情報実験・第10回（2018/07/06）

数値計算入門

北海道大学大学院 理学院 宇宙理学専攻
博士課程 1年
松岡 亮／Matsuoka Ryo

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

人間が使う数と計算機が使う数

人間が使う数と計算機が使う数

人はなぜ「1013.25」を「1013.25」と表現するのだろう

人間が使う数と計算機が使う数

人はなぜ「1013.25」を「1013.25」と表現するのだろう

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

人間が使う数と計算機が使う数

人はなぜ「1013.25」を「1013.25」と表現するのだろう

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

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（10進法）

人間が使う数と計算機が使う数

人はなぜ「1013.25」を「1013.25」と表現するのだろう

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

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（10進法）

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

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.

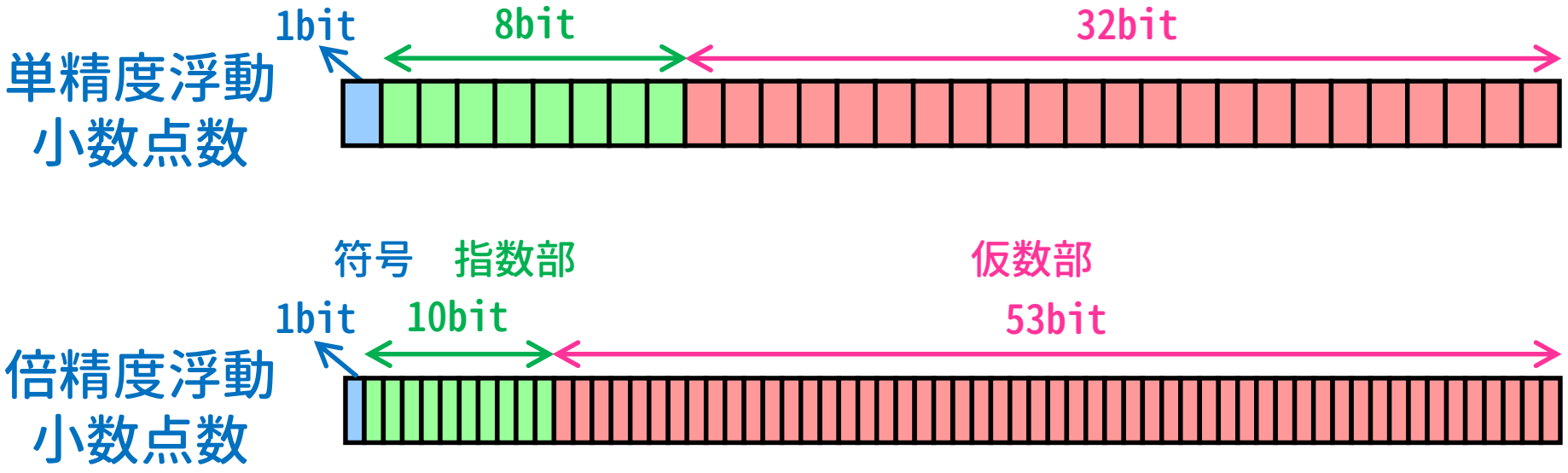
計算機における実数の表現

浮動小数点表現…符号，仮数部，指数部で実数を表現

$$\pm(0.a_1 a_2 a_3 \cdots a_k)_N \times N^{\pm(E)_N}$$

符号　仮数部　指数部

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



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を浮動小数点数で表現すると，

01000000010010010000111111011010（単精度）
0100000000001001001000011111101101010100010001000010110100011001（倍精度）

これらの数値を10進数へ変換すると，

単精度：3.141592502593994　（まるめ誤差：約1.5×10^{-7}）
倍精度：3.141592653589793 6（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）

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

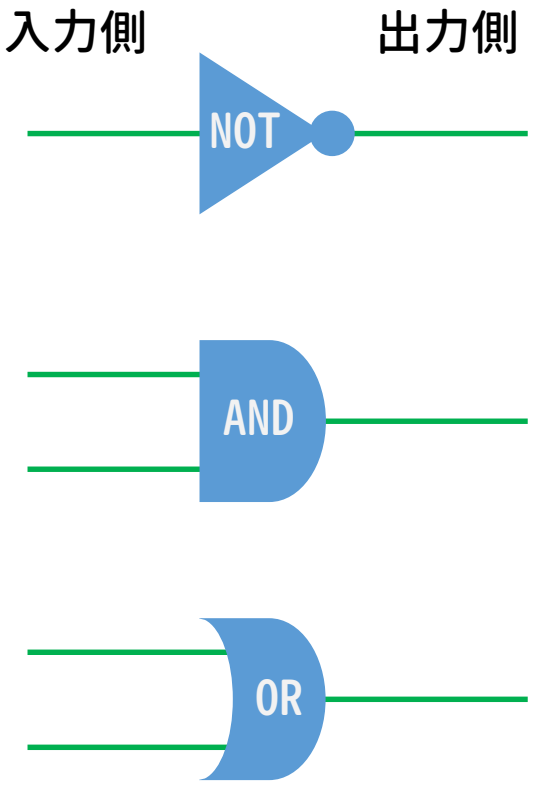
$$= \frac{402 - 401}{\sqrt{401} + \sqrt{402}} = \frac{1}{40.074922} = 2.4953261 \times 10^{-2}$$

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．



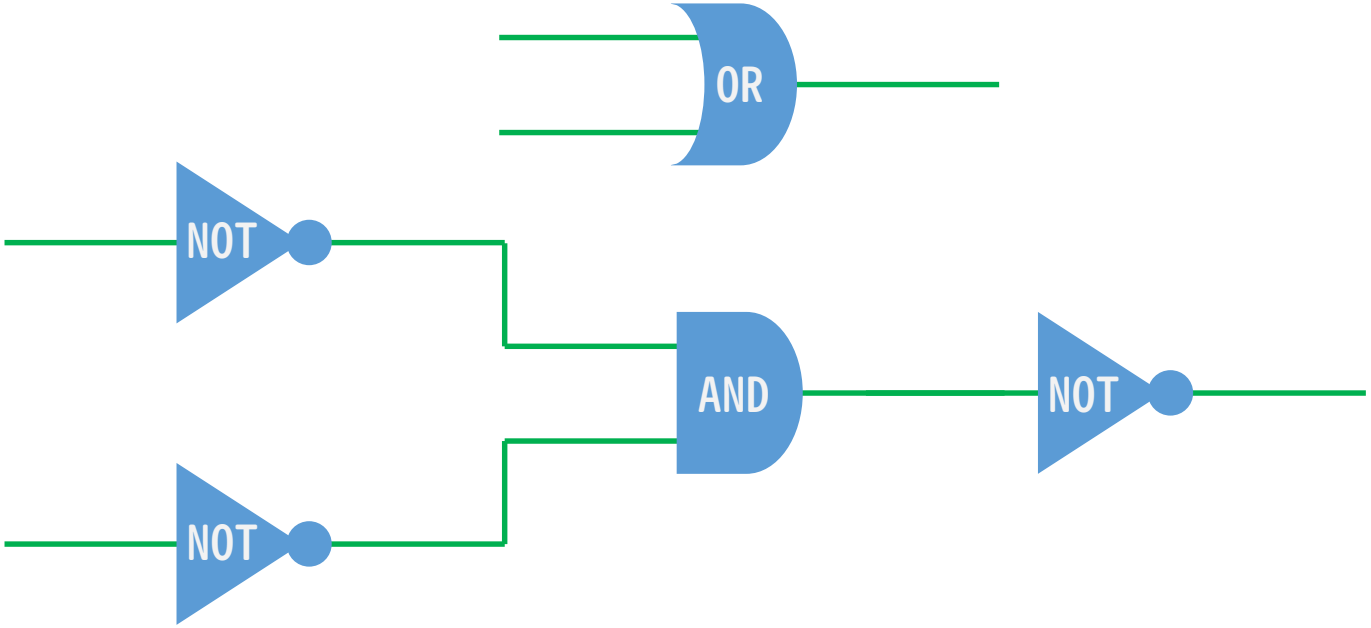
NOT回路（「否定」に相当）
入力信号の0,1を反転したものを出力する

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

回路の組み合わせ

ある2つの論理回路（例えばNOT回路とAND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．

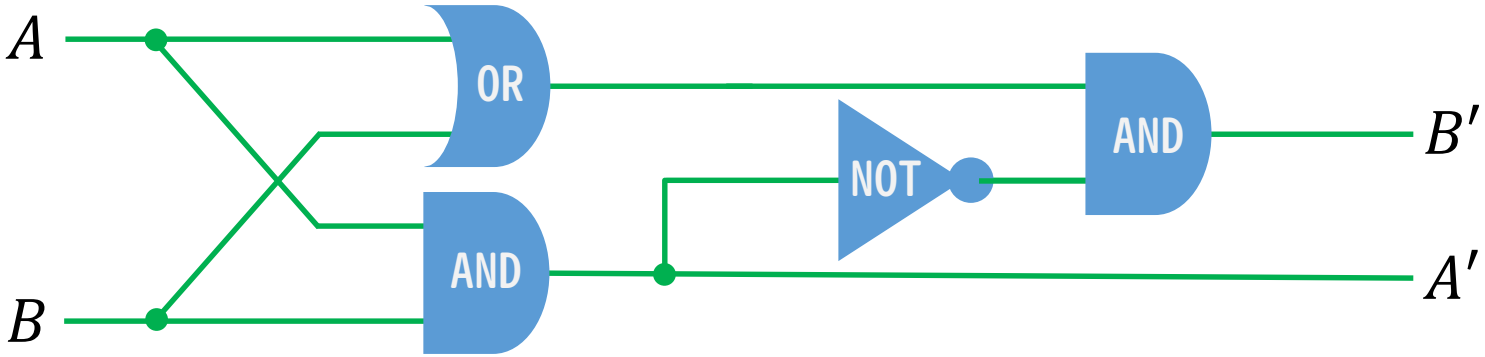


OR回路をNOT回路とAND回路で実装した例

（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数）の加法と繰り下がり
e.g. 25-6 → 25+4 = 29 → 19 (繰り下がり)

乗法：AND回路とビットシフト，加法の組み合わせ
e.g.
1010×110（10×6）

1010×1→1010
1010×1→ 1010
1010×0→ 0000
111100（60）

除法：減法の繰り返し

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 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できる
- 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

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

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

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

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

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

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

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）：

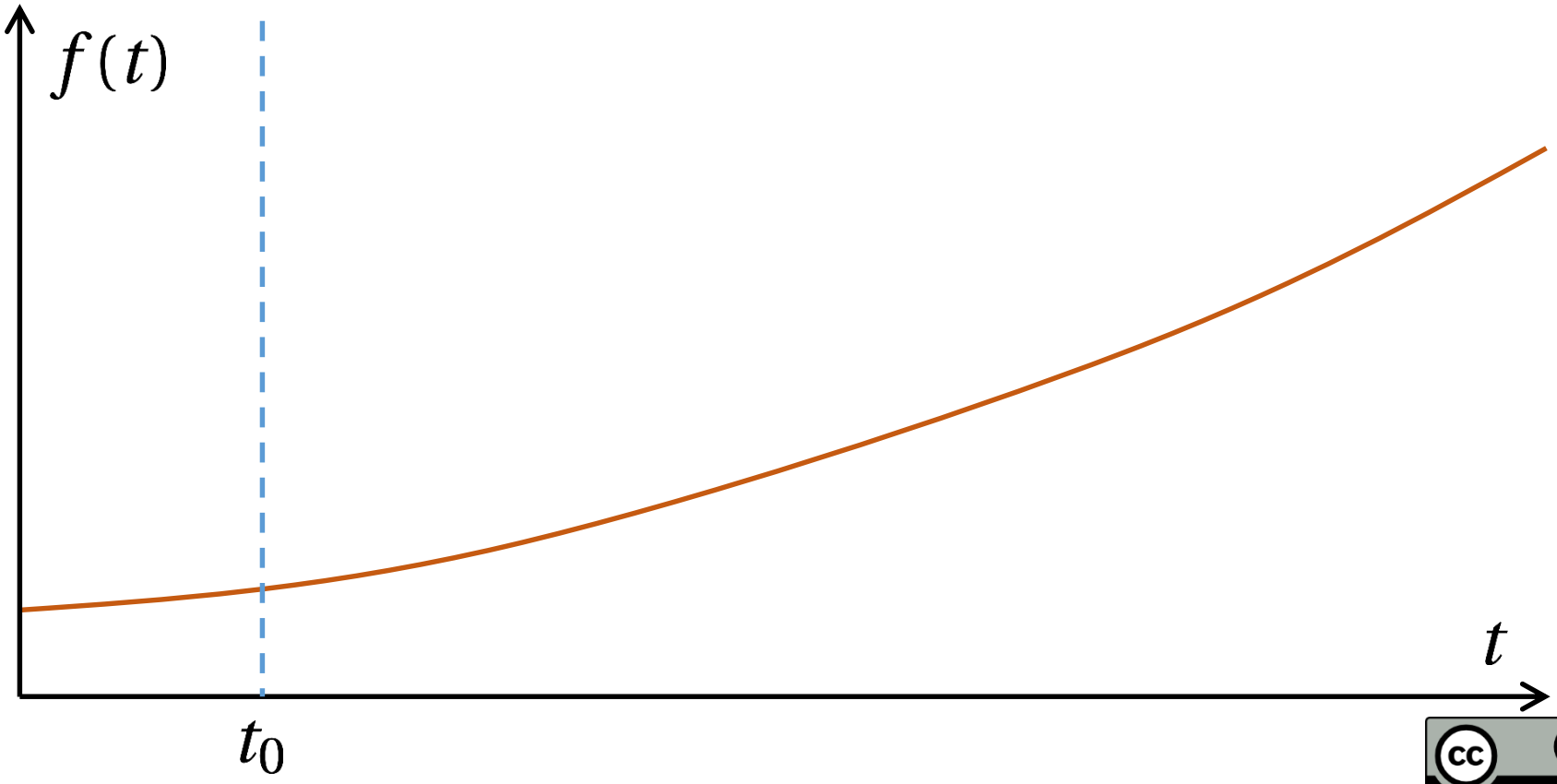
$$\frac{\partial v_i}{\partial t} + \sum_{j=1}^{3} v_j \frac{\partial v_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \sum_{j=1}^{3} \frac{\partial^2 v_i}{\partial {x_j}^2} + f_i(x_1, x_2, x_3, t)$$

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

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解く

計算機で微分方程式を解く

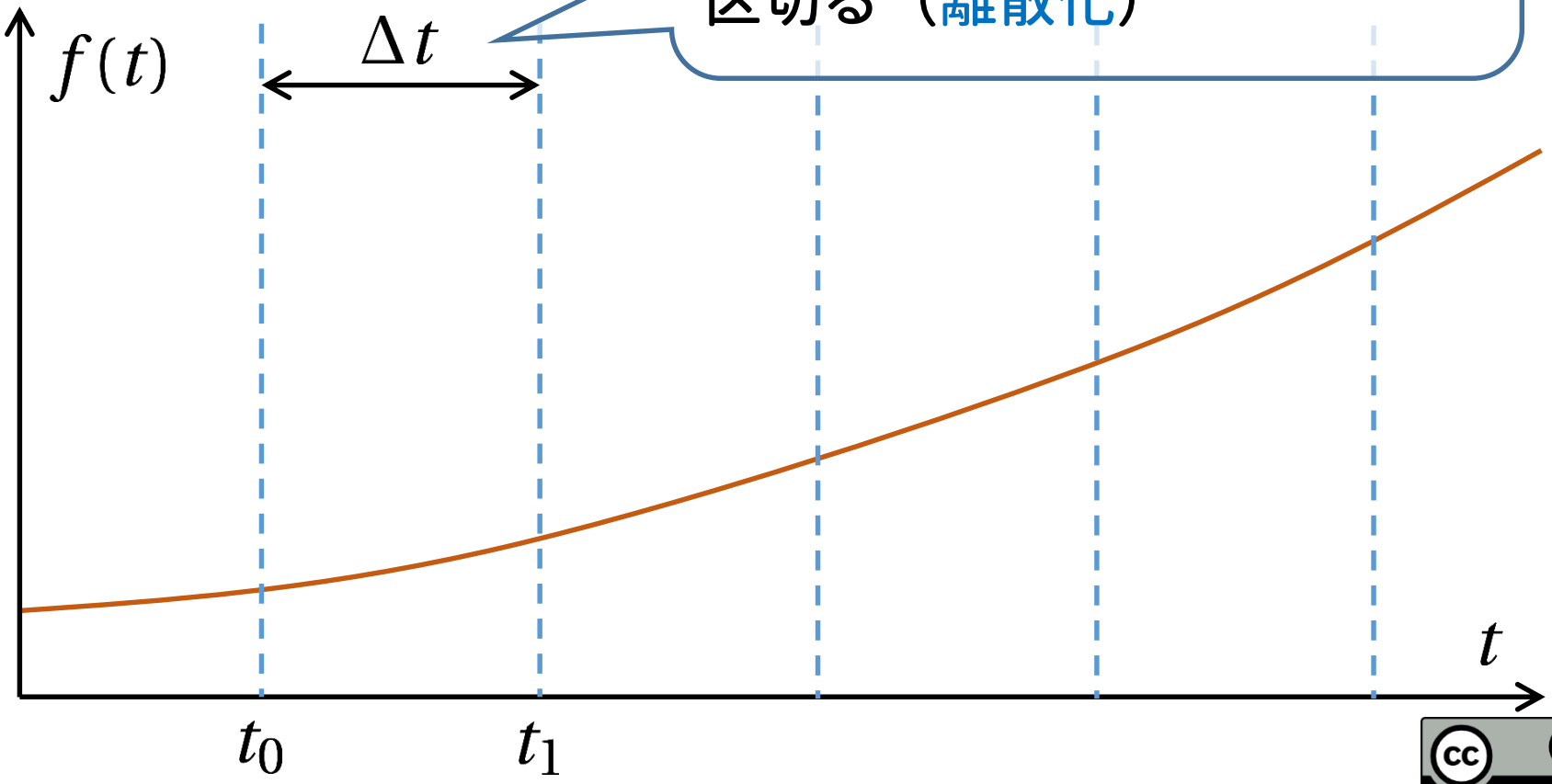
$$\frac{df}{dt} = G(t, f(t))$$



計算機で微分方程式を解く

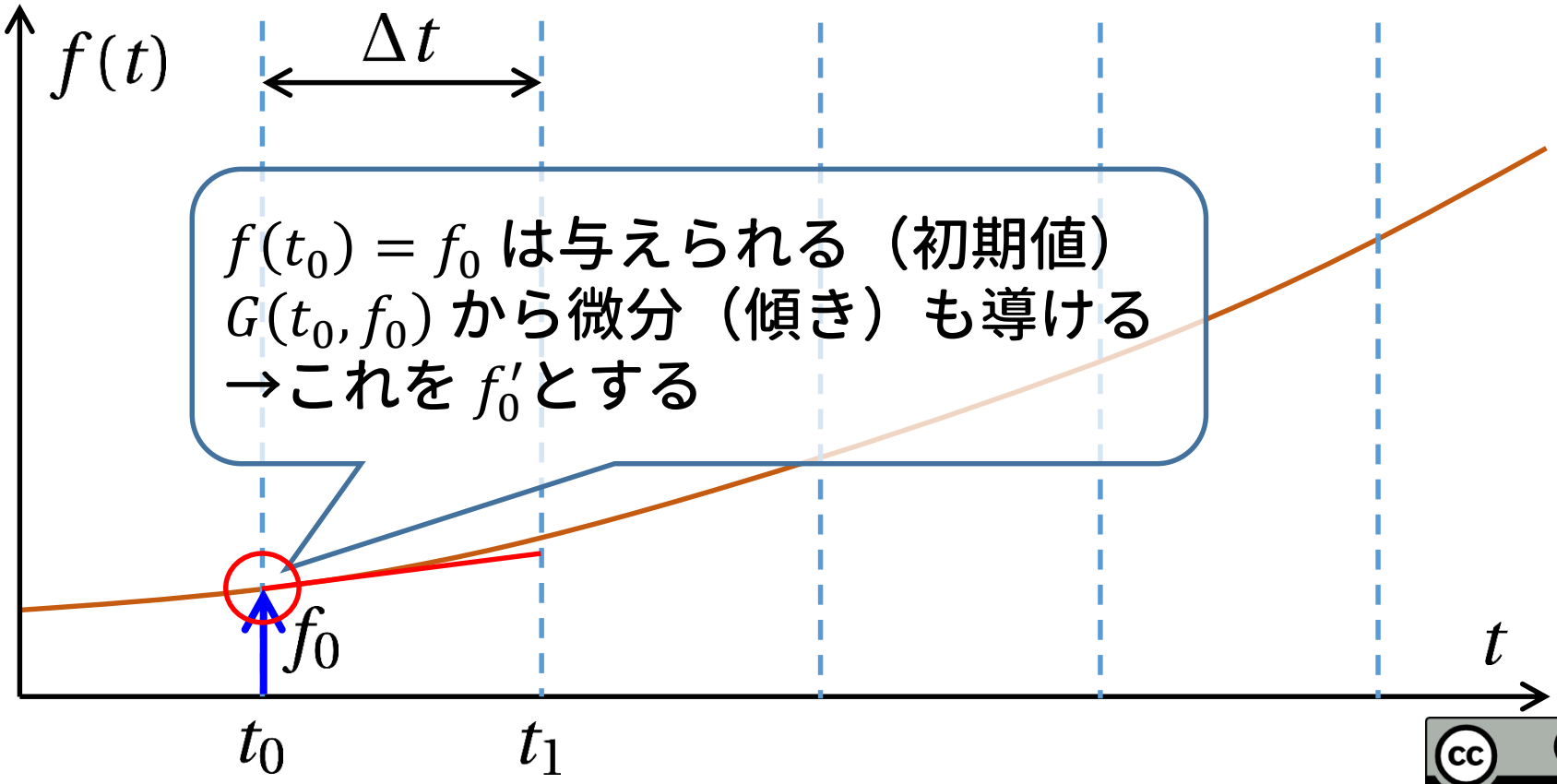
$$\frac{df}{dt} = G(t, f(t))$$

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）



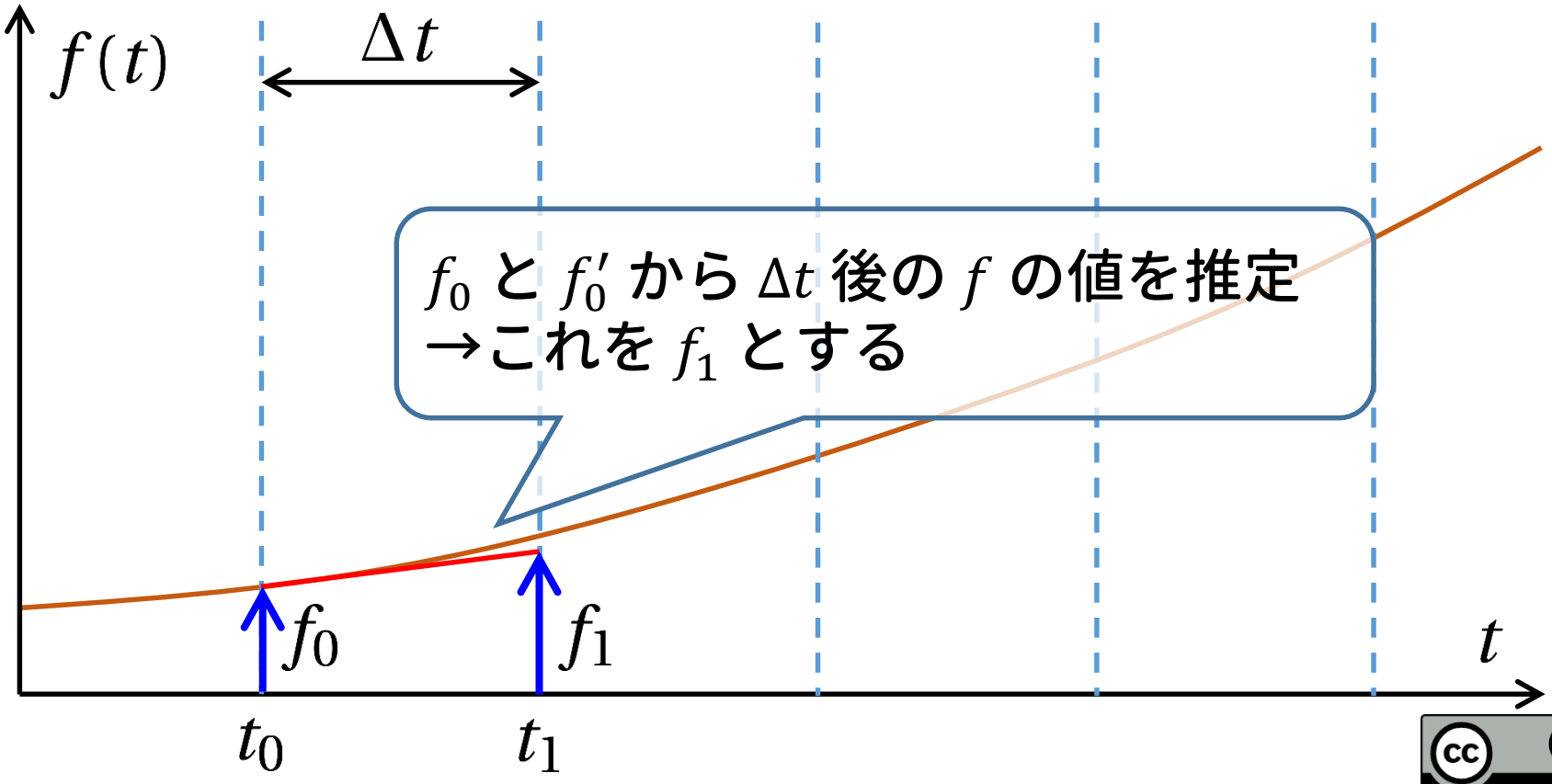
計算機で微分方程式を解く

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



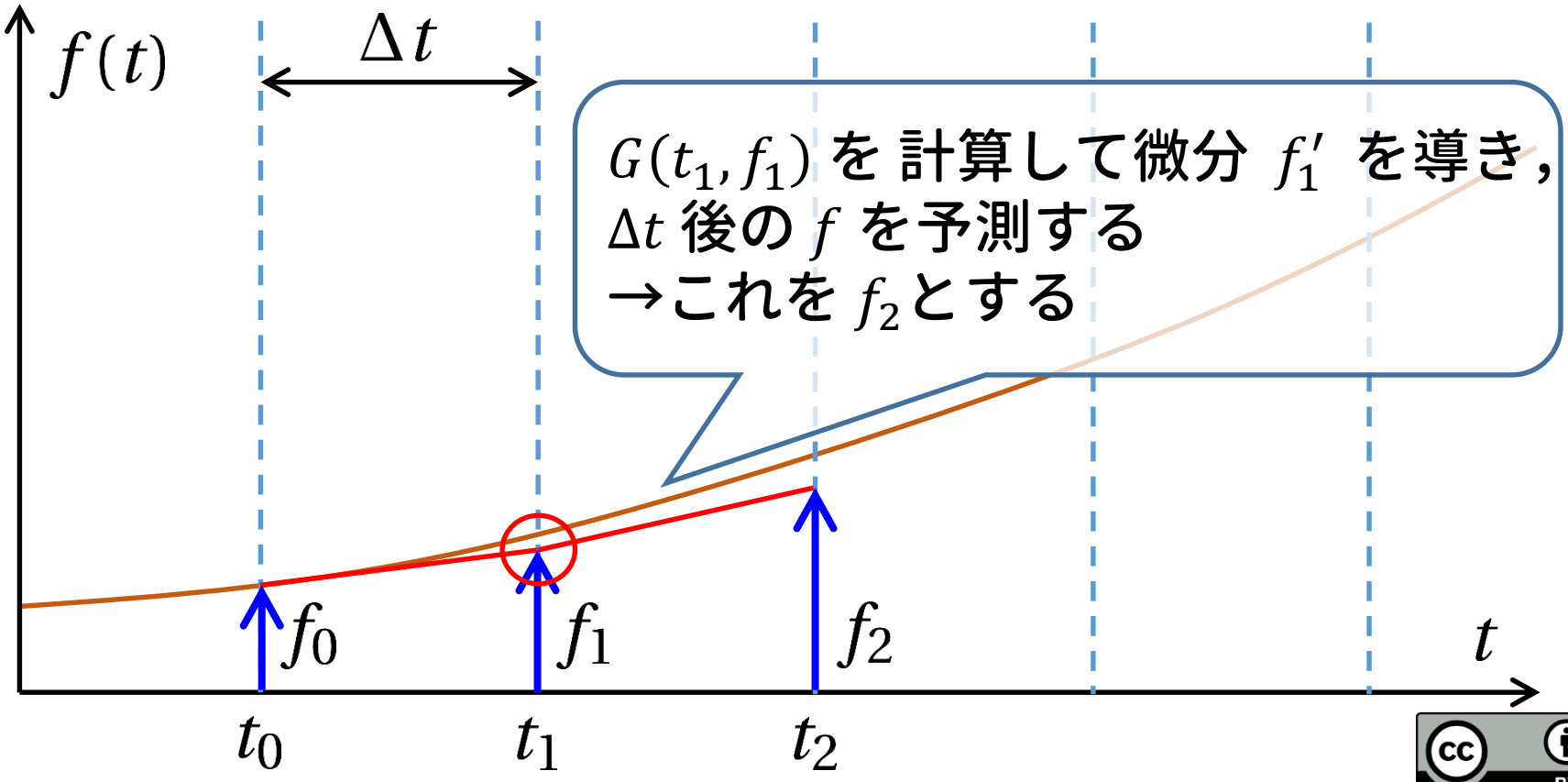
計算機で微分方程式を解く

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



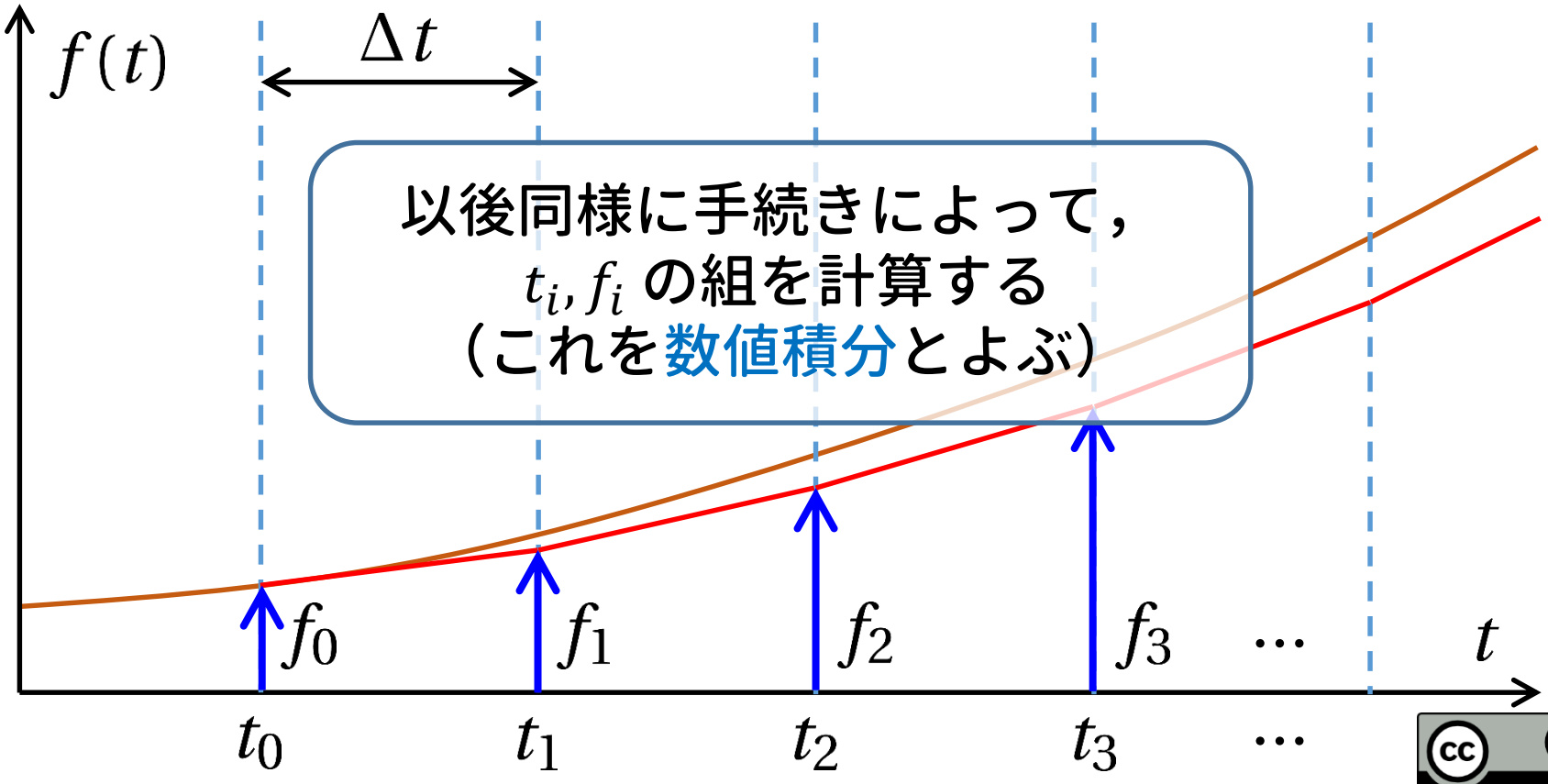
計算機で微分方程式を解く

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



計算機で微分方程式を解く

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



Euler法

Euler法 …「微分の定義式」から作られる数値積分法

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

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

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

様々な数値積分法

Euler法は誤差の増大が著しいため，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
  2018年7月3日閲覧