情報実験・第 10 回 (2025/07/04)

数値計算入門

北海道大学大学院 理学院 宇宙理学専攻
修士 2 年
吉川 颯真

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？
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

本日のレクチャー内容

- 数値の表現と誤差
- 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

1. 数値の表現と誤差

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

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」を用いて構成してみる

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

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

「Nの累乗」を用いた和での構成を考えれば, 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 で数値を記述
- 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 で数値を記述（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

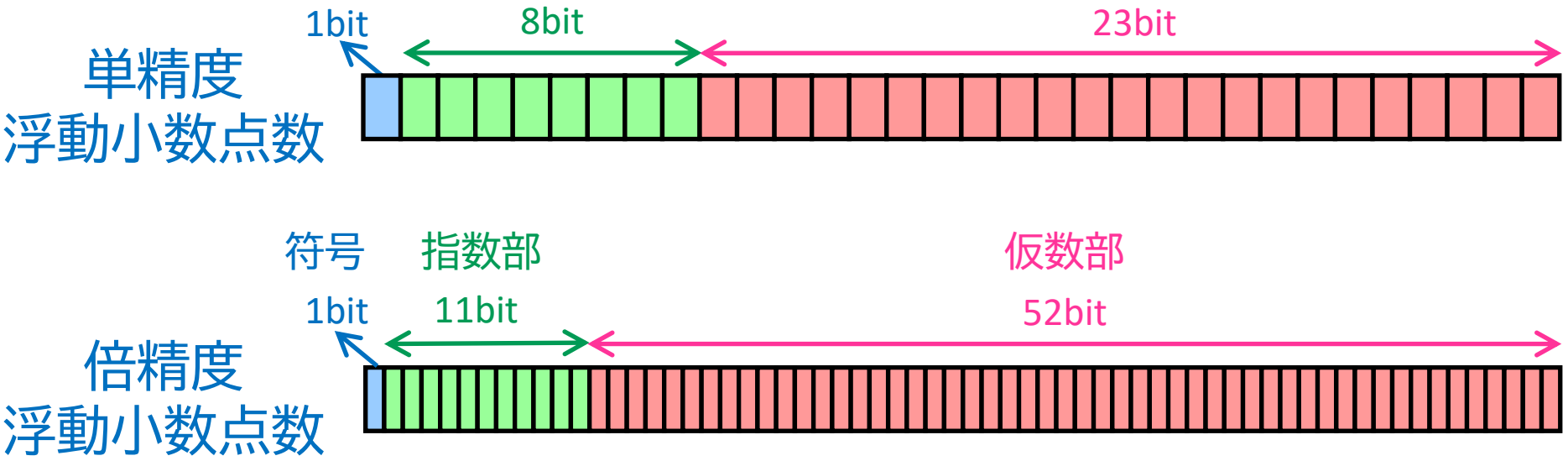
計算機における実数の表現

計算機内部では，実数は2進法の浮動小数点表現で表現される

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

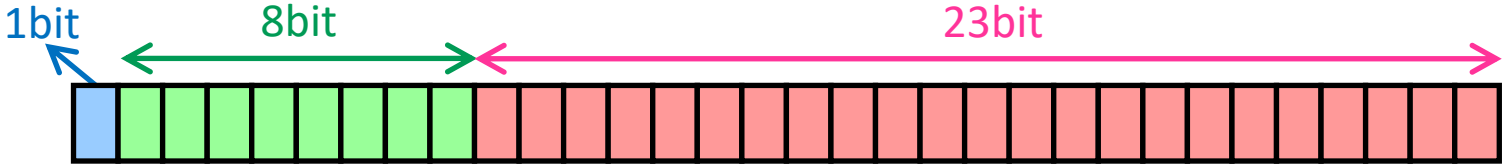
1013.25を単精度浮動小数点数で表現する

1013.25を2進数で表すと, $(1111110101.01)_2$

$(1111110101.01)_2 = +(1.1111010101)_2 \times 2^9$

本来の指数に127を加える $+(1.1111010101)_2 \times 2^{136}$

指数部分を2進数で表す　$+(1.1111010101)_2 \times 2^{(10001000)_2}$



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

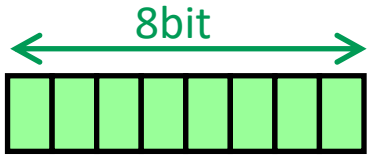
10

10

本

指

符号を用いる場合，
符号＋2進数 で表す
(1bit＋7bit)



＋1111111	＋127
・・・	・・・
＋0000000	＋0
−0000000	−0
・・・	・・・
−1111111	−127
（2進数）	（10進数）

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

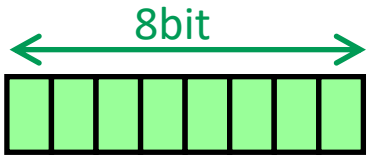
10

10

本

指

符号を用いない場合2進数のみ(8bit)で表す



（2進数）	本来の指数＋127（10進数）	本来の指数（10進数）
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

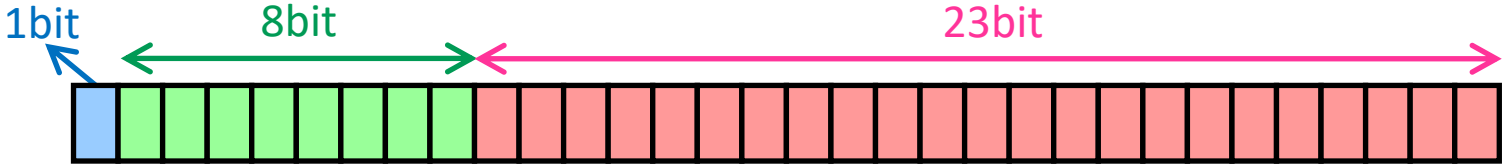
1013.25を単精度浮動小数点数で表現する

1013.25を2進数で表すと, $(1111110101.01)_2$

$(1111110101.01)_2 = +(1.1111010101)_2 \times 2^9$

本来の指数に127を加える $+(1.1111010101)_2 \times 2^{136}$

指数部分を2進数で表す　$+(1.1111010101)_2 \times 2^{(10001000)_2}$



01000100011111010101000000000000

まるめ誤差

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まるめ誤差という

例
3.14159265358979323846を浮動小数点表現で表現する

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

符号　　仮数部　　指数部

まるめ誤差

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まるめ誤差という

例

3.14159265358979323846を2進数で表すと,

$(11.00100100001111110110101010001000100001011010001100 1\cdots)_2$

単精度浮動小数点表現で表現すると

$$\pm(1.a_1a_2a_3\cdots a_{23})_2 \times 2^{(E)_2}$$

符号　仮数部　指数部

まるめ誤差

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まるめ誤差という

例

3.14159265358979323846を2進数で表すと,

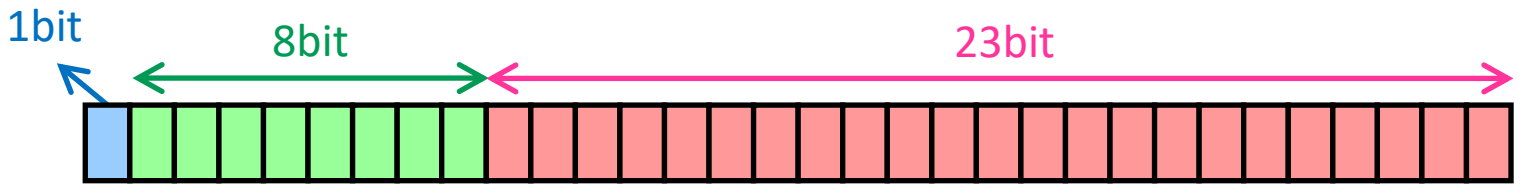
$(11.00100100001111110110101010001000100001011010001100 1\cdots)_2$

単精度浮動小数点表現で表現すると

$$\pm(1.a_1a_2a_3\cdots a_{23})_2 \times 2^{(E)_2}$$

符号　仮数部　指数部

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



01000000100100100001111111011010

まるめ誤差

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まるめ誤差という

例
3.14159265358979323846を2進数で表すと,
$(11.001001000011111101101010100010001000010110100011001\cdots)_2$
単精度浮動小数点表現で表現すると

$$\pm(1.a_1a_2a_3\cdots a_{23})_2 \times 2^{(E)_2}$$

符号　仮数部　指数部

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

10進数へ変換すると,
3.141592502593994 (まるめ誤差:約1.5×10^{-7})

まるめ誤差

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まるめ誤差という

例

3.14159265358979323846を2進数で表すと,

$(11.001001000011111101101010100010001000010110100011001\cdots)_2$

倍精度浮動小数点表現で表現すると

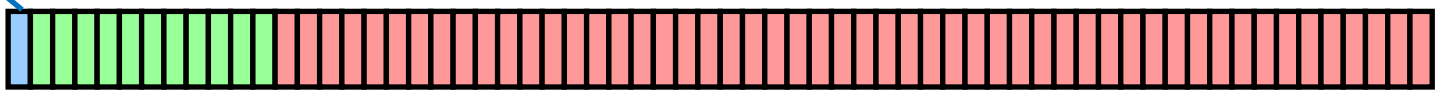
$$\pm(1.a_1a_2a_3\cdots a_{52})_2 \times 2^{(E)_2}$$

符号　仮数部　指数部

$$+(1.1001001000011111101101010100010001000010110100011001)_2$$

1bit　11bit　52bit

$\times 2^{1+1023}$



0100000000001001001000011111101101010100010001000010110100011001

まるめ誤差

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まるめ誤差という

例

3.14159265358979323846を2進数で表すと,

$(11.001001000011111101101010100010001000010110100011001\cdots)_2$

倍精度浮動小数点表現で表現すると

$$\pm(1.a_1a_2a_3\cdots a_{52})_2 \times 2^{(E)_2}$$

符号　仮数部　指数部

$$+(1.1001001000011111101101010100010001000010110100011001)_2 \times 2^{1+1023}$$

10進数へ変換すると,

3.141592653589793<u>6</u>(まるめ誤差:約4.4×10^{-16})

桁落ち

近接した実数同士の減算で，有効数字が減ることを
桁落ちという

例
有効数字8桁で計算することを仮定

$$\sqrt{402} = 20.049938$$
$$\sqrt{401} = 20.024984$$

有効数字が5桁に減ってしまう…

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

桁落ち

近接した実数同士の減算で，有効数字が減ることを
桁落ちという

例
有効数字8桁で計算することを仮定

$$\sqrt{402} = 20.049938$$
$$\sqrt{401} = 20.024984$$

有効数字が5桁に減ってしまう…

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

桁落ちを回避する方法例(近接実数間の減算を回避)

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

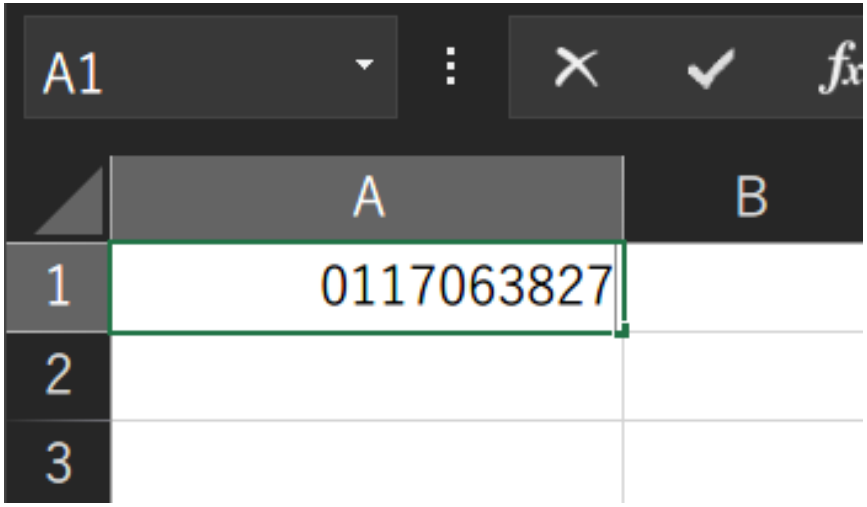
$$= \frac{402 - 401}{\sqrt{401} + \sqrt{402}} = \frac{1}{40.074922} = 2.4953261 \times 10^{-2}$$

データ型

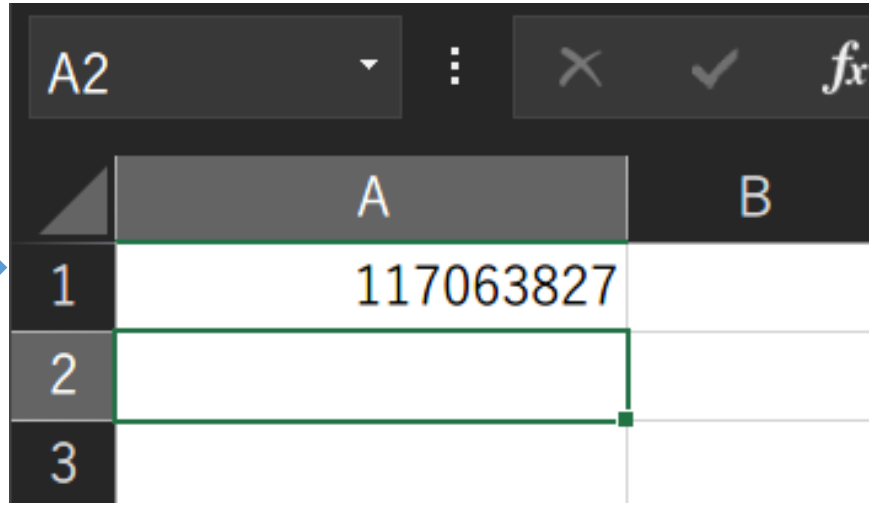
データ型: データの種類のこと

データ型

データ型: データの種類のこと



電話番号を入力



数として解釈される

データ型を指定することは重要！

データ型

データ型: データの種類のこと

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あらかじめ設定しておく必要がある(型の宣言)

あらかじめ定義されているデータ型の例

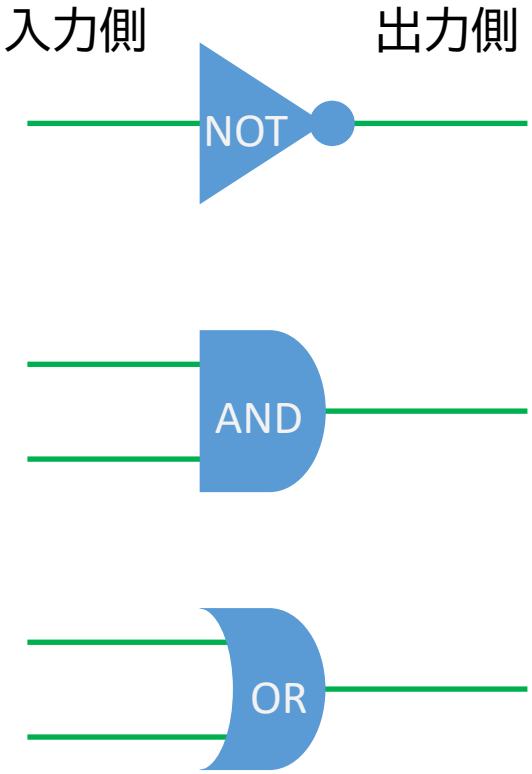
- 整数型 (integer)
- 文字列型 (string)
- 単精度実数型 (float)
- 倍精度実数型 (double)

2. 計算処理の原理

論理回路

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などを組み合わせてつくる. 0, 1は電圧に対応する場合が多い



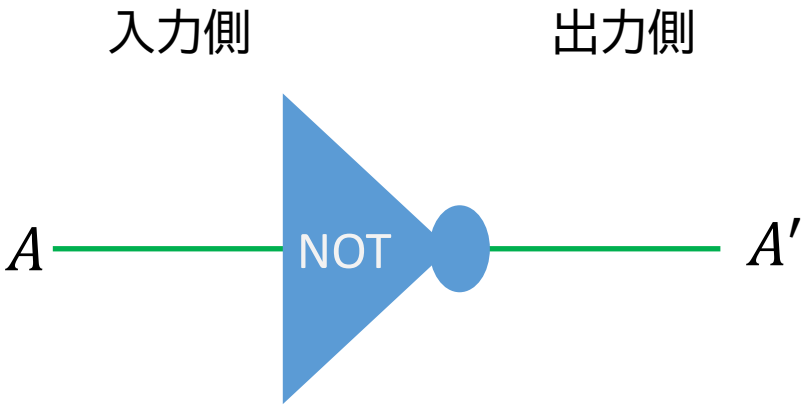
NOT 回路(「否定」に相当)
入力信号の0,1を反転したものを出力する

AND 回路(「かつ」に相当)
入力信号がどちらも1のとき1を出力する

OR 回路(「または」に相当)
入力信号のどちらかが1のとき1を出力する

論理回路

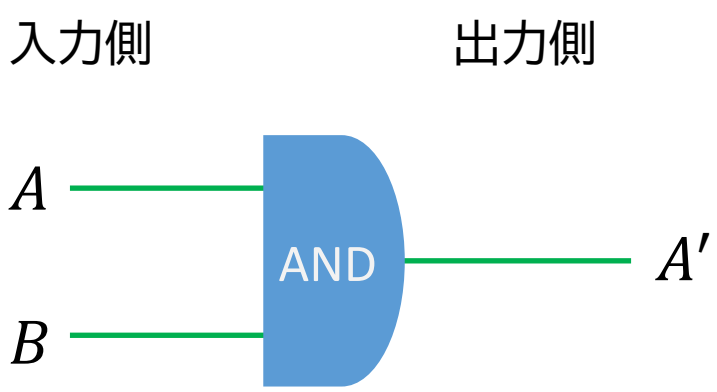
NOT 回路(「否定」に相当)
入力信号の0,1を反転したものを出力する



A	A'
0	1
1	0

論理回路

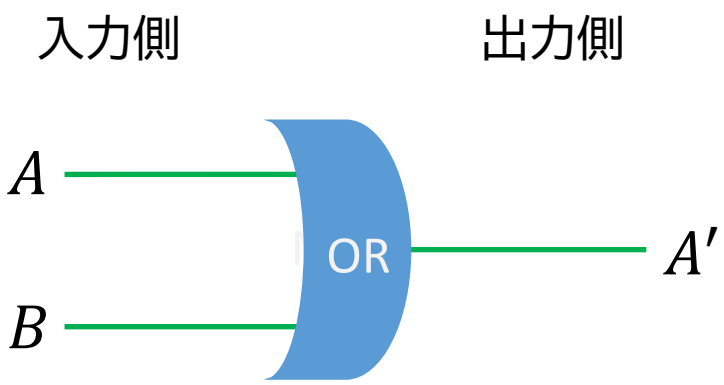
AND 回路(「かつ」に相当)
入力信号がどちらも1のとき1を出力する



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

論理回路

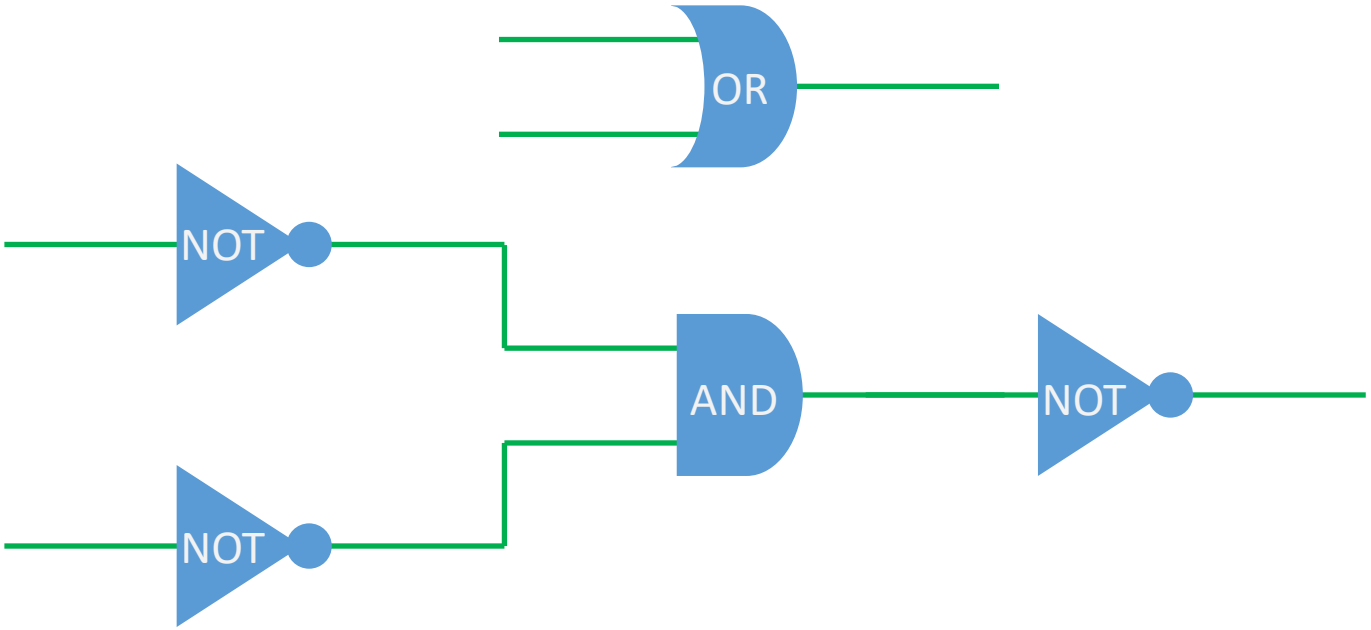
OR 回路(「または」に相当)
入力信号のどちらかが1のとき1を出力する



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

回路の組み合わせ

ある 2 つの論理回路を組み合わせれば,
任意の「0, 1 の組み合わせの変換」を構成できる

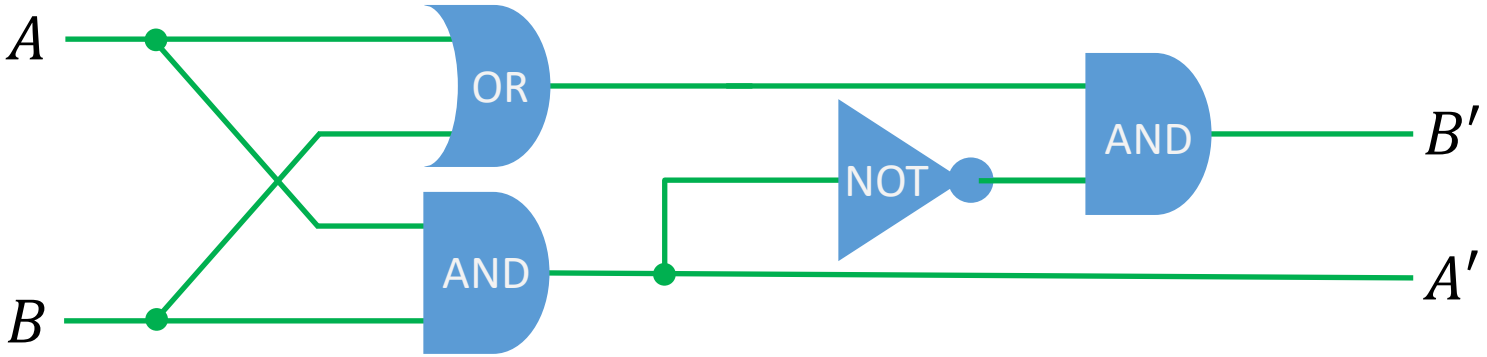


OR 回路を NOT 回路と AND 回路で実装した例

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

加算器(半加算器)

論理回路を組み合わせることで，足し算ができる（加算器）



A		B		A'	B'
0	+	0	=	0	0
0	+	1	=	0	1
1	+	0	=	0	1
1	+	1	=	1	0

色々な演算

四則演算は加法を元に構成できる

減法：補数（足すと桁上がりする最小の数）の加法と
繰り下がり

例

10 進数での引き算 14-6 を補数を用いて表す

10 進数において，6 の補数は 4 であるから，

14-6 = 14+4-10 = 18-10 = 8 (繰り下がり)

色々な演算

四則演算は加法を元に構成できる

減法：補数（足すと桁上がりする最小の数）の加法と
繰り下がり

例

10 進数での引き算 14-6 を 2 進数を用いて表すと

1110-0110

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

0110 → 1001 → 1010

$$\begin{array}{r} 0110 \\ +\ 1010 \\ \hline 10000 \end{array}$$

よって

1110-0110 → 1110+1010 = 11000 → 1000 (繰り下がり)

色々な演算

四則演算は加法を元に構成できる

除法：減法の繰り返し

乗法：AND 回路とビットシフト，加法の組み合わせ

例

10進数での 10×6 を2進数に変換して計算する

$$10\times6 = 60 = (111100)_2$$

10×6 を2進数に変換する

1010×110

1010×1→1010

1010×1→ 1010

1010×0→ 0000

111100

3. プログラムとその実行

プログラム言語

プログラム
計算機への命令を記述したもの
プログラム言語によって記述できる

プログラム言語は高級言語と低級言語の二つに大別

高級言語

- 人間が解釈しやすい抽象的なプログラム言語
- ハードウェアを意識せずにプログラミングができる
- Fortran, C, C++, Java, Lisp, Python, R, Ruby など

低級言語

- 機械語（CPU が直接解釈できる命令データ）と一対一に対応した言語
- ハードウェアを意識したプログラミングが必要
- 機械語そのものとアセンブリ言語が該当

コンパイル

高級言語によるプログラムを実行するには,
機械語に翻訳しなければならない(コンパイル)

コンパイラ
コンパイルを行うプログラム
e.g. GFortran (Fortran), GCC (C言語), G++ (C++) など

※ コンパイラの他にも, プログラムを解釈しながら
実行する処理系も存在する (インタプリタ;第3回参照)

プログラム実行までの流れ （Fortran の場合）

1. コンパイルの実行とプログラム実行ファイルの生成

```
$ gfortran program.f90
```

2. 実行ファイルを実行する

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

4. 数値計算で微分方程式を解く

微分方程式を解くということ

$$\frac{df(t)}{dt} = -tf(t),\ f(0) = 1$$ を解いてみよう

微分方程式を解くということ

$$\frac{df(t)}{dt} = -tf(t),\ f(0) = 1 \text{ を解いてみよう}$$

$$\frac{df(t)}{dt} = -tf(t) \iff \frac{df(t)}{f(t)} = -t\,dt$$

$$\iff \ln|f(t)| = -\frac{t^2}{2} + C_1$$

$$\iff f(t) = \pm\exp\left(-\frac{t^2}{2} + C_1\right)$$

$$\iff f(t) = C\exp\left(-\frac{t^2}{2}\right).$$

$$f(0) = 1 \Rightarrow f(t) = \exp\left(-\frac{t^2}{2}\right).$$

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

ただし，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）：

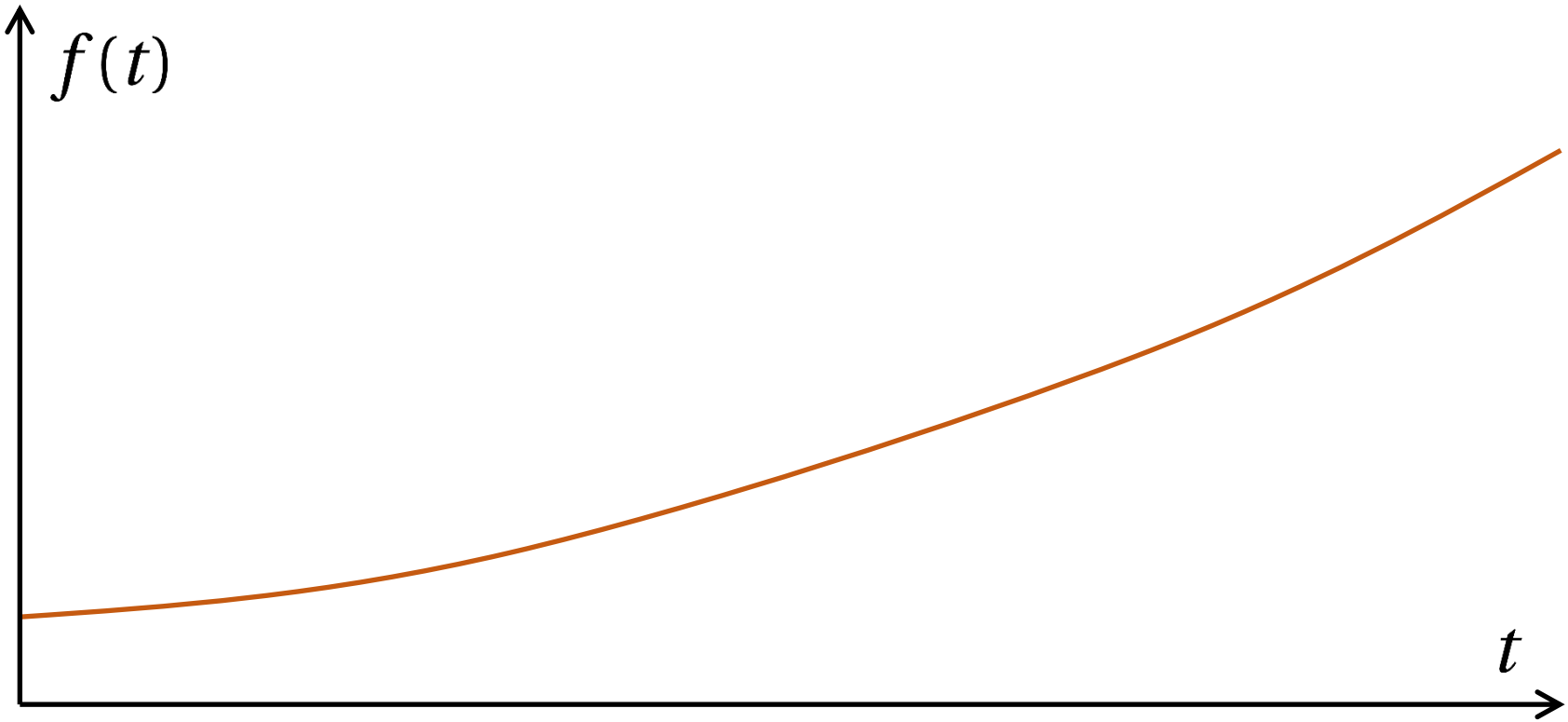
$$\frac{\partial v_i}{\partial t} + \sum_{j=1}^{3} v_j \frac{\partial v_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \sum_{j=1}^{3} \frac{\partial^2 v_i}{\partial {x_j}^2} + f_i(x_1, x_2, x_3, t)$$

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

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

計算機で微分方程式を解く

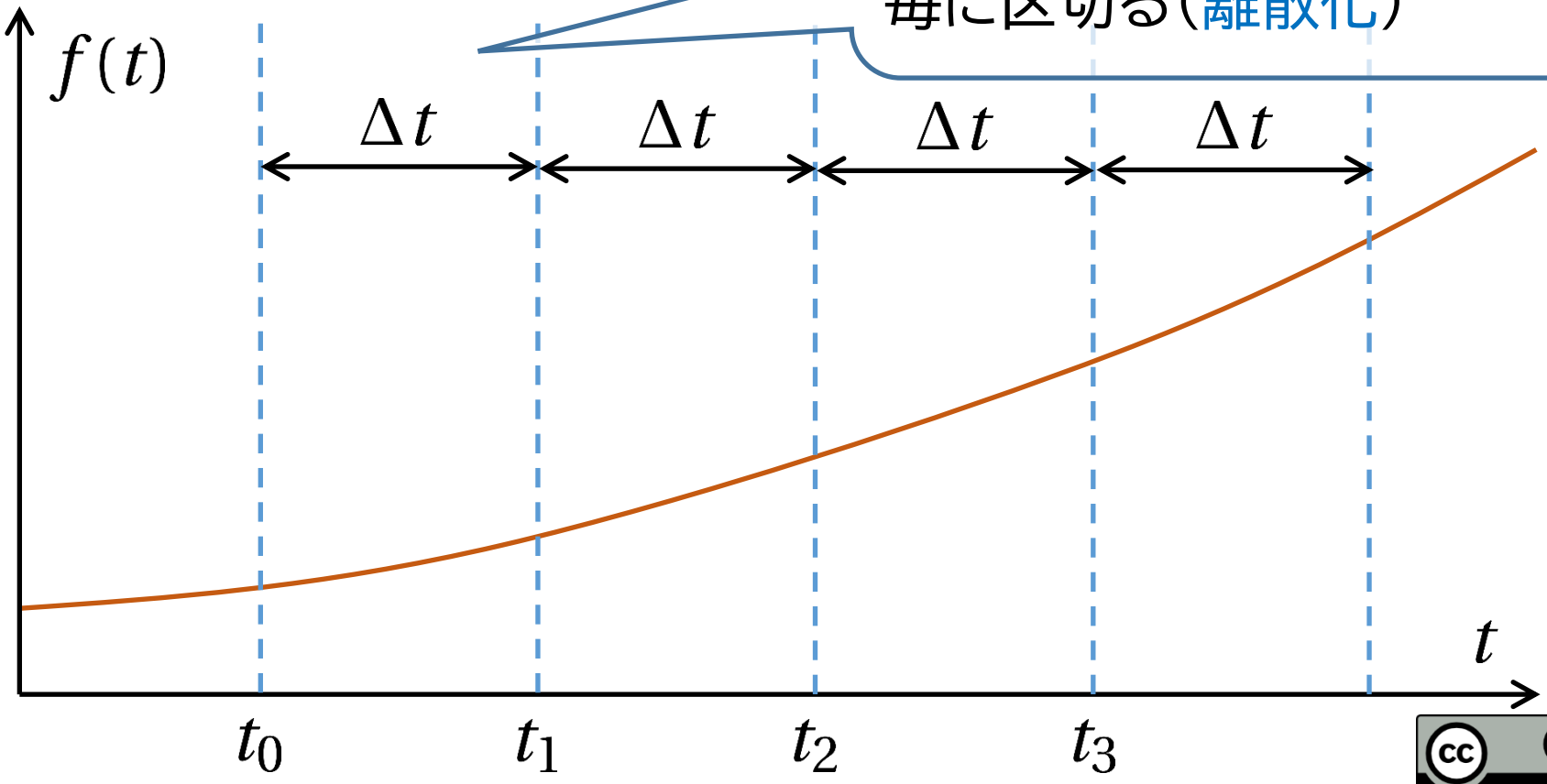
$$\frac{df}{dt} = G(t, f(t)), f(t_0) = f_0$$



計算機で微分方程式を解く

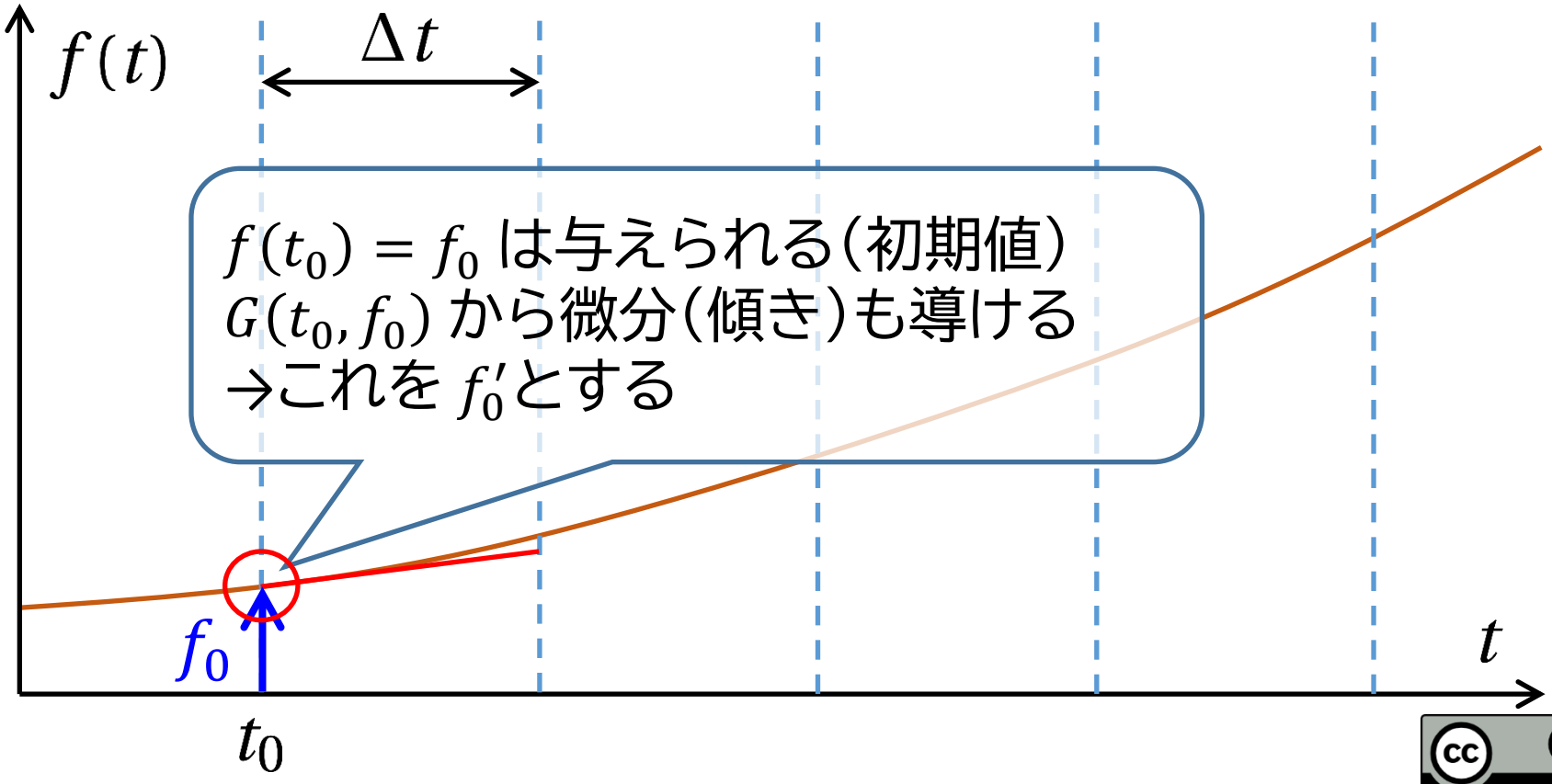
$$\frac{df}{dt} = G(t, f(t)), f(t_0) = f_0$$

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）



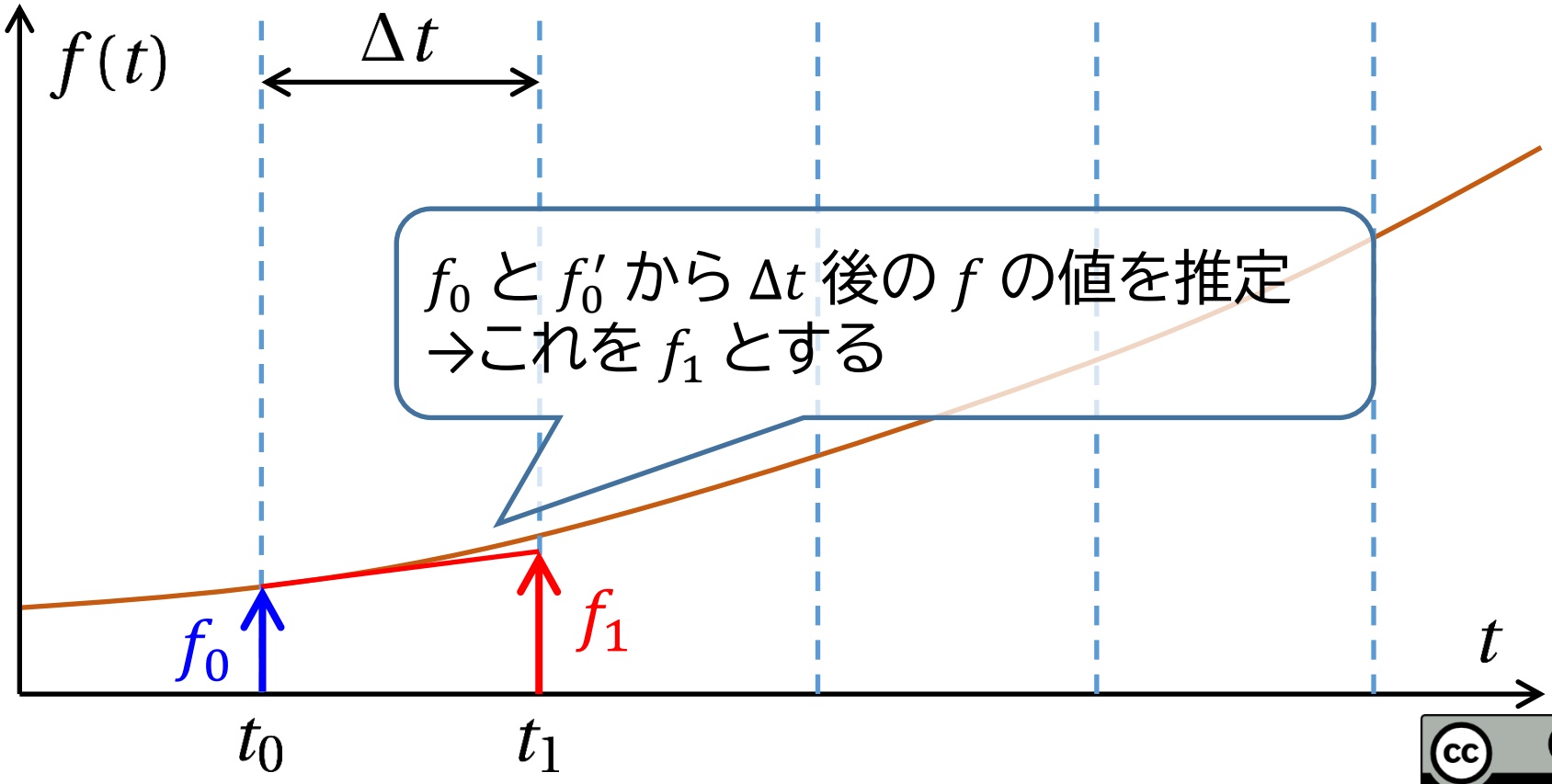
計算機で微分方程式を解く

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



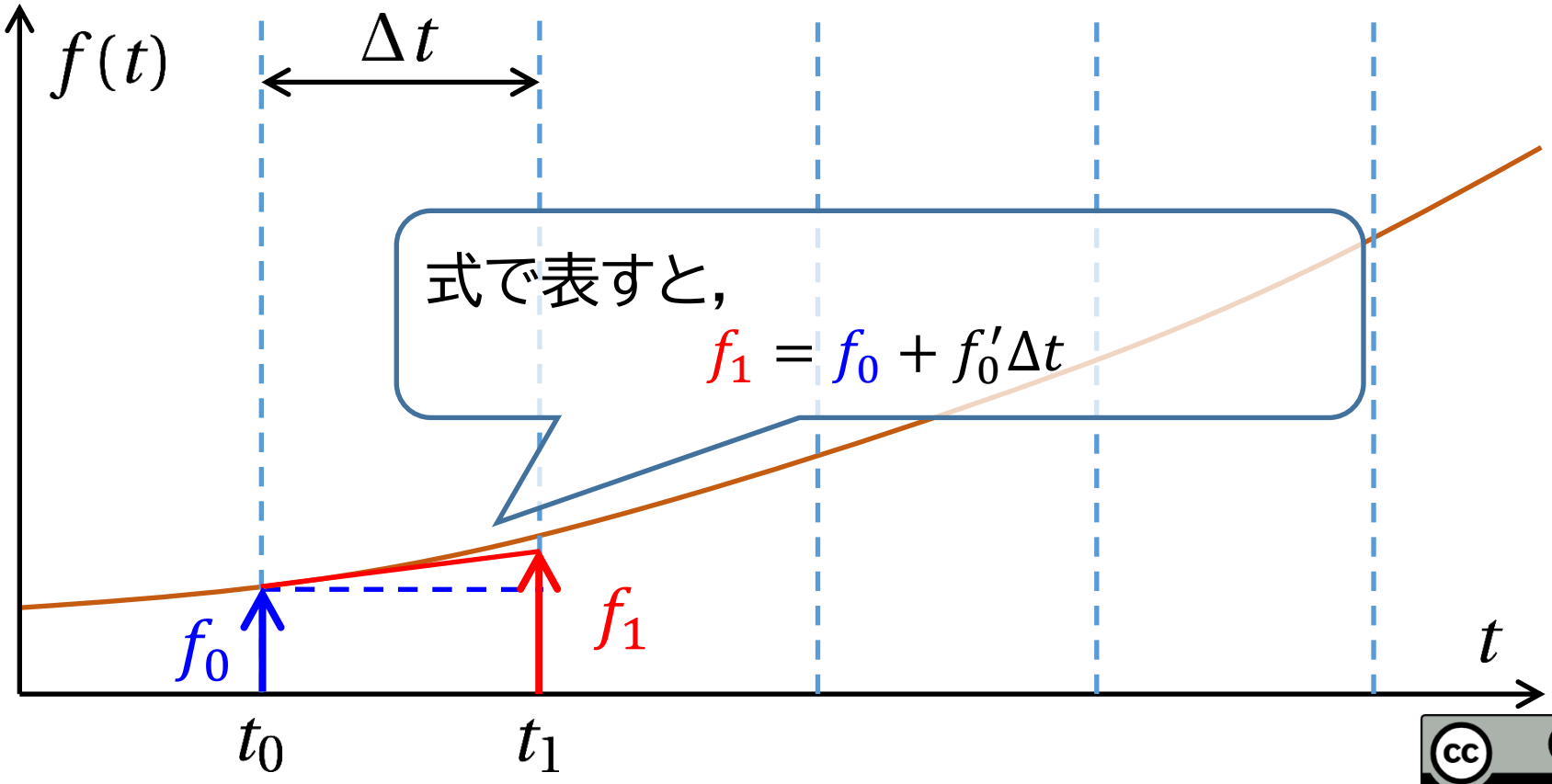
計算機で微分方程式を解く

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



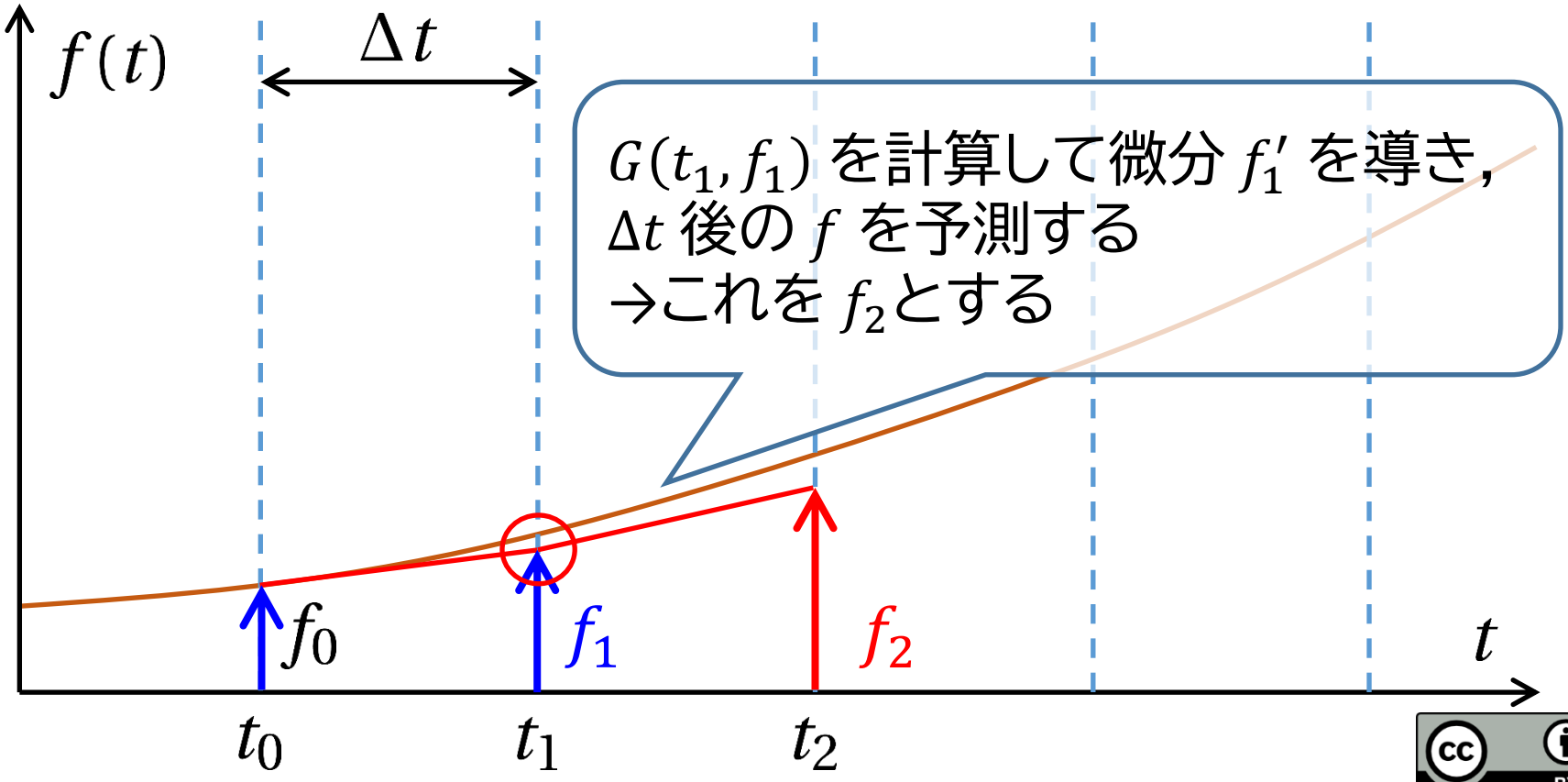
計算機で微分方程式を解く

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



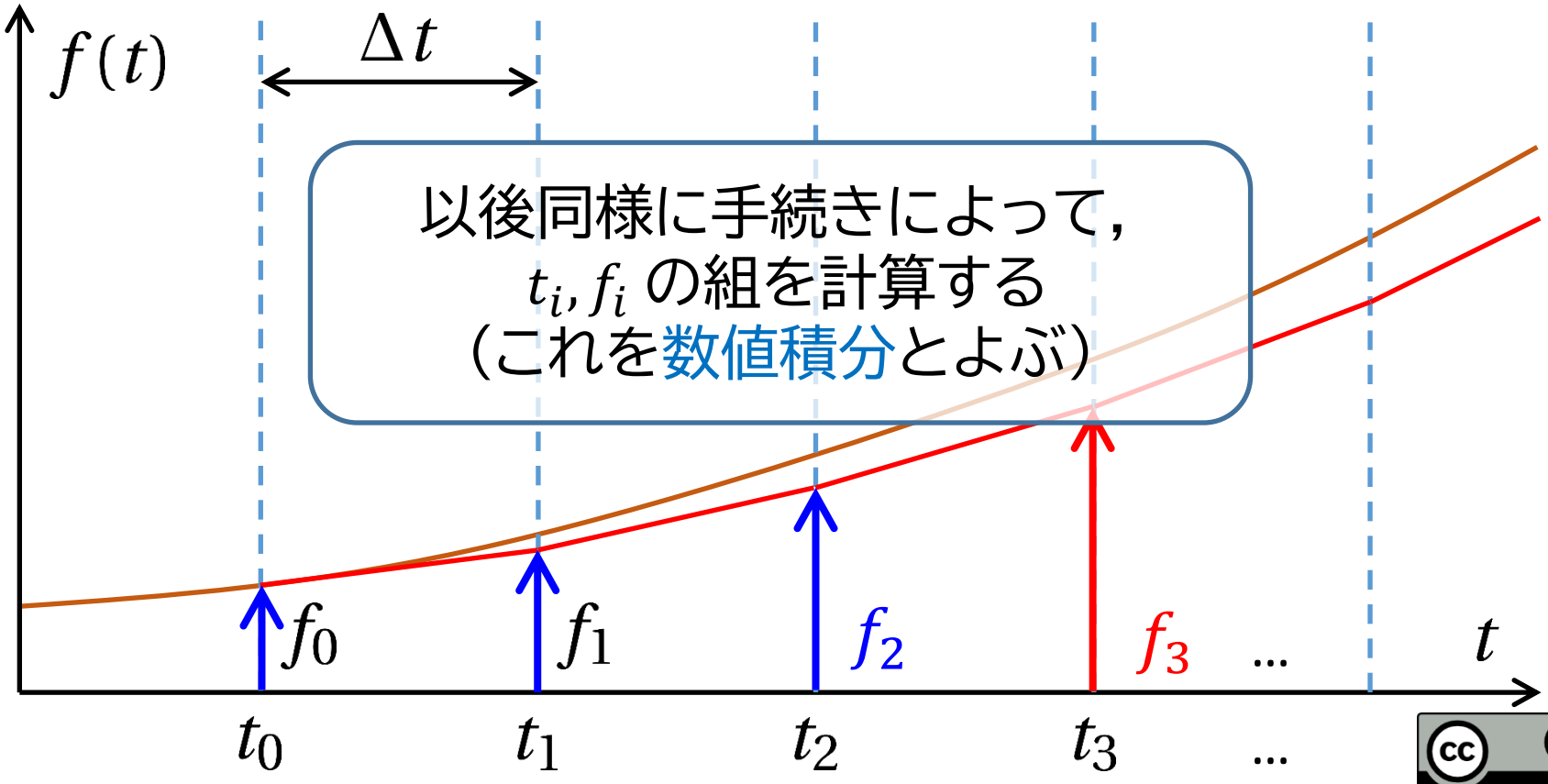
計算機で微分方程式を解く

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



計算機で微分方程式を解く

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



Euler 法

Euler 法 …「微分の定義式」から作られる数値積分法の 1 つ

微分を定義から書き下す

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

τ を小さい正の数 Δt に置き換え, Δt ごとに時間を区切る

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

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

その他の数値積分法

Euler 法は誤差の増大が著しいため, (あまり)実用的ではない

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(傾き)をチューニング

- シンプレクティック数値積分法
  運動方程式を解く際に, エネルギー誤差が一定以下になる.
  e.g. シンプレクティック Euler 法, リープフロッグ法

まとめ

まとめ

計算機が数を扱う方法

- 計算機は 2 進法で数を表現する
- 特に実数は 2 進法の浮動小数点表現で記述する

計算機が計算する原理

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

計算機に計算をさせる手続き

- 計算機に計算を実行させるには, 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

まとめ

計算機で微分方程式を解く手続き

- 変数の離散化を行い, 漸化式を解いて
  次の瞬間の関数の値を求める（数値積分）
- 数値積分の最も簡単な例：Euler 法
  - その他, 4 次 Runge-Kutta 法や
    シンプレクティック数値積分法などがある

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.
- 松岡 亮,「シンプレクティック数値積分法」, EPNetFaN座学編, 2017 年 4 月 21 日.
  http://www.ep.sci.hokudai.ac.jp/~epnetfan/zagaku/2017/0421/pub/
- IT用語辞典 e-words
  http://e-words.jp
- Web で学ぶ 情報処理概論「半加算器」
  http://www.infonet.co.jp/ueyama/ip/logic/adder.html
  2024 年 7 月 11 日閲覧
- いらすとや
  http://www.irasutoya.com