u}{dx^2} + p(x)\frac{du}{dx} + q(x) = f(x) \tag{12}$$

ここで、変数 u の値を知りたい点 x_0 が区間 $[a, b]$ に含まれている。このとき、$u(x_0)$ を求める問題を境界値問題と呼ぶ。

1.6 境界条件

境界値問題では、境界条件が与えられている。境界条件とは、図 5 において、$x = a$ および $x = b$ において、u の値が与えられていることである。境界条件には、次の 3 つがある。Dirichlet 境界条件、Neumann 境界条件、および Robin 境界条件である。多くの場合は、Dirichlet 境界条件と Neumann 境界条件が用いられる。しかし、練成問題においては、Robin 境界条件が用いられることもある。

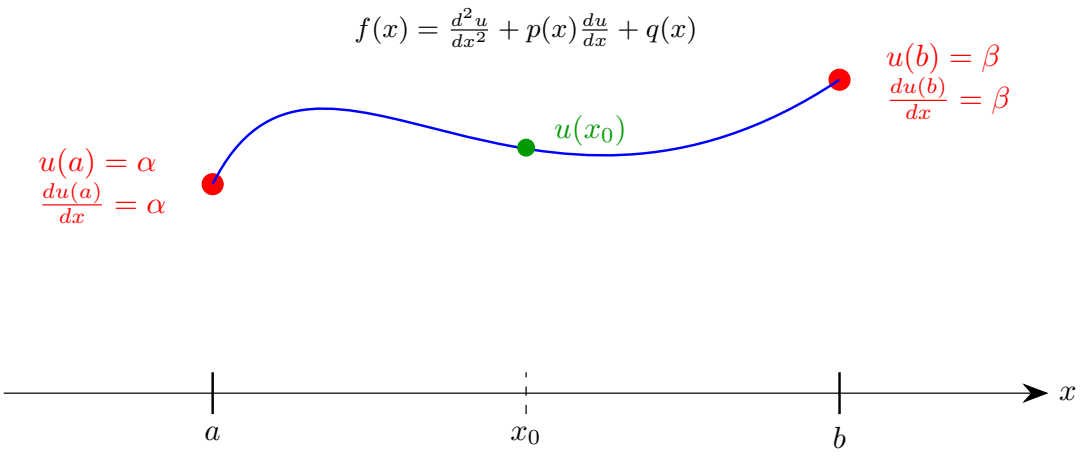

図 5　Boundary value problem.

1.6.1 Dirichlet 境界条件（第 1 種境界条件）

関数値自体が境界上で指定される条件である。

$$u(a) = \alpha \tag{13}$$
$$u(b) = \beta \tag{14}$$

ここで α と β は与えられた定数である。

1.6.2 Neumann 境界条件（第 2 種境界条件）

関数の導関数（勾配）が境界上で指定される条件である。

$$\left.\frac{du}{dx}\right|_{x=a} = \gamma \tag{15}$$

$$\left.\frac{du}{dx}\right|_{x=b} = \delta \tag{16}$$

ここで γ と δ は与えられた定数である。

1.6.3 Robin 境界条件（第 3 種境界条件または混合境界条件）

関数値とその導関数の線形結合が境界上で指定される条件である。

$$a_1 u(a) + b_1 \left.\frac{du}{dx}\right|_{x=a} = c_1 \tag{17}$$

$$a_2 u(b) + b_2 \left.\frac{du}{dx}\right|_{x=b} = c_2 \tag{18}$$

ここで a_1, b_1, c_1, a_2, b_2, c_2 は与えられた定数である。

2 数値微分

Neumann 境界条件（第 2 種境界条件）と Robin 境界条件（第 3 種境界条件を数値的に求めるためには、数値微分を用いる。数値微分は、関数の導関数を数値的に求める手法である。ここでは、中心差分法を用いて数値微分を行う方法について説明する。

2.1 中心差分法

中心差分法とは、関数の導関数を数値的に求める手法である。関数 $f(x)$ の導関数 $f'(x)$ を求めるためには、次の式を用いる。

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{19}$$

ここで、h は微小な値である。この式は、関数 $f(x)$ の x における導関数 $f'(x)$ を、x の前後の値を用いて求めるものである。この式を用いることで、関数の導関数を数値的に求めることができる。

注目すべきは、中心差分法では、分母に $2h$ が現れることである。導関数の定義は以下の通りである。

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{20}$$

計算機では、h を限りなく小さくすることはできないため、h を有限の値として計算する必要がある。この考えに従うと、数値微分として適切なのは

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h} \tag{21}$$

であると言える。

2.2 誤差の可視化

前方差分法（Forward Difference Method）と中心差分法（Central Difference Method）の誤差解析を行う。図 6 に、関数 $f(x)$ の導関数 $f'(x)$ を求めるために、前方差分法と中心差分法を用いたときの誤差を示す。刻み h が小さくなるほど誤差が小さくなるが、前方差分法は中心差分法よりも誤差が大きいことがわかる。また、刻みが小さくなりすぎると、丸め誤差が支配的になる。この影響で、刻みが小さすぎると誤差が大きくなる。このように、数値微分においては、刻みの選択が重要である。

2.3 基本的な誤差特性

図 6 は、関数 $f(x)$ の導関数 $f'(x)$ を求めるために、前方差分法と中心差分法を用いたときの誤差を刻み幅 h の関数として示している。図から明らかなように、刻み幅 h が小さくなるほど両手法の誤差は減少するが、同じ刻み幅 h に対して前方差分法の誤差は中心差分法よりも大きい。これは理論的な誤差のオーダーの違いによるものである。

しかし、図 6 は両手法において、刻み幅 h が一定の閾値より小さくなると、誤差が再び増加する現象も示している。この現象は、コンピュータの浮動小数点演算における丸め誤差に起因する。数値微分における総誤差は、大きく分けて以下の二つの要素から構成される：

$$\text{Total Error} \approx \text{Truncation Error} + \text{Rounding Error} \approx C_1 h^p + \frac{C_2 \varepsilon}{h} \tag{22}$$

ここで、p は差分法の次数（前方差分法では $p = 1$、中心差分法では $p = 2$）、ε はマシンイプシロン（計算機の浮動小数点精度の限界）、C_1 と C_2 は定数である。刻み幅 h が小さい領域では、丸め誤差項 $\frac{C_2 \varepsilon}{h}$ が支配的になる。丸め誤差は h に反比例するため、h が極端に小さくなると総誤差は増大する。

数値微分において、刻み幅 h の選択は結果の精度に直接影響する重要な要素である。刻み幅が大きすぎれば切断誤差が大きくなり、小さすぎれば丸め誤差が支配的になる。最適な刻み幅の選択により、最小の総誤差を達成することが可能である。

この分析は、数値計算において理論的な精度と計算機の制約のバランスを取ることの重要性を示している。実際の応用では、問題の性質や要求される精度に応じて、適切な差分法と刻み幅を選択することが肝要である。

2.3.1 前方差分法（Forward Difference Method）

点 x における関数 $f(x)$ の導関数 $f'(x)$ の前方差分近似は次式で与えられる：

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

ここで $h > 0$ は差分のステップサイズである。

テイラー展開を用いると、$f(x+h)$ は次のように展開できる：

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + O(h^4)$$

これを前方差分の式に代入すると：

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{h}{2} f''(x) + \frac{h^2}{6} f'''(x) + O(h^3)$$

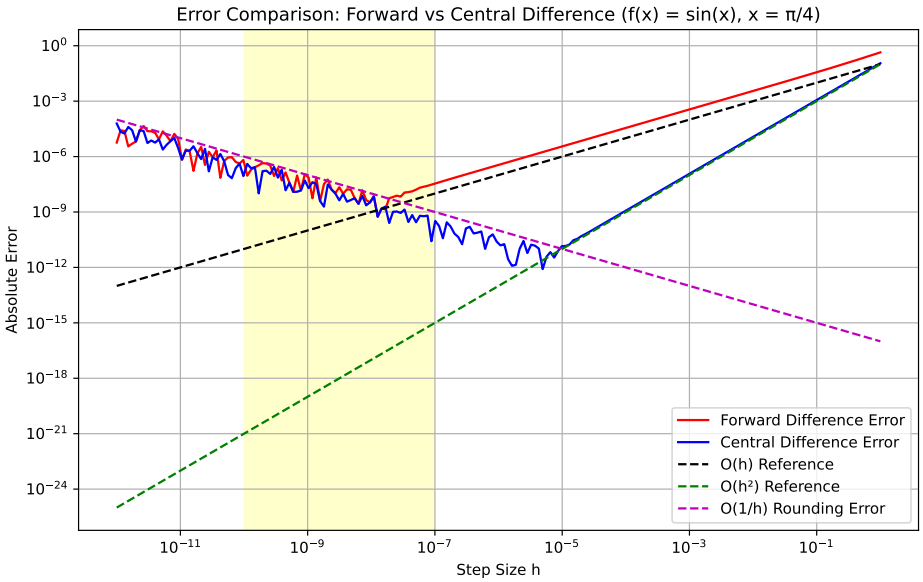

図 6　Boundary value problem.

したがって、前方差分の誤差項は $\frac{h}{2}f''(x) + O(h^2)$ であり、前方差分は $O(h)$ の精度（一次精度）を持つことがわかる。

2.4　中心差分法（Central Difference Method）

2.4.1　定式化

点 x における関数 $f(x)$ の導関数 $f'(x)$ の中心差分近似は次式で与えられる：

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

2.4.2　誤差解析

同様にテイラー展開を用いると：

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4)$$

これらの差を取ると：

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{2h^3}{6}f'''(x) + O(h^5)$$

したがって：

$$\frac{f(x+h)-f(x-h)}{2h} = f'(x) + \frac{h^2}{6}f'''(x) + O(h^4)$$

中心差分の誤差項は $\frac{h^2}{6}f'''(x)+O(h^4)$ であり、中心差分は $O(h^2)$ の精度（二次精度）を持つことがわかる。

2.5 ステップサイズ h の選択

数値微分における重要な考慮点は、ステップサイズ h の選択である。

- h が大きすぎると、近似誤差（切断誤差）が大きくなる。
- h が小さすぎると、浮動小数点演算における丸め誤差が支配的になる。

理論的には、ステップサイズ h と総誤差 E の関係は次のように表される：

$$E \approx C_1 h^p + \frac{C_2}{\epsilon}h^{-1}$$

ここで、p は差分法の次数、C_1 と C_2 は定数、ϵ は計算機のマシンイプシロンである。最適なステップサイズは、この総誤差を最小化する値となる。

2.6 比較

中心差分は前方差分と比較して、同じステップサイズでより高い精度が得られるため、効率的である。

微分方程式を数値的に解く際、差分法の安定性が重要な考慮点となる。中心差分は、ある種の問題（例えば放物型偏微分方程式）において不安定になる可能性がある。

前方差分は $O(h)$ の精度を持ち、中心差分は $O(h^2)$ の精度を持つ。誤差の観点からは、中心差分が前方差分よりも優れているが、境界条件や問題の性質によっては前方差分または他の差分スキームが適している場合もある。数値微分の選択は、精度要件、計算効率、および問題の特性を考慮して行うべきである。

3 有限差分法

3.1 有限差分法の基本原理

有限差分法（Finite Difference Method）は、微分方程式を差分方程式に変換し、数値的に解く手法である。有限差分法は、微分方程式の境界値問題を数値的に解くために広く用いられている。

1 次元蝸牛モデルでは、1 次元 Poisson 方程式に対して有限差分法を用いて解く。ここでは、Poisson 方程式が次の形式で与えられるとする。

$$\frac{d^2u}{dx} = -f(x) \tag{23}$$

二階微分の中心差分近似は次のように表される：

$$\frac{d^2u}{dx^2} \approx \frac{u(x+h)-u(x)+u(x-h)}{h^2} \tag{24}$$

1 次元ポアソン方程式 $\frac{d^2\phi}{dx^2} = -f(x)$ を離散化すると、計算領域を等間隔 h の格子点に分割に分割する:

$$x_0, x_1, \ldots, x_n \ (x_i = x_0 + i \cdot h) \tag{25}$$

各格子点での u の値を $u_0, u_1, \ldots, u_n$ と表す。

二階微分の差分近似を適用すると：

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = -f(x_i) \tag{26}$$

これは次の線形方程式系に変形できる：

$$u_{i+1} - 2u_i + u_{i-1} = -h^2 \cdot f(x_i) \quad (i = 1, 2, \ldots, n-1) \tag{27}$$

この方程式系は、未知数 u_i を求めるための連立方程式系であり、次の行列形式で表すことができる。

$$\mathbf{A}\mathbf{u} = \mathbf{b} \tag{28}$$

ここで、ベクトル $\mathbf{u}$ は未知数 u_i の値を格納するベクトルであり、ベクトル $\mathbf{b}$ は右辺項 $f(x_i)$ の値を格納するベクトルである。行列 $\mathbf{A}$ は境界条件によって異なる。

Dirichlet 境界条件の場合、u_0 と u_n が与えられている。そのため、u_0 と u_n を用いて、u_1 から u_{n-1} までの未知数を求めることができる。

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ n-1 \end{bmatrix} = \begin{bmatrix} h^2 f(x_1) + u_0 \\ h^2 f(x_2) \\ h^2 f(x_3) \\ \vdots \\ h^2 f(x_{n-1}) + u_n \end{bmatrix} \tag{29}$$

Neumann 境界条件では、u_{-1} もしくは u_{n+1} の値が必要となる。しかし、これらの値は未知である。そのため、Neumann 境界条件を用いる場合は、境界条件を数値微分を用いて求める必要がある。

$$\left.\frac{du}{dx}\right|_{x=a} = \gamma \simeq \frac{u_1 - u_{-1}}{2h} \tag{30}$$

$$\left.\frac{du}{dx}\right|_{x=b} = \delta \simeq \frac{u_{n+1} - u_{n-1}}{2h} \tag{31}$$

これより、Neumann 境界条件を用いた場合の行列形式は次のようになる。

$$\begin{bmatrix} 1 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \begin{bmatrix} \frac{h^2 f(x_0)}{2} + h\gamma \\ h^2 f(x_1) \\ h^2 f(x_2) \\ \vdots \\ h^2 f(x_{n-1}) \\ \frac{h^2 f(x_n)}{2} - h\delta \end{bmatrix} \tag{32}$$