CC BY

情報実験・第 10 回 (2026/07/03)

数値計算入門

北海道大学大学院 理学院 宇宙理学専攻
博士 1 年
吉川 颯真

CC BY

はじめに

計算機は「計算をする機械」です

「計算機が計算をする」ということは
どういうことなのでしょうか？

本日は計算機での計算の仕組みについて学びましょう！

今回の目標
- 計算機が計算をする原理
- 計算機で微分方程式を解く手続き

を説明できるようになる

CC BY

本日のレクチャー内容

- 計算機が扱うデータ
- 実数の表現と誤差
- 計算処理の原理
- プログラムとその実行
- 数値計算で微分方程式を解く

CC BY

1. 計算機が扱うデータ

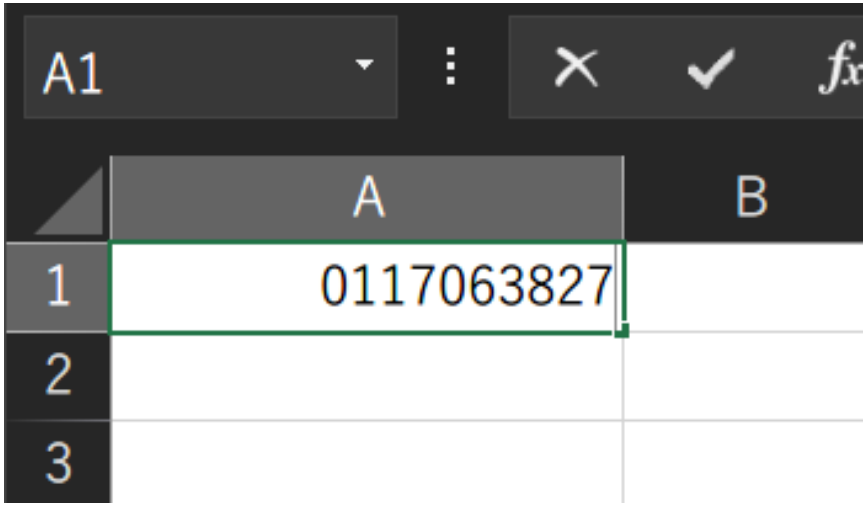
CC BY

データ型

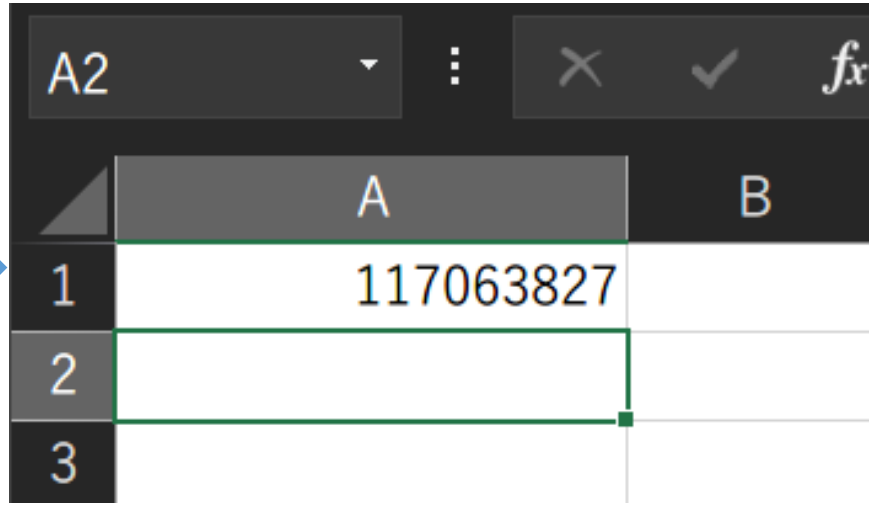
- データ型: データの種類のこと

データ型

- データ型を指定することで
  ソフトウェアのエラーを未然に防ぐことができる



電話番号を入力



数として解釈される

データ型

- 自分が扱うデータがどのように記録され，どのように処理されるかを予め設定しておく必要がある (型の宣言)
- Fortran に組み込まれているデータ型
  - 整数型 (integer)
  - 実数型 (real)
  - 複素数型 (complex)
  - 文字型 (character)

CC BY

2. 実数の表現と誤差

我々が使う数と計算機が使う数

- 我々はなぜ 1013.25 を 1013.25 と表現するのだろう

我々が使う数と計算機が使う数

- 我々はなぜ 1013.25 を 1013.25 と表現するのだろう
- 人間は 10 種類の数字 (0-9)と
  10 の累乗を基に数を表記しているから (10 進法)

$$1013.25 = 1 \times 1000 + 0 \times 100 + 1 \times 10 + 3 \times 1 + 2 \times 0.1 + 5 \times 0.01$$

我々が使う数と計算機が使う数

- 我々はなぜ 1013.25 を 1013.25 と表現するのだろう
- 人間は 10 種類の数字 (0-9)と
  10 の累乗を基に数を表記しているから (10 進法)

$$1013.25 = 1 \times 1000 + 0 \times 100 + 1 \times 10 + 3 \times 1 + 2 \times 0.1 + 5 \times 0.01$$

- ただし，計算機は 0 と 1 の 2 種類の数字しか使えない

↓

- 2 の累乗を基にした数の表記法が必要！ (2 進法)

2 進法

- 1013.25 を「2 の累乗」の和で構成してみる

$$1013.25 = 1 \times 512 + 1 \times 256 + 1 \times 128 + 1 \times 64 + 1 \times 32 + 1 \times 16$$
$$+ 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 + 0 \times \frac{1}{2} + 1 \times \frac{1}{4}$$
$$= (1111110101.01)_2$$

- このように，数を「2 の累乗」の和で構成し 0, 1 で数を表記する表記法を 2 進法という

N 進法

- 数を N の累乗の和で構成すれば，N 進法を定義できる

実数 x の表示法を考える．正の整数 N に対して，

$$x = a_k N^k + a_{k-1} N^{k-1} \cdots + a_1 N^1 + a_0 + a_{-1} N^{-1} + a_{-2} N^{-2} + \cdots$$

を満たす N 未満の非負整数 a_i (i は整数) が一意的に定まり，このとき，

$$x = (a_k a_{k-1} \cdots a_1 a_0 . a_{-1} a_{-2} \cdots)_N$$

と表記する．この表記法を N **進法**という

N 進法の利用

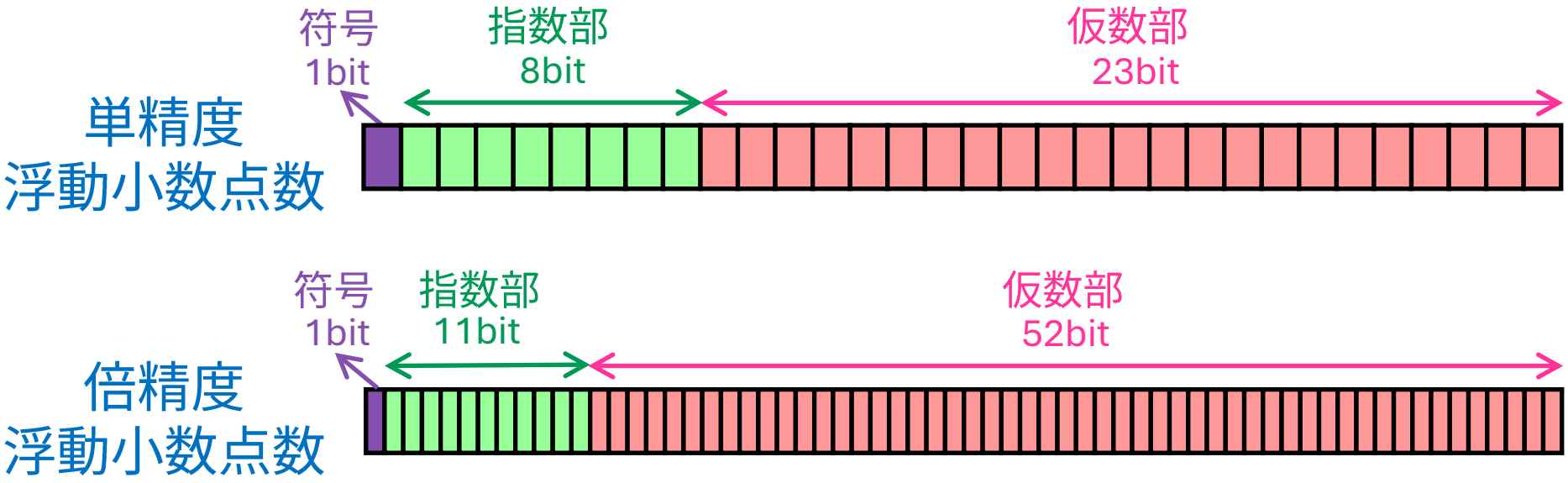
- 10 進法
  - 0-9 で数を記述
  - 私たちが通常使用している数の表記法
- 2 進法
  - 0 と 1 で数を記述
  - 計算機内部で利用されている．
- 16 進法
  - 0-9, A-F で数を記述（例：$1013.25 = (3\mathrm{F}5.4)_{16}$）
  - 桁数が少なく，２進法からの変換も容易
  - ２進法表記の数を人間にとって読みやすくするために用いられる．

計算機における実数の表現

- 計算機は実数を 2 進法の浮動小数点で表現する
  - 浮動小数点表現: 符号，仮数部，指数部で実数を表現

$$\pm(1.a_1a_2a_3\cdots a_k)_2 \times 2^{(E)_2}$$

符号　仮数部　指数部



計算機における実数の表現

- 浮動小数点表現　$\pm(1.a_1a_2a_3\cdots a_k)_2 \times 2^{(E)_2}$

符号　仮数部　指数部

例 (IEEE754 方式)

1013.25 を単精度浮動小数点数で表現する

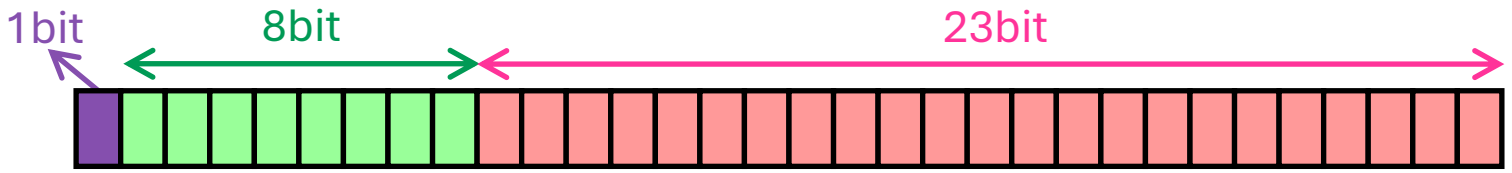
1013.25 を 2 進数で表すと，$(1111110101.01)_2$

正規化: 仮数部の小数点位置を調整 $+(1.1111101010 1)_2 \times 2^9$

本来の指数に 127 を加える $+(1.1111101010 1)_2 \times 2^{136}$

指数部分を 2 進数で表す　$+(1.1111101010 1)_2 \times 2^{(10001000)_2}$

01000100011111010101000000000000



計算機における実数の表現

- 浮動小数点表現　$\pm(1.a_1a_2a_3\cdots a_k)_2 \times 2^{(E)_2}$

符号　仮数部　指数部

例

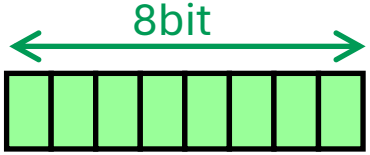
10

10

正

本

指



指数部の表現に
符号を用いない場合，
8 bit の 2 進数で表現

(2 進数)	(10 進数)	実際の指数
11111111	＋255	＋128
⋮	⋮	⋮
01111111	+127	0
⋮	⋮	⋮
00000000	0	－127

計算機における実数の表現

- 浮動小数点表現　$\pm(1.a_1a_2a_3\cdots a_k)_2 \times 2^{(E)_2}$

符号　仮数部　指数部

例 (IEEE754 方式)

1013.25 を単精度浮動小数点数で表現する

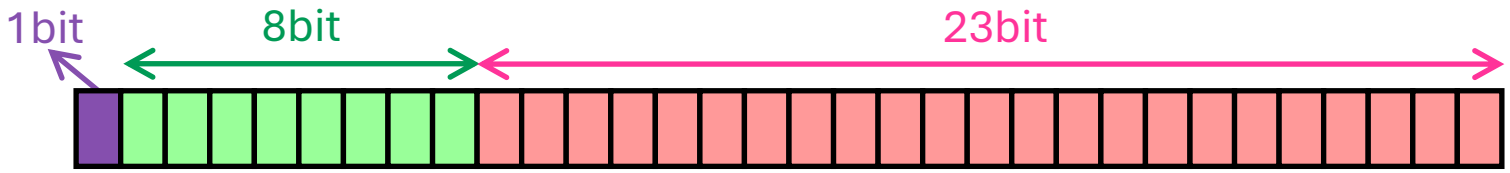
1013.25 を 2 進数で表すと，$(1111110101.01)_2$

正規化: 仮数部の小数点位置を調整 $+(1.1111101010 1)_2 \times 2^9$

本来の指数に127を加える $+(1.1111101010 1)_2 \times 2^{136}$

指数部分を2進数で表す　$+(1.1111101010 1)_2 \times 2^{(10001000)_2}$

01000100011111010101000000000000

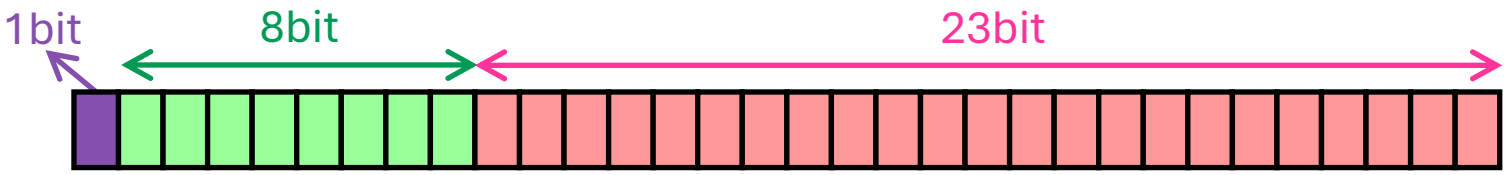


丸め誤差

- 計算機の数表現は実数の厳密な値を表現できない．
  このとき発生する誤差を丸め誤差という．

例: 3.14159265358979323846 の単精度浮動小数点表現

$$+(1.10010010000111111011010)_2 \times 2^{1(+127)}$$



01000000010010010000111111011010

10 進数へ変換すると，

3.141592502593994 (丸め誤差: 約 1.5×10^{-7})

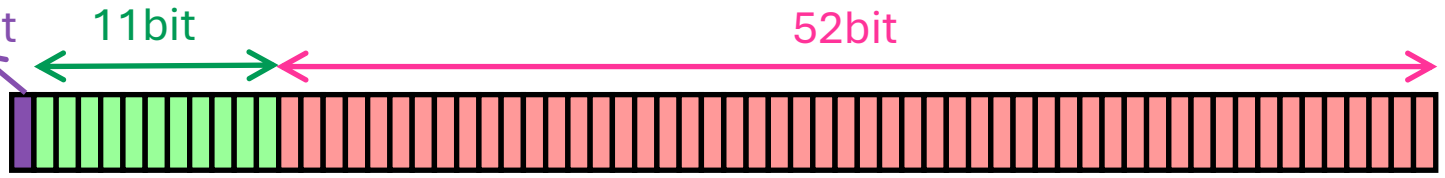
CC BY

丸め誤差

- 計算機の数表現は実数の厳密な値を表現できない．
  このとき発生する誤差を丸め誤差という．

例: 3.14159265358979323846264 の倍精度浮動小数点表現

$+(1.10010010000111111011010101000100010000101101000110 01)_2 \times 2^{(1+127)_2}$



0100000000001001001000011111101101010100010001000010110100011001

10 進数へ変換すると，

3.1415926535897936 (丸め誤差: 約 4.4×10^{-16})

桁落ち

- 近接した実数同士の減算で，
  有効数字の桁数 (有効桁数) が減ることを桁落ちという．

例

有効数字 8 桁で $\sqrt{402} - \sqrt{401}$ を計算する．

$\sqrt{402} = 20.049938$

$\sqrt{401} = 20.024984$

$\sqrt{402} - \sqrt{401} = 0.024954$

有効数字が 5 桁に減ってしまう…

桁落ち

- 近接した実数同士の減算で，
  有効数字の桁数 (有効桁数) が減ることを桁落ちという．

例

有効数字 8 桁で $\sqrt{402}-\sqrt{401}$ を計算する．

$\sqrt{402}=20.049938$

$\sqrt{401}=20.024984$

$\sqrt{402}-\sqrt{401}=0.024954$

有効数字が 5 桁に減ってしまう…

桁落ちを回避する方法例（近接実数間の減算を回避）

$$\sqrt{402}-\sqrt{401}=\frac{\left(\sqrt{402}-\sqrt{401}\right)\left(\sqrt{401}+\sqrt{402}\right)}{\sqrt{401}+\sqrt{402}}$$

$$=\frac{402-401}{\sqrt{401}+\sqrt{402}}=\frac{1}{40.074922}=2.4953261\times10^{-2}$$

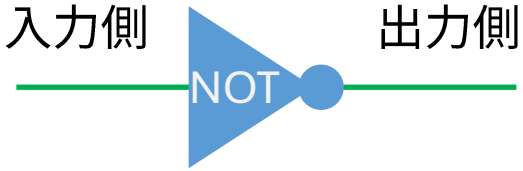
CC BY

3. 計算処理の原理

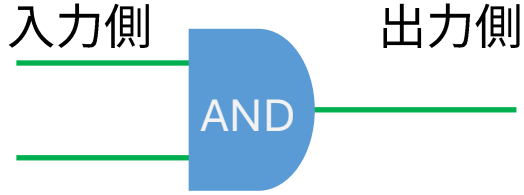
論理回路

- 論理演算を実装した回路．トランジスタやダイオードで構成される．0, 1 は電圧に対応する場合が多い．

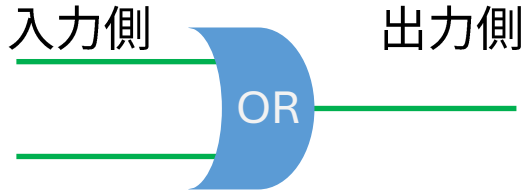
基本論理回路



NOT 回路 (反転回路)
入力信号の 0, 1 を
反転して出力する



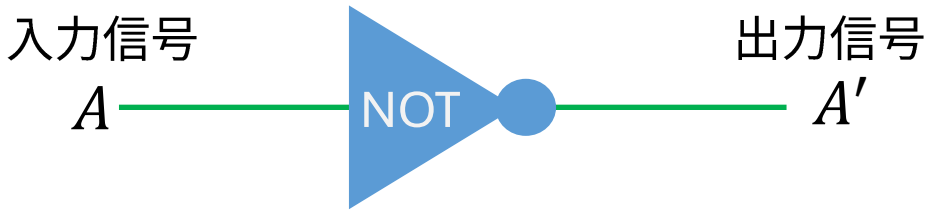
AND 回路 (論理積)
入力信号がどちらも 1 のとき
1 を出力する



OR 回路 (論理和)
入力信号のどちらかが 1 のとき
1 を出力する

NOT 回路

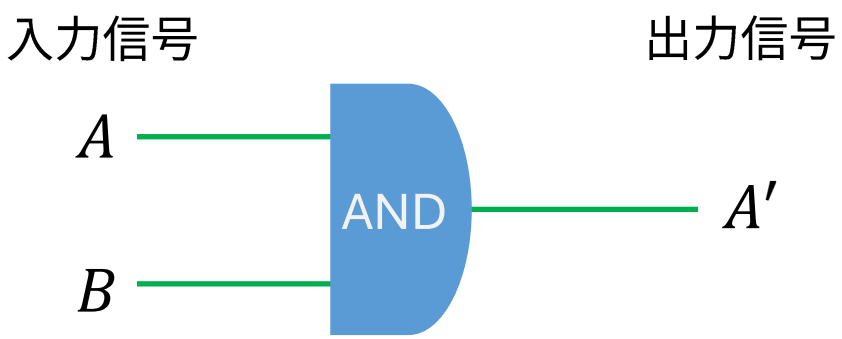
- 入力信号の 0, 1 を反転した信号を出力する．
- 「否定」に対応



A	A'
0	1
1	0

AND 回路

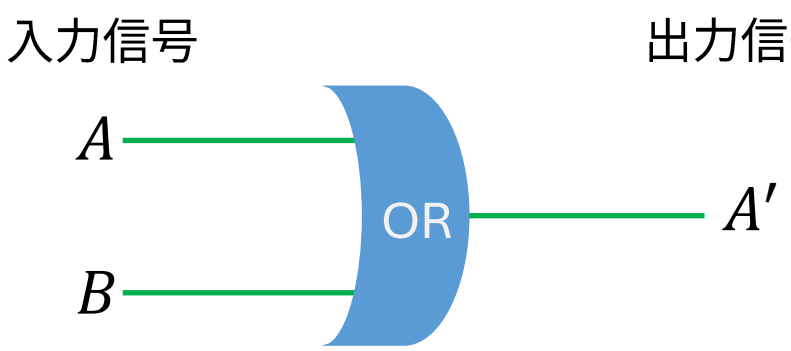
- 2 つの入力信号がどちらも 1 のときのみ 1 を出力する.
- 「かつ」に対応



A	B	A'
0	0	0
0	1	0
1	0	0
1	1	1

OR 回路

- 2 つの入力信号のうち
  どちらかが 1 のとき 1 を出力する.
- 「または」に対応



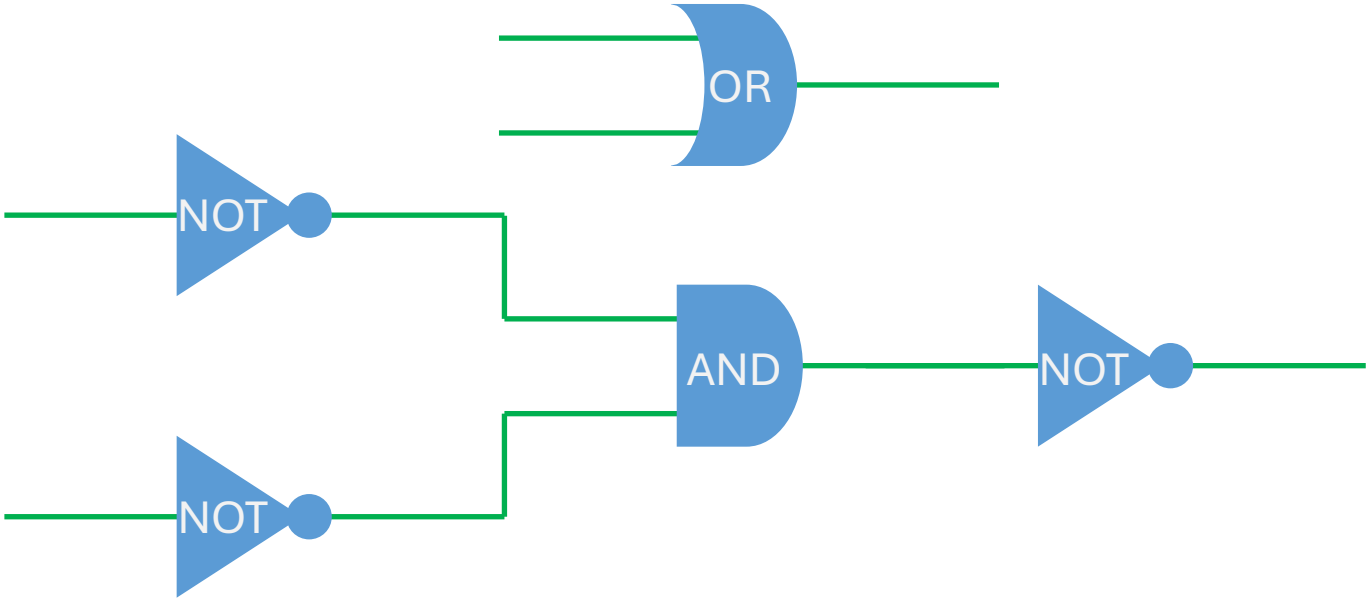
A	B	A'
0	0	0
0	1	1
1	0	1
1	1	1

論理回路の構成

- 基本論理回路 (NOT, AND, OR 回路) を組み合わせれば，任意の論理回路を構成できる

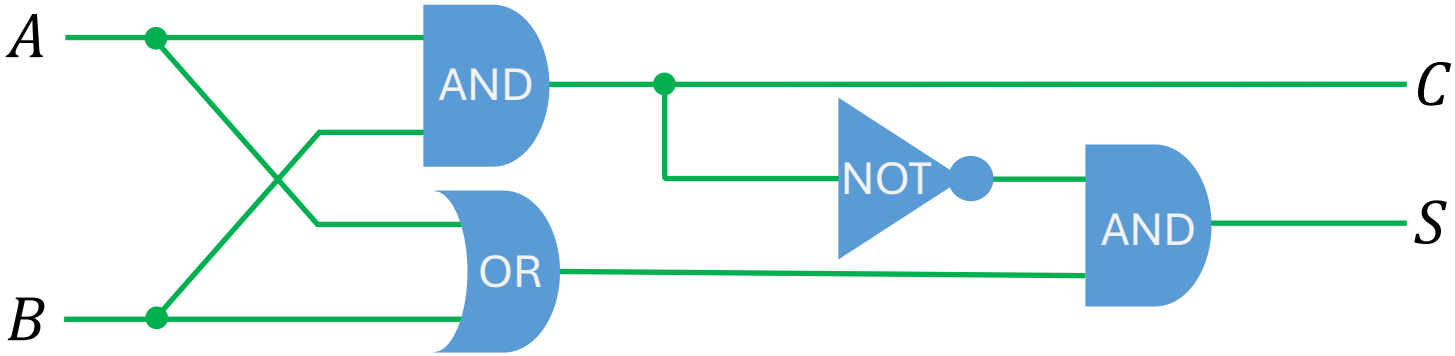
OR 回路を NOT 回路と AND 回路で実装した例

(De Morgan の法則: $A \cup B = \overline{\bar{A} \cap \bar{B}}$)



半加算器

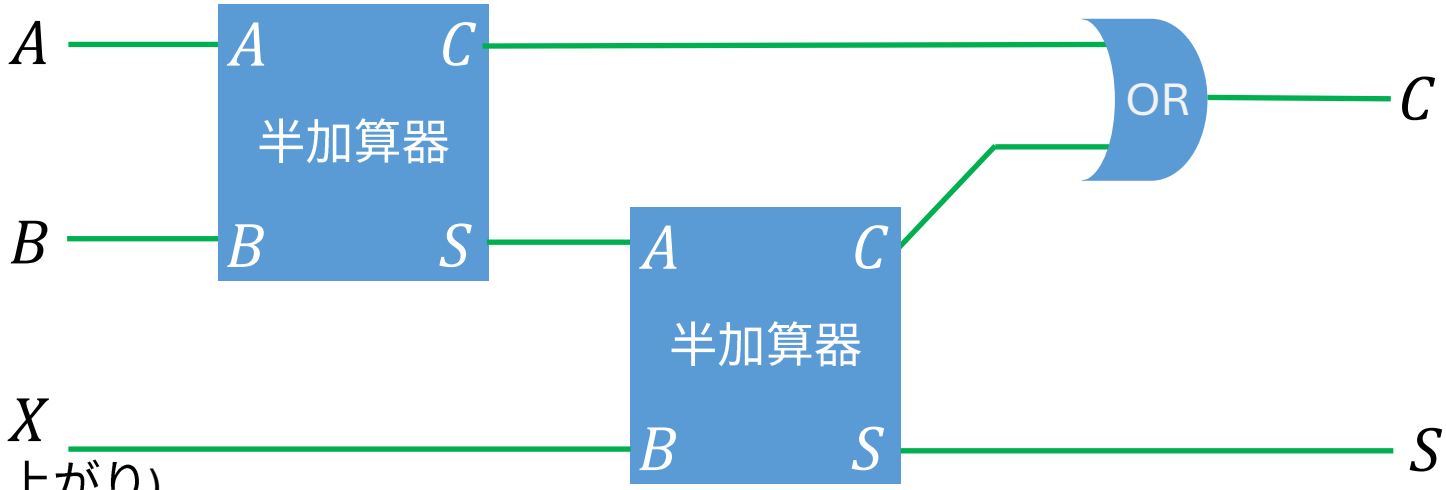
- 1ビットの数の加算処理を行う論理回路



A		B		C	S
0	+	0	=	0	0
0	+	1	=	0	1
1	+	0	=	0	1
1	+	1	=	1	0

全加算器

- 下位からの繰り上がりを含む加算処理を行う論理回路
  - 全加算器を直列に繋げば任意の桁数の加算ができる



A		B		X		C	S
0	+	0	+	0	=	0	0
0	+	1	+	0	=	0	1
1	+	0	+	0	=	0	1
1	+	1	+	0	=	1	0

A		B		X		C	S
0	+	0	+	1	=	0	1
0	+	1	+	1	=	1	0
1	+	0	+	1	=	1	0
1	+	1	+	1	=	1	1

四則演算の処理方法: 減法と除法

- 補数 (足すと桁上がりする最小の数) を加算して桁上がりを破棄する.
- 除法は減法の繰り返しによって処理できる

例

10 進数での引き算 15-9 を行う.

2 桁の範囲での 9 の補数 (足すと 3 桁になる最小数) は 91 なので,

$$15-9 = 15+91-100 = 106-100 = 6$$

四則演算の処理方法: 減法と除法

- 補数 (足すと桁上がりする最小の数) を加算して
  桁上がりを破棄する.
- 除法は減法の繰り返しによって処理できる

例

4 ビットの範囲で 15-9 を 2 進表記に直してから計算する

1111-1001

2 進数における補数は，ビットを反転させ，
1 を足すことで求められる

1001 → 0110 → 0111

```
  1001
+ 0111
------
 10000
```

よって

1111-1001 → 1111+0111 = 10110 → 0110 (桁上がり破棄)

四則演算の処理方法: 乗法

- AND 回路とビットシフト，加法の組み合わせ

例

10×6 の計算結果を 2 進数に変換する．

$$10 \times 6 = 60 = (111100)_2$$

10, 6 を2進数に変換してから計算する．

$$\begin{array}{r} 1010 \times 110 \\ 1010 \times 1 = 1010 \\ 1010 \times 1 = \ \ 1010 \\ 1010 \times 0 = \ \ \ 0000 \\ \hline 111100 \end{array}$$

CC BY

4. プログラムとその実行

CC BY

プログラム言語

- プログラムを記述するのに用いられる言語
  計算機が行う処理を順序立てて記述したもの．
- 高級言語と低級言語の 2 種類がある．
  - **高級言語**
    - 人間が解釈しやすいプログラム言語
    - ハードウェアを意識しなくてもプログラムを書ける．
    - Fortran, C, C++, Java, Lisp, Python, R, Ruby などがある．
  - **低級言語**
    - CPU が直接解釈・実行できる機械語と
      それと一対一に対応したアセンブリ言語の総称
    - ハードウェアを意識したプログラミングが必須

CC BY

コンパイル

- 高級言語で書かれたプログラムは，
  機械語に翻訳 (コンパイル) しなければ実行できない．
- コンパイラ
  - コンパイルを行うプログラム
  - 最近は大部分が高級言語で書かれている．
  - ➢例: GFortran, iFort (Fortran), GCC (C 言語) など．

※コンパイル言語の他にも，プログラムを解釈しながら
実行するプログラム言語もある (インタプリタ: 第 3 回)

プログラム実行までの流れ

(Fortran の場合)

1. コンパイルし，実行ファイルを生成

```
$ gfortran program.f90
```

2. 実行ファイルを実行する

```
$ ./a.out
```

CC BY

5. 数値計算で微分方程式を解く

微分方程式を (手で) 解く

$$\frac{df(x)}{dx} = \frac{af}{x^2}, \quad f(1) = 1 \ (a \text{ は定数}) \text{ を解いてみよう.}$$

微分方程式を (手で) 解く

$$\frac{df(x)}{dx} = \frac{af}{x^2}, \quad f(1) = 1 \ (a \text{ は定数})$$

$$\frac{df(x)}{f} = \frac{adx}{x^2}$$ 両辺積分すると

$$\int \frac{df(x)}{f} = \int \frac{adx}{x^2}$$

$$\ln|f(x)| = -\frac{a}{x} + C_1 \quad (C_1 \text{は定数})$$

$$f(x) = C \exp\left(-\frac{a}{x}\right) \quad (C \text{ は定数})$$

$f(1) = Ce^{-a} = 1 \therefore C = e^a$ なので

$$f(x) = \exp\left(a - \frac{a}{x}\right)$$

微分方程式を解くということ

- 既知の演算や関数の有限な組み合わせで微分方程式の厳密解を解くことを，
  解析的に解く (solve analytically) という．
  ※解析的に解けるのは極めて単純な場合のみ

解析的に解けない微分方程式の例

Navier-Stokes方程式（流体の運動方程式）：

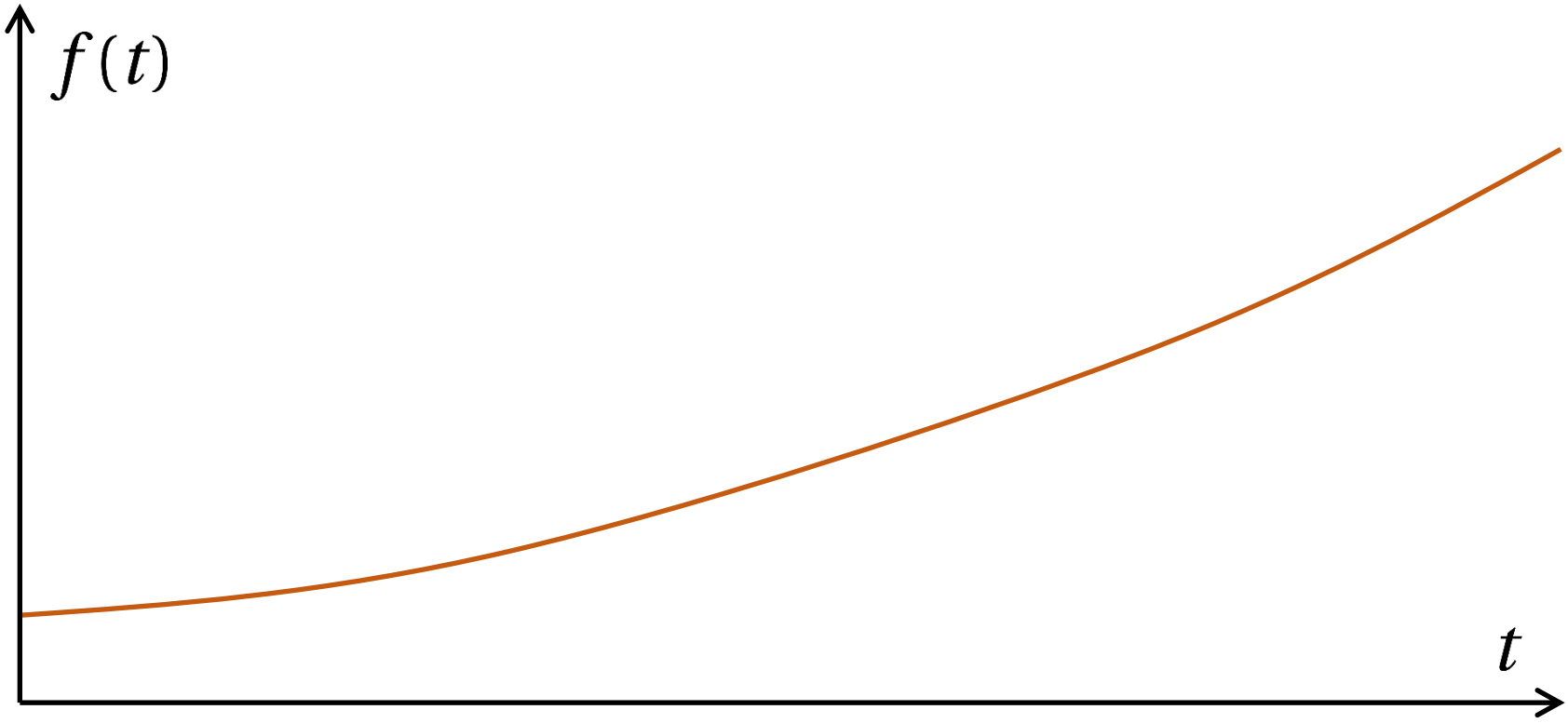
$$\frac{\partial v_i}{\partial t} + \sum_{j=1}^{3} v_j \frac{\partial v_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \sum_{j=1}^{3} \frac{\partial^2 v_i}{\partial {x_j}^2} + f_i(x_1, x_2, x_3, t)$$

$$(i = 1, 2, 3)$$

- 計算機では，微分方程式を解析的に解かず，
  変数に近似を施して数値的に解く (solve numerically)．

微分方程式を数値的に解く

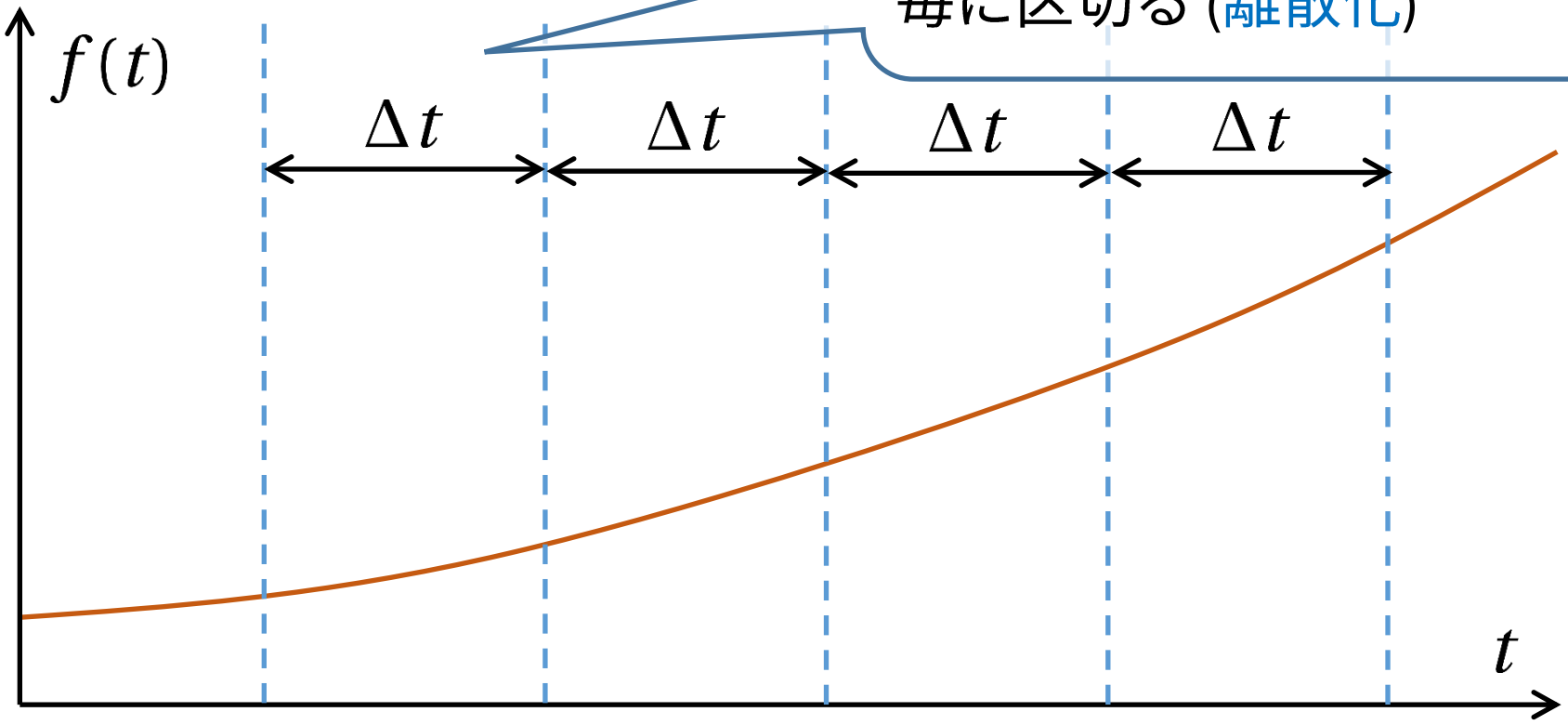
$$\frac{df}{dt} = G\big(t, f(t)\big), \qquad f(t_0) = f_0$$



微分方程式を数値的に解く

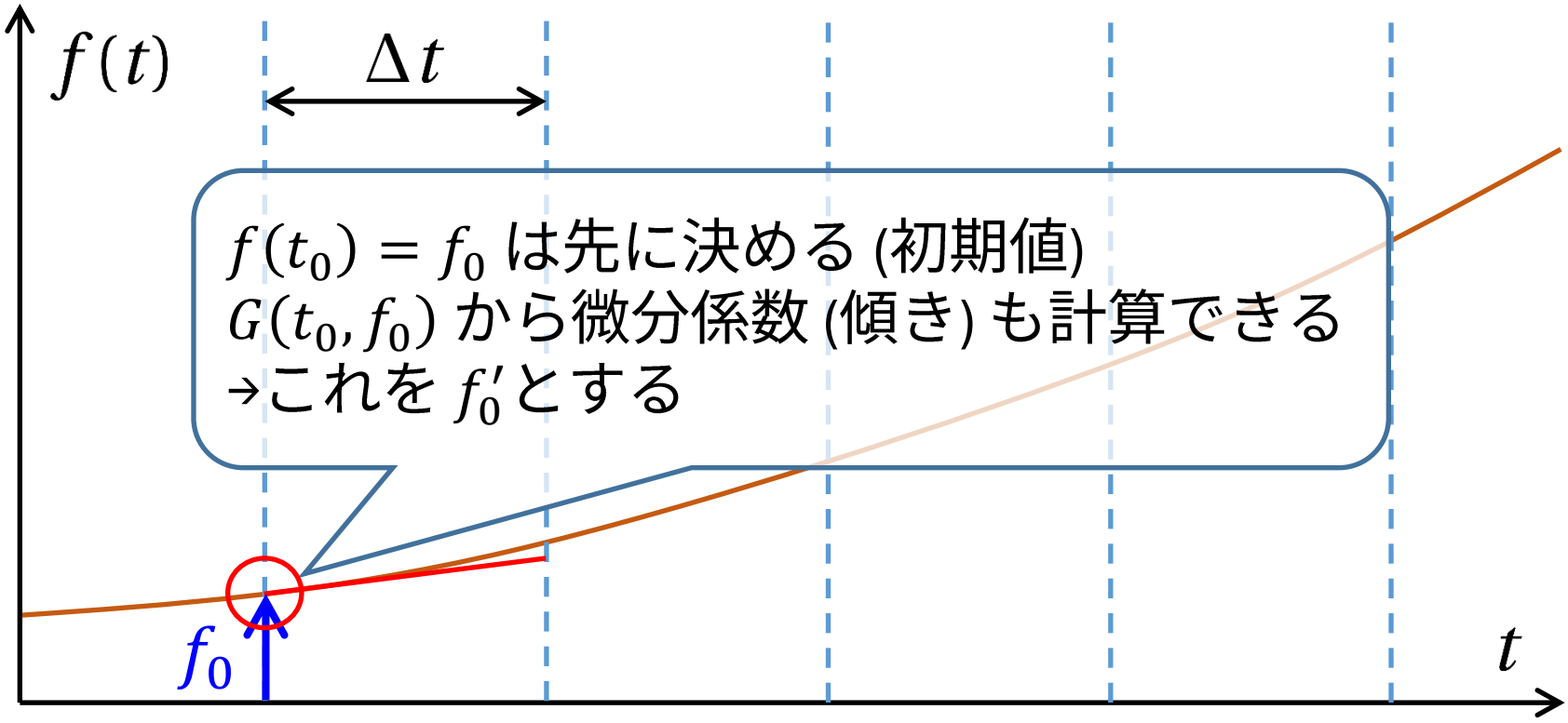
$$\frac{df}{dt} = G\big(t, f(t)\big), \quad f(t_0) = f_0$$

計算機で微分方程式を解くときは，変数をある大きさ毎に区切る (離散化)



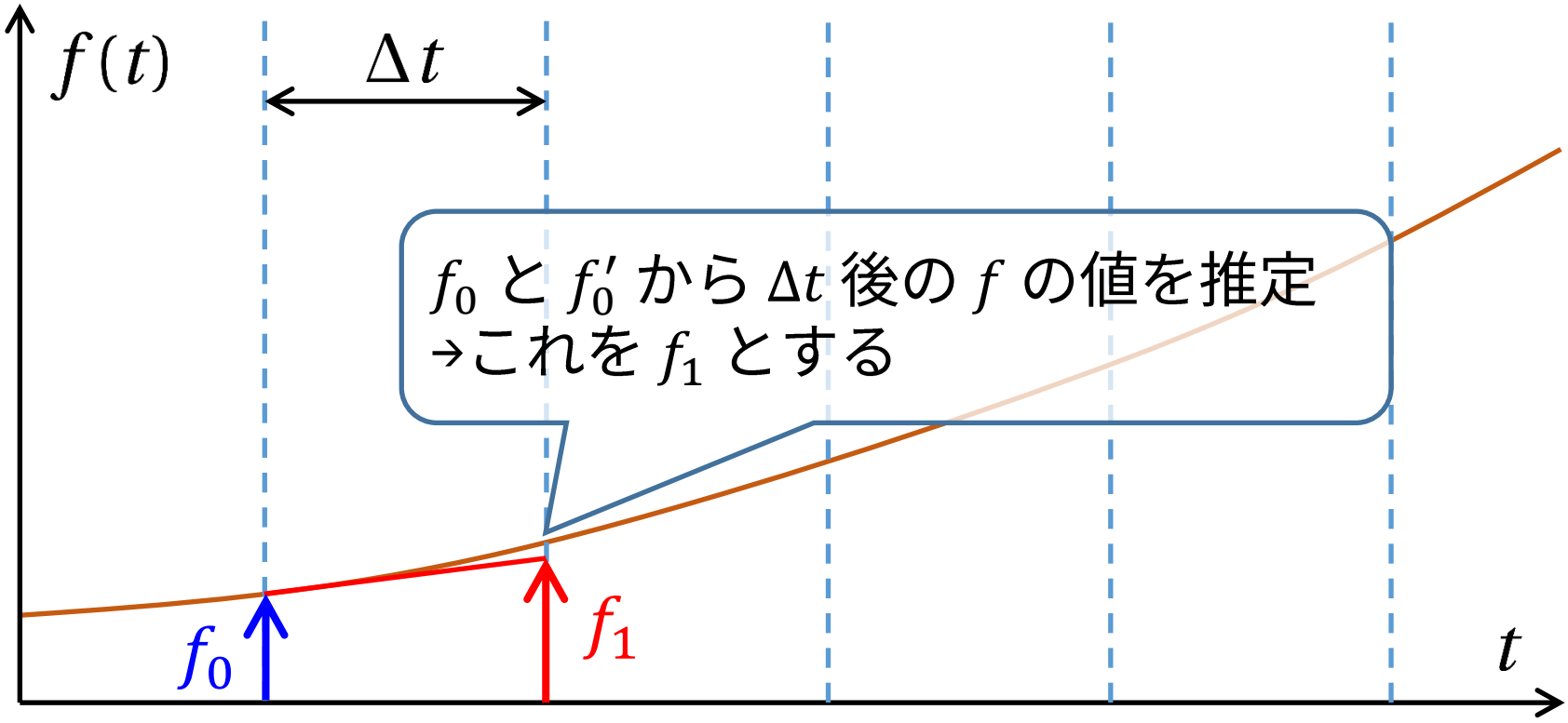
微分方程式を数値的に解く

$$\frac{df}{dt} = G\big(t, f(t)\big), \quad f(t_0) = f_0$$



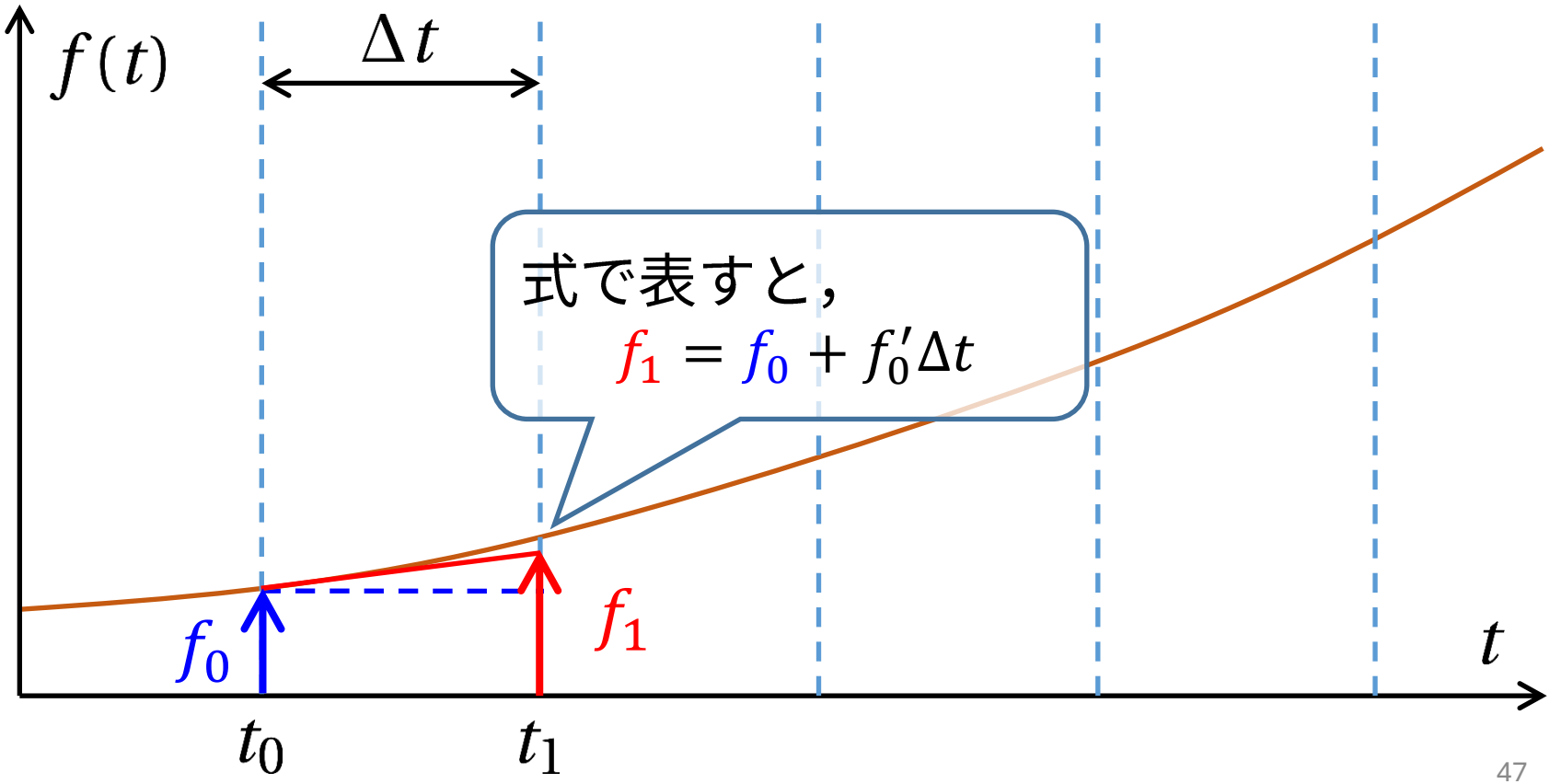
微分方程式を数値的に解く

$$\frac{df}{dt} = G\big(t, f(t)\big), \quad f(t_0) = f_0$$



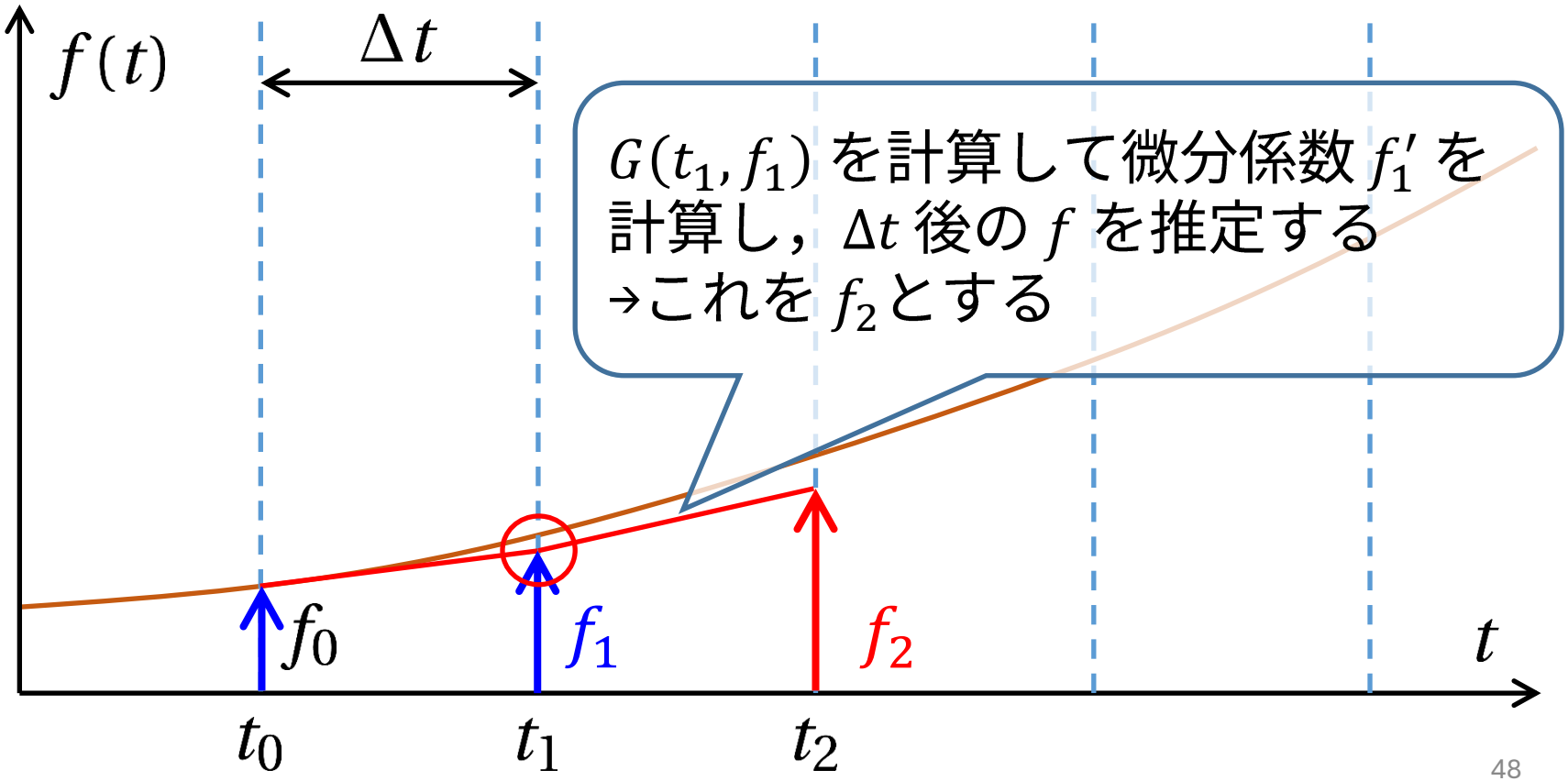
微分方程式を数値的に解く

$$\frac{df}{dt} = G\big(t, f(t)\big), \quad f(t_0) = f_0$$



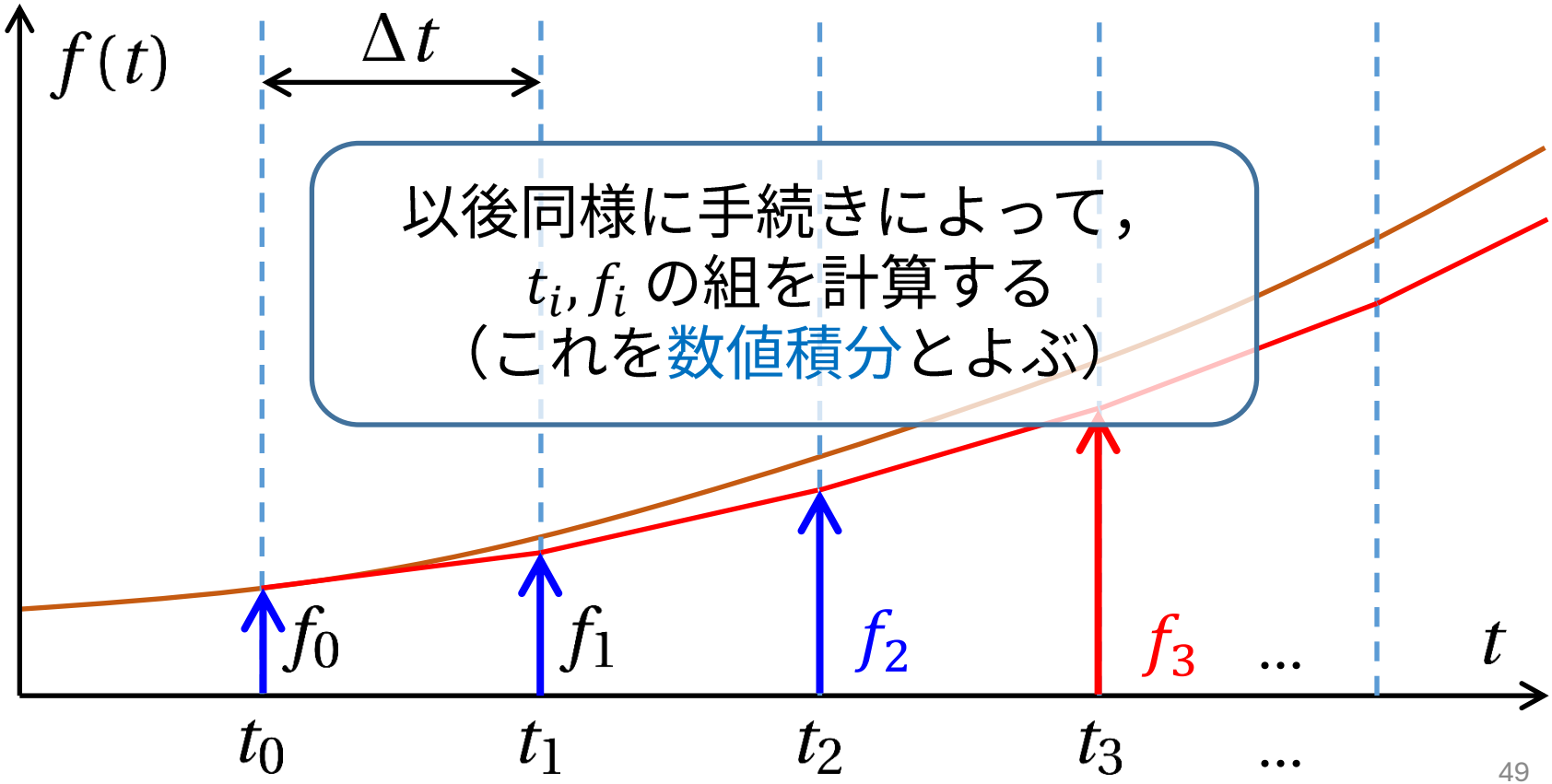
微分方程式を数値的に解く

$$\frac{df}{dt} = G\big(t, f(t)\big), \quad f(t_0) = f_0$$



微分方程式を数値的に解く

$$\frac{df}{dt} = G\big(t, f(t)\big), \qquad f(t_0) = f_0$$



Euler 法

- 最も基本的な数値積分法 (離散化の方法).

1. 微分方程式について，定義から微分を書き下す.

$$\frac{df}{dt} = G\big(t, f(t)\big) \Leftrightarrow \lim_{\tau \to 0} \frac{f(t+\tau) - f(t)}{\tau} = G\big(t, f(t)\big)$$

2. τ を小さい正の数 (刻み幅) Δt に置き換え，
   Δt ごとに時間を区切る.

$$\frac{f_{i+1} - f_i}{\Delta t} = G(t_i, f_i), \quad \Delta t > 0$$

$$\Leftrightarrow f_{i+1} = f_i + G(t_i, f_i)\Delta t$$

CC BY

その他の数値積分法

- Euler 法は誤差の増大が著しいため，
  実用的でない場合もある．

- 有用な数値積分法の例
  - 4 次 Runge-Kutta 法
    - 厳密解に近づくように微分係数をチューニングする
  - シンプレクティック数値積分法
    - エネルギー誤差が一定以下になるように運動方程式を解ける．
    - シンプレクティック Euler 法，リープフロッグ法

CC BY

まとめ

CC BY

まとめ①

計算機が扱うデータ

- データ型を指定することでプログラムのエラーを防ぐ．

計算機における実数の表現

- 計算機は 2 進法の浮動小数点表現で表現する．

計算機が計算処理を行う原理

- 加法は様々な論理回路を組み合わせた半加算器で実現
- 計算機における四則演算は，加法をもとに構成される

CC BY

まとめ②

計算機に計算させる手続き

- 計算機にプログラムを実行させるには，コンパイルをしてプログラムを機械語に翻訳する必要がある．

計算機で微分方程式を解く手続き

- 変数を離散化し，漸化式を解いて次のステップの関数の値を求める (数値積分)
- 数値積分法
  - 最も簡単な例: Euler 法
  - より複雑な方法: 4 次 Runge-Kutta 法，シンプレクティック数値積分法など

参考文献

- 伊理 正夫・藤野 和建，「数値計算の常識」，ISBN 4-320-01343-3，共立出版，1985 年 6 月 1 日．
- 牛島 省，「数値計算のための Fortran90/95 プログラミング入門 (第 2 版)」，ISBN 978-4-627-84722-4，森北出版，2021 年 7 月 28 日．
- 松岡 亮，「シンプレクティック数値積分法」，EPNet FaN座学編，2017 年 4 月 21 日．http://www.ep.sci.hokudai.ac.jp/~epnetfan/zagaku/2017/0421/pub/
- ロン・ホワイト，「ビジュアル版 コンピュータ＆テクノロジー解体新書」，SB クリエイティブ，2016 年 1 月 15 日．
- いらすとや http://www.irasutoya.com