プログラミングで数学を楽しむ会(5)

中村晃一 (nineties)

東京大学情報理工学系研究科

1 **月** 20 **日**

本日の内容

常微分方程式の数値解法

- 浮動小数点数と誤差について
- 常微分方程式の数値解法
- 簡単な物理シミュレーション

お断り

- 前回「数値流体解析がしたい」というリクエストを頂きましたが、さすがに一日でそこまで行くのは大変なので複数回に分けます。
- 数学の初心者の方も居ますので、数学的に難しい話題は割愛しています。

数値解法とは

方程式の解や無限級数の値などを近似的に求める事。

例

2 次方程式 $x^2 - 2 = 0 \quad (x > 0)$ の解は代数的に求めると

$$x = \sqrt{2}$$

である。
一方数値的に (精度 5 桁で) 求めると

$$x \approx 1.4142$$

である。

何故数値解法が必要か？

代数的・解析的に解くことが困難もしくは不可能な問題が多数ある。

- 無理数の各桁の値を求める事。
  - 例えば $\pi = 3.141592653589\cdots$
- 超越方程式の解を求める事。
  - 例えば $x = \cos x$
- 物理シミュレーションの問題の多く。
  - 例えば多体問題は一般には解析的に解けない。

など。

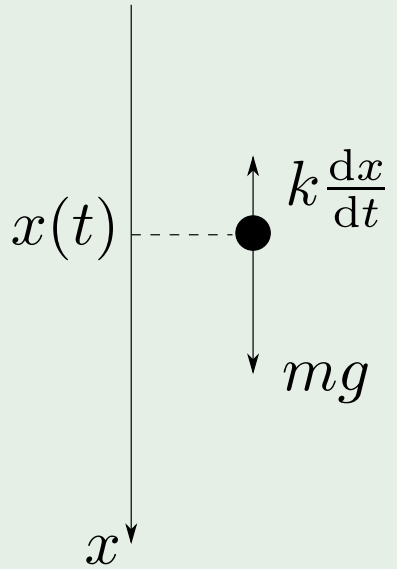
常微分方程式とは

未知関数 $x(t)$ とその導関数 $\left(\frac{\mathrm{d}x}{\mathrm{d}t}, \frac{\mathrm{d}^2x}{\mathrm{d}t^2}, \cdots\right)$ からなる、変数が一つの方程式の事。

例

速度に比例する抵抗 (比例定数 k) を受けながら落下する落体 (質量 m) の運動は、以下の常微分方程式で表される (g は重力加速度)。

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = mg - k\frac{\mathrm{d}x}{\mathrm{d}t}$$

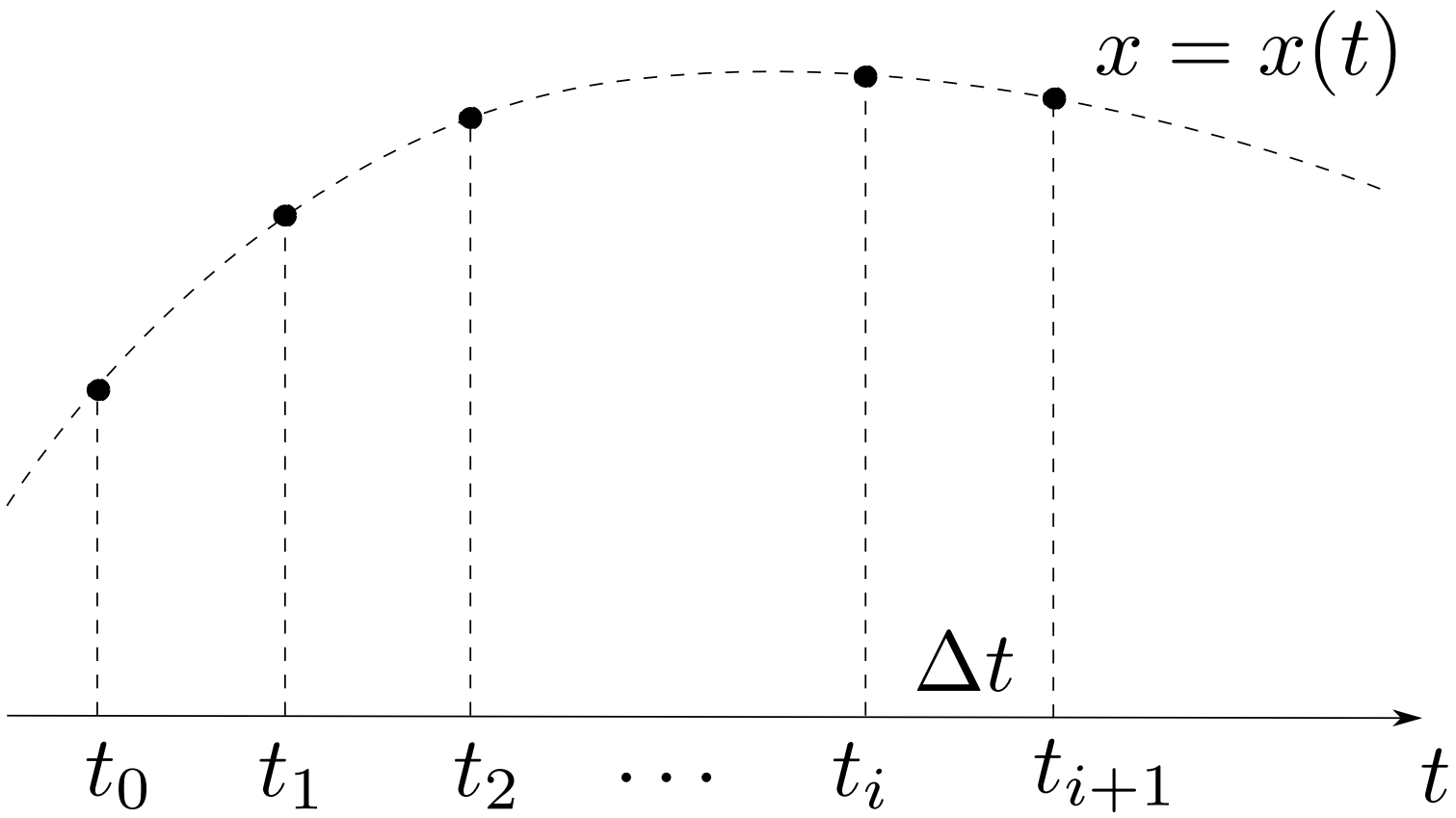


この方程式とある初期条件 $x(t_0) = x_0$ を満たす $x(t)$ が落体の運動を表す。

常微分方程式 (の初期値問題) は常に一意な解を持つわけではなく、リプシッツ連続性などの条件を考える必要があるが、この講義ではその説明を割愛する。

常微分方程式の数値解とは

微分方程式の解 $x(t)$ を、飛び飛びの $t = t_0, t_1, \cdots$ について求める。
以後、話を簡単にする為に t_i の間隔は一定値 Δt であるとする。



常微分方程式の数値解法の原理

微積分学の基本定理

$\dfrac{\mathrm{d}x}{\mathrm{d}t} = f(t)$ **であるならば**

$$x_{i+1} = x_i + \int_{t_i}^{t_{i+1}} f(t)\mathrm{d}t$$

である。(ただし $x_i = x(t_i)$ **と略記する。)**

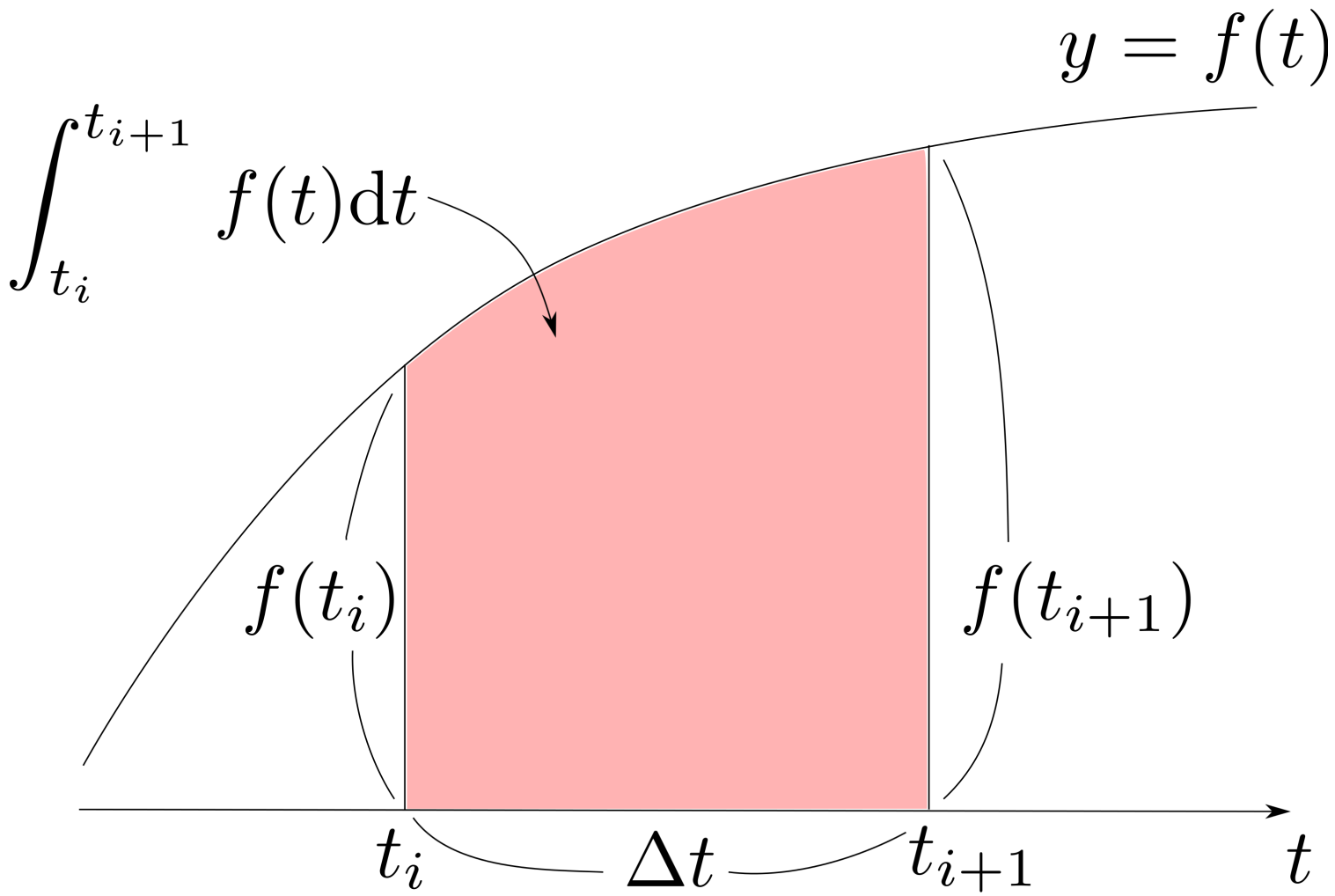
つまり定積分

$$\int_{t_i}^{t_{i+1}} f(t)\mathrm{d}t$$

を計算する事が出来れば、与えられた初期値 x_0 **から始めて、順番に** $x_1, x_2, \cdots$ **を求める事が出来る。**

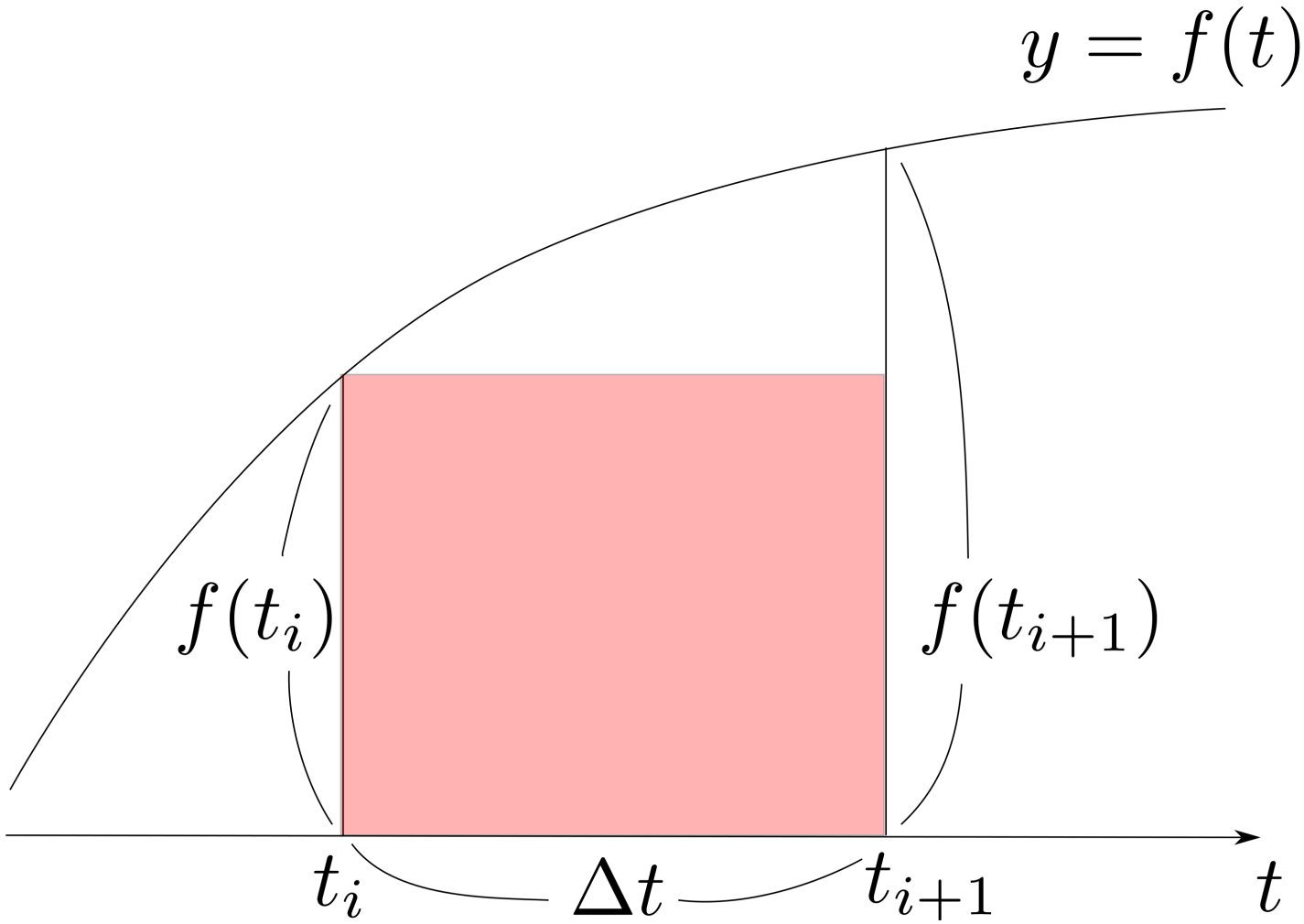
$\int_{t_i}^{t_{i+1}} f(t)\mathrm{d}t$ を求めるには

$\int_{t_i}^{t_{i+1}} f(t)\mathrm{d}t$ の値は以下の領域の面積として解釈する事が出来る。
この面積を何らかの方法によって近似して求めれば良い。



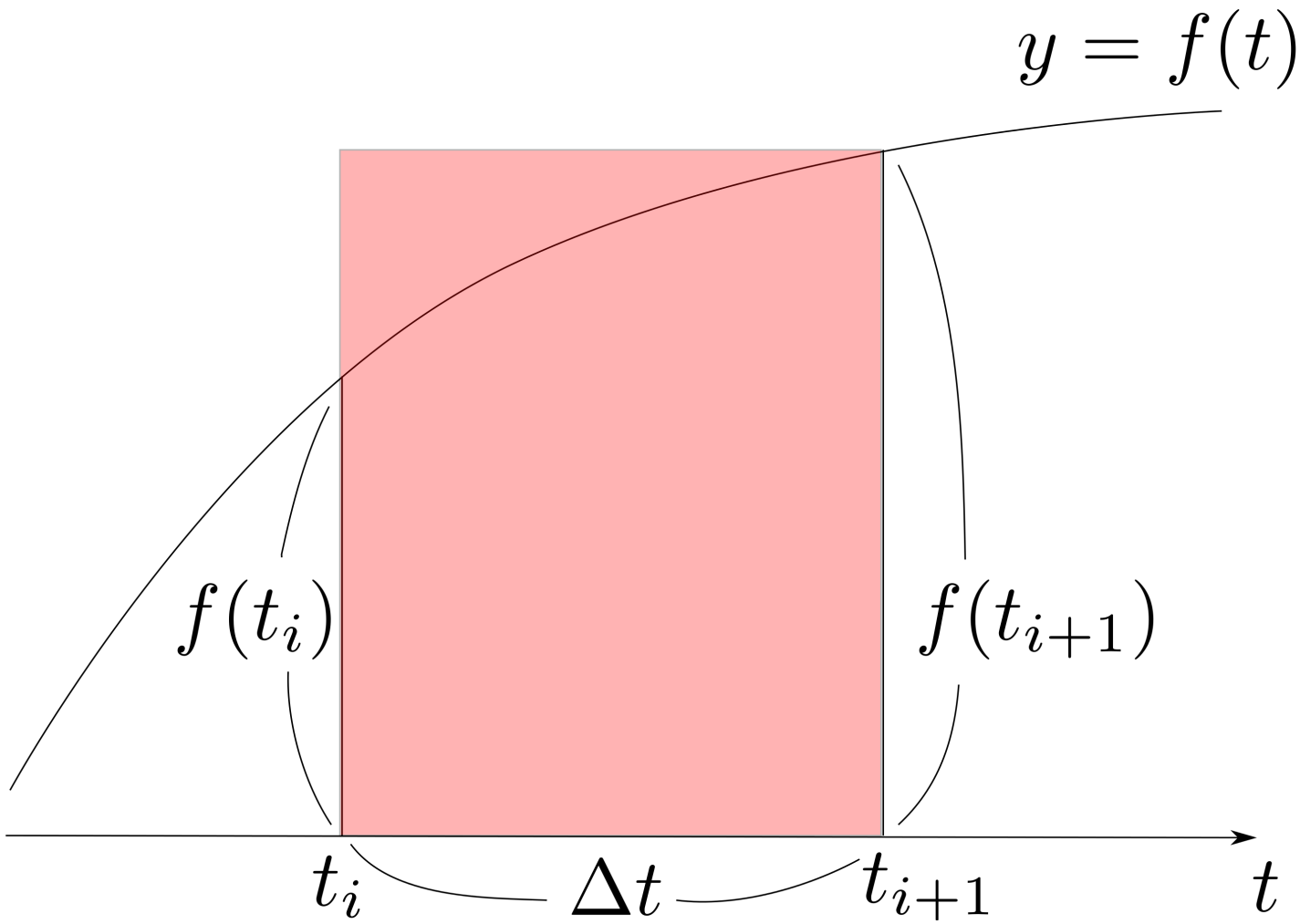
前進オイラー法

$\displaystyle\int_{t_i}^{t_{i+1}} f(t)\mathrm{d}t \approx f(t_i)\Delta t$ と近似。



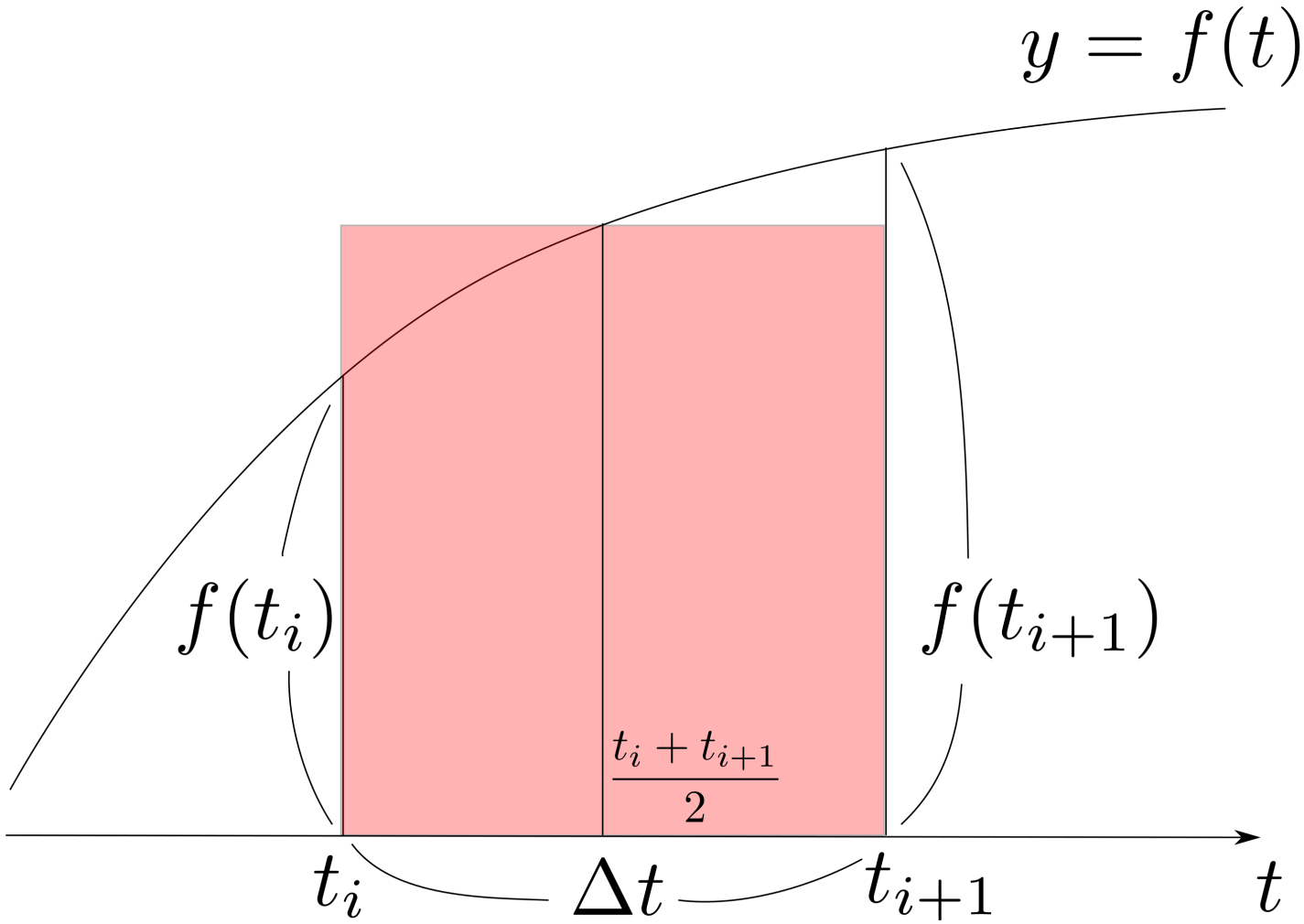
後退オイラー法

$\int_{t_i}^{t_{i+1}} f(t)\mathrm{d}t \approx f(t_{i+1})\Delta t$ と近似。



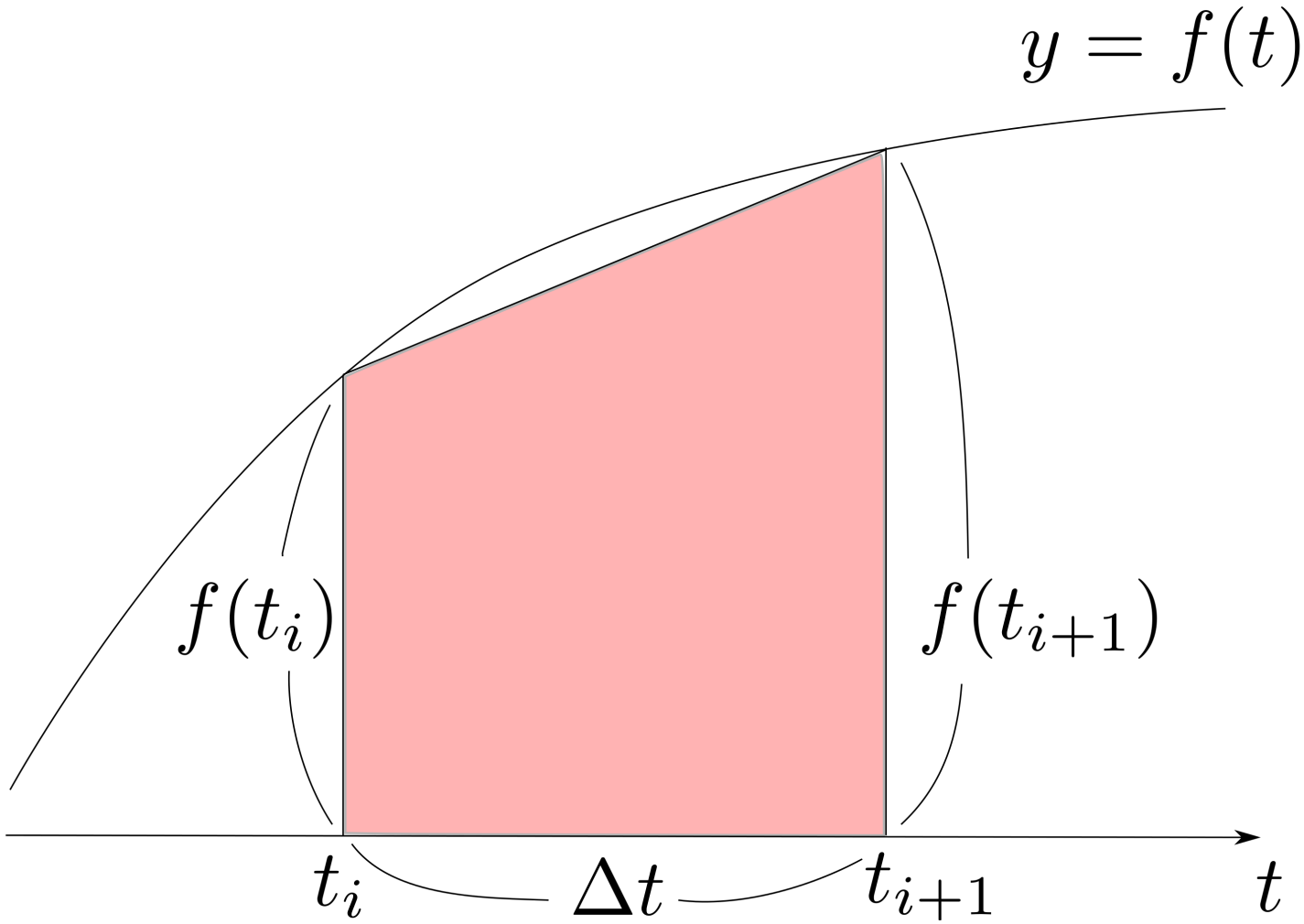
中点法

$$\int_{t_i}^{t_{i+1}} f(t)\mathrm{d}t \approx f\left(\frac{t_i + t_{i+1}}{2}\right)\Delta t = f\left(t_i + \frac{\Delta t}{2}\right)\Delta t$$ と近似。



台形近似

$$\int_{t_i}^{t_{i+1}} f(t)\mathrm{d}t \approx \frac{f(t_i)+f(t_{i+1})}{2}\Delta t$$ と近似。



簡単な例題

微分方程式

$$\frac{\mathrm{d}x}{\mathrm{d}t} = 2t$$

を初期値 $x(0) = 0$ の元で、区間 $[0, 1]$ について解け。
ただし、区間の分割数は 10 とせよ。

前進オイラー法によるプログラム例

```
N:  10          # 分割数
x:  0.0         # 初期値
dt: 1.0/N

printf("%f %f\n", 0.0, x)
for(i in 0..N-1) {
    t1: i*dt              # t_i の値
    t2: (i+1)*dt          # t_{i+1}の値

    x += (2*t1) * dt   # 前進オイラー法による更新

    printf("%f %f\n", t2, x)
}
```

gnuplot での解の表示

計算で得た t と $x(t)$ の値を以下のようにファイルに書き出す (ファイル名は `forward_euler.dat` とした)。

```
0.0 0.0
0.1 0.0
0.2 0.020000000000000004
...
```

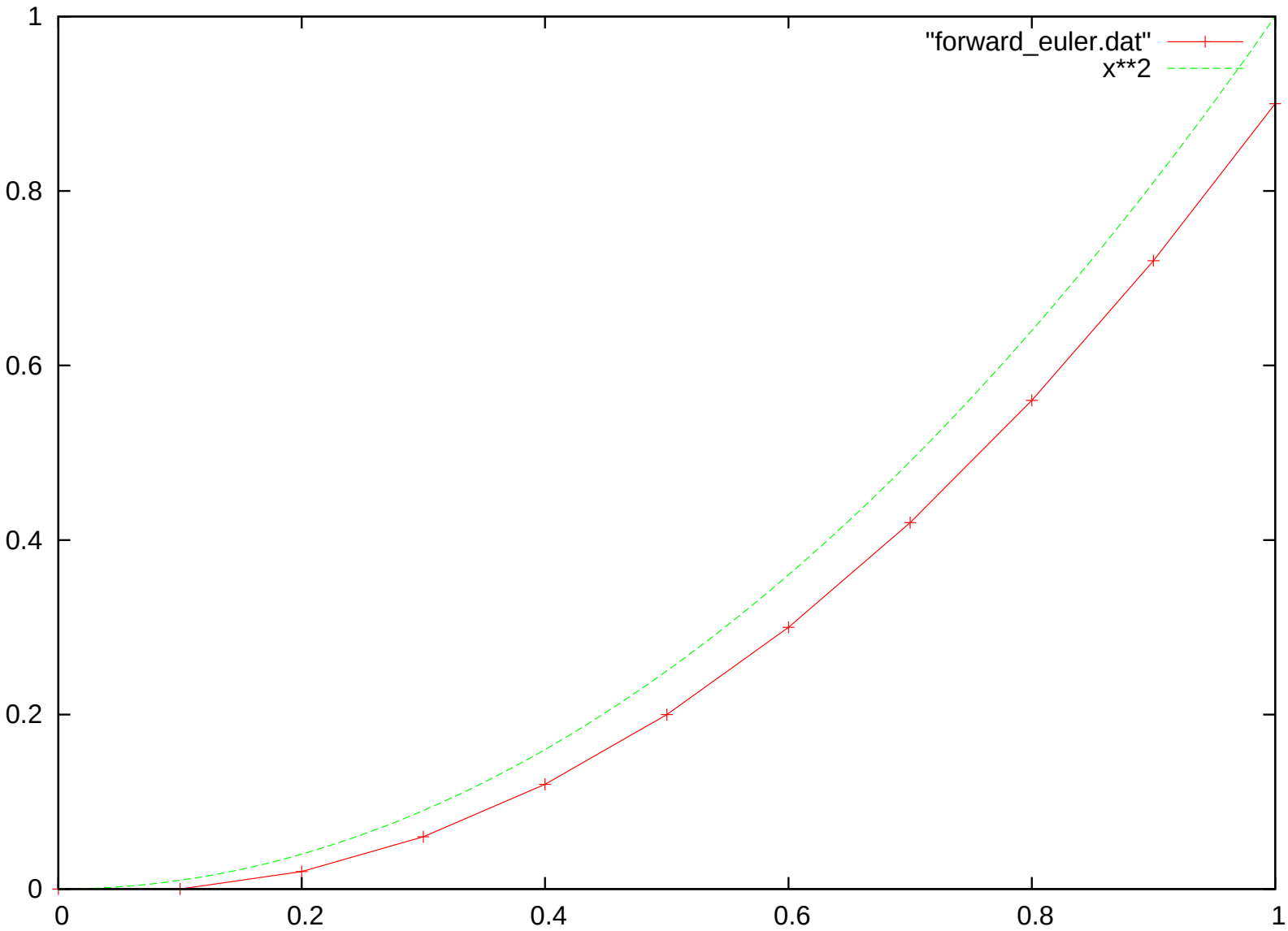
続いて gnuplot のプロンプトで以下のコマンドを入力すれば、これを表示することが出来る。

```
gnuplot> plot "forward_euler.dat" w lp
```

また、例題の微分方程式の厳密解は $x(t) = t^2$ であるので、これも表示してみる。

```
gnuplot> plot "forward_euler.dat" w lp, x**2
```

gnuplot での解の表示



演習 1

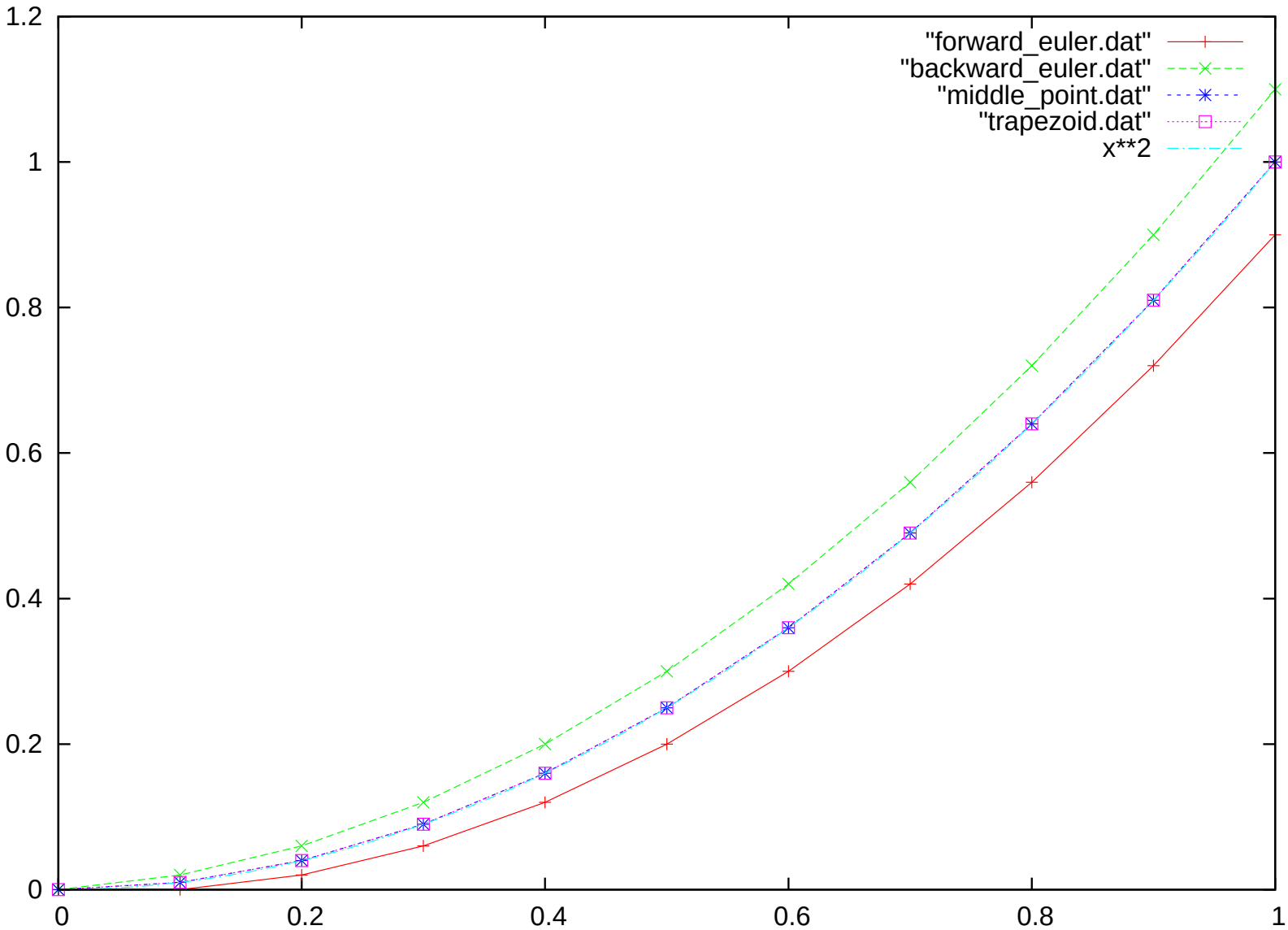
1. 前進オイラー法のプログラムを参考にして、後退オイラー法・中点法・台形公式のプログラムを作成せよ。
2. gnuplot で厳密解と共に表示せよ。
3. 微分方程式

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \cos t$$

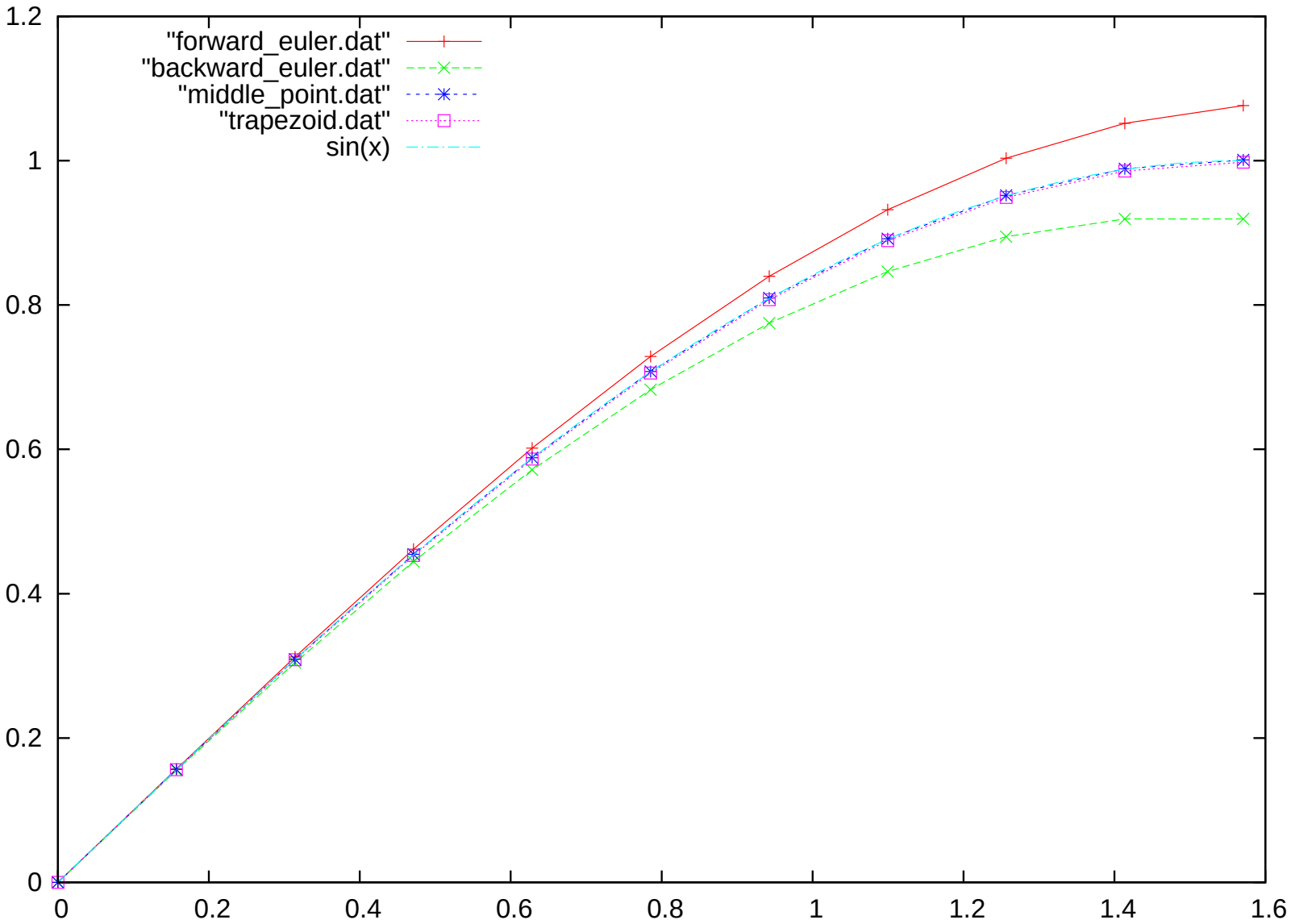
について同様の実験を行え。

- 但し $x(0) = 0$、区間は $[0, \frac{\pi}{2}]$、分割数は 10 とせよ。
- 厳密解は $x(t) = \sin t$ である。

演習 1-(2):結果


"forward_euler.dat"
"backward_euler.dat"
"middle_point.dat"
"trapezoid.dat"
x**2
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1

演習 1-(3):結果


"forward_euler.dat"
"backward_euler.dat"
"middle_point.dat"
"trapezoid.dat"
sin(x)
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6

近似精度の計算

近似の精度はテイラー展開を利用して調べる。(話を簡単にする為、ここでは無限回微分出来る関数を扱う)

テイラー展開

無限回微分出来る関数 $x(t)$ は、定数 a の近傍で以下のように展開出来る。

$$x(t) = x(a) + x'(a)(t-a) + \frac{x''(a)}{2!}(t-a)^2 + \frac{x'''(a)}{3!}(t-a)^3 + \cdots$$

厳密解の級数展開

テイラー展開において、t, a に t_{i+1}, t_i を代入すれば

$$x_{i+1} = x_i + x'(t_i)\Delta t + \frac{x''(t_i)}{2!}\Delta t^2 + \frac{x'''(t_i)}{3!}\Delta t^3 + \cdots$$

となる。よって $\dfrac{\mathrm{d}x}{\mathrm{d}t} = f(t)$(つまり $x'(t) = f(t)$) であるときは

厳密解の級数展開

$$x_{i+1} = x_i + f(t_i)\Delta t + \frac{f'(t_i)}{2!}\Delta t^2 + \frac{f''(t_i)}{3!}\Delta t^3 + \cdots$$

となる。

オイラー法の精度

前進オイラー法では

$$x_{i+1} = x_i + f(t_i)\Delta t$$

となり、厳密解と Δt^1 の項まで一致している。

同様に、後退オイラー法では

$$x_{i+1} = x_i + f(t_{i+1})\Delta t$$

となり、$f(t_{i+1})$ をさらにテイラー展開すると

$$f(t_{i+1}) = f(t_i) + f'(t_i)\Delta t + \cdots$$

となるので

$$x_{i+1} = x_i + f(t_i)\Delta t + f'(t_i)\Delta t^2 + \cdots$$

となる。これも厳密解と Δt^1 の項まで一致している。

以上の事実を**前進・後退オイラー法は 1 次の精度を持つ**と言う。

中点法の精度

中点法では

$$x_{i+1} = x_i + f\left(t_i + \frac{\Delta t}{2}\right)\Delta t$$

となる。$f\left(t_i + \frac{\Delta t}{2}\right)$ を更にテイラー展開すると

$$f\left(t_i + \frac{\Delta t}{2}\right) = f(t_i) + f'(t_i)\frac{\Delta t}{2} + \frac{f''(t_i)}{2!}\left(\frac{\Delta t}{2}\right)^2 + \cdots$$

となるので

$$x_{i+1} = x_i + f(t_i)\Delta t + \frac{f'(t_i)}{2!}\Delta t^2 + \frac{f''(t_i)}{8}\Delta t^3 + \cdots$$

となる。すなわち中点法は 2 次の精度を持つ。

台形公式の精度

台形公式では

$$x_{i+1} = x_i + \frac{f(t_i) + f(t_{i+1})}{2}$$

となり、$f(t_{i+1})$ を更にテイラー展開すると

$$f(t_{i+1}) = f(t_i) + f'(t_i)\Delta t + \frac{f''(t_i)}{2!}\Delta t^2 + \cdots$$

となるので

$$x_{i+1} = x_i + f(t_i)\Delta t + \frac{f'(t_i)}{2!}\Delta t^2 + \frac{f''(t_i)}{4}\Delta t^3 + \cdots$$

となる。すなわち**台形公式は 2 次の精度を持つ。**

近似誤差の累積

n 次精度の公式では 1 ステップあたり、Δt^{n+1} にほぼ比例する誤差が発生する。
積分区間が一定である場合、分割数は $\frac{1}{\Delta t}$ に比例するから、累積誤差はほぼ Δt^n に比例する事になる。

n 次精度の公式で累積する近似誤差はほぼ Δt^n に比例する。

演習 2

- 演習 1-(3) のプログラムを用いて、以下の表を完成させよ。
- 近似精度が n 次ならば、分割数 N を 10 倍する度に誤差がほぼ n 桁減少するはずである。

N	$\sin\frac{\pi}{2}=1$ の近似値 前進オイラー　後退オイラー　中点法　台形公式
10	
10^2	
10^3	
10^4	
10^5	
10^6	
10^7	
10^8	

演習 2:結果

N	$\sin\frac{\pi}{2}=1$ の近似値 前進オイラー	後退オイラー	中点法	台形公式
10	1.07648280269410	0.91940317001461	1.00102882414271	0.99794298635436
10^2	1.00783341987358	0.99212545660563	1.00001028091191	0.99997943823961
10^3	1.00078519254663	0.99921439621984	1.00000010280839	0.99999979438323
10^4	1.00007853776017	0.99992145812749	1.00000000102808	0.99999999794383
10^5	1.00000785396107	0.99999214599780	1.00000000001028	0.99999999997944
10^6	1.00000078539796	0.99999921460163	1.00000000000014	0.99999999999983
10^7	1.00000007853993	0.99999992146030	0.99999999999989	0.99999999999989
10^8	1.00000000785418	0.99999999214622	1.00000000000028	1.00000000000028

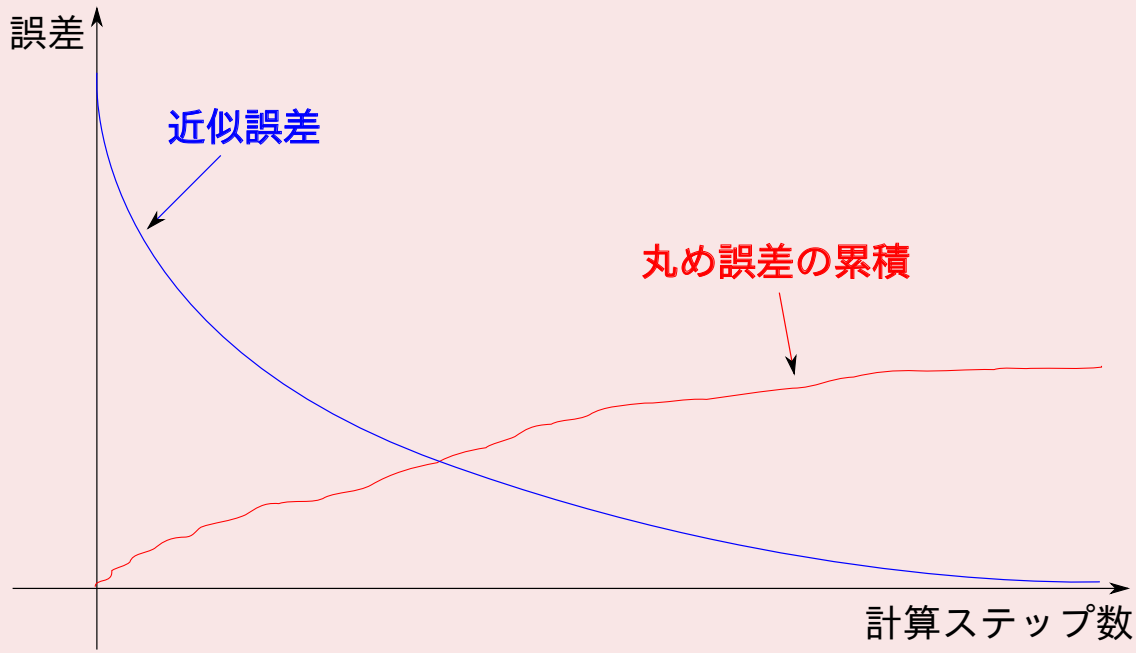
- 確かに n 次の解法では n 桁ずつ誤差が減少している。
- ところが、中点法・台形公式では分割数 10^8 で誤差が増加。

丸め誤差

計算機では実数値を有限 bit で表す為に**丸め**を行う必要がある。
丸めによる誤差を**丸め誤差**という。

近似誤差と丸め誤差

近似誤差を減らす為に分割数を増やせば、累積丸め誤差が増加してしまう。



浮動小数点数

(2 進) 浮動小数点数は

- **符号** s 0 または 1
- **仮数** f 1 以上 2 未満の有限小数
- **指数** e 有限の整数

を用いて

$$(-1)^s \times f \times 2^e$$

と表される。

倍精度浮動小数点数

最もよく用いられる IEEE 方式の倍精度浮動小数点数 (C 言語で言う `double` 型) は以下の様な形式である。

ビット列

s	e (11 ビット)	m (52 ビット)

で

$$(-1)^s \times \left(1 + \frac{m}{2^{52}}\right) \times 2^{e-1023}$$

という数を表す。

この他に単精度や四倍精度浮動小数点数も存在するがこの講義では倍精度に限定して説明をする。

エンコーディングの例

数 3.14 の C 言語の double 型での表現は

```
01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111
```

となる。これをデコードすると

$$
\begin{aligned}
s &= 0 \\
e &= 10000000000_{(2)} = 1024 \\
m &= 1001000111101011100001010001111010111000010100011111_{(2)} \\
&= 2567051787601183
\end{aligned}
$$

となっているから、実際には

$$(-1)^0 \times \left(1 + \frac{2567051787601183}{2^{52}}\right) \times 2^1 = \frac{70706514149716 79}{2251799813685248} = 3.14000000000000124344\cdots$$

という数を表しており、誤差が含まれている。

計算機イプシロン

正の実数 r に対応する浮動小数点数を $f \times 2^e$ とすると、仮数部には誤差 ε_r が含まれる。 つまり

$$r = (f + \varepsilon_r) \times 2^e$$

と表せる。倍精度小数点数の場合に、正しく丸めが行われていれば

$$|\varepsilon_r| < \frac{1}{2^{52}}$$

を満たす。この $\frac{1}{2^{52}}$ を**計算機イプシロン**と呼び ε_m と表す事にすると、

$$\varepsilon_m = 2.220446049250313 \cdots \times 10^{-16}$$

となる。

- 浮動小数点数は仮数部に ε_m で抑えられる程度の誤差を持つ。
- 浮動小数点数の精度は 10 進数で 15〜16 桁程度となる。

情報落ち

仮数が 53bit しかないため、絶対値の差が大きい２数の加減算を行うと**絶対値の小さい方の数が無視される**という現象が起こる。

情報落ちの例

```
> 1.0 + 1.0/2^52
=> 1.0000000000000002
> 1.0 + 1.0/2^53
=> 1.0                          # 情報落ち
> 10.0^15 + 1
=> 1000000000000001.0
> 10.0^16 + 1
=> 1.0e+16                      # 情報落ち
```

情報落ちを防ぐ方法

複数の数の和を計算する場合には、**絶対値の小さい数から順に加える**

演習 3

$$1 + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \cdots$$

の最初の 10^5 項の和を

1. **大きい項から順に足す**
2. **小さい項から順に足す**

の２通りで計算し結果を比較せよ。
この無限級数の厳密な値は

$$1 + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \cdots = \frac{\pi^4}{90}$$

である。

演習 3:結果

大きい順	1.082323233710861
小さい順	1.082323233711138
$\dfrac{\pi^4}{90}$	$1.0823232337113819\cdots$

桁落ち

値が非常に近い、丸め誤差を持つ２数の減算を行うと精度が極端に低下してしまう。

例

```
> 1.0000000000000001 - 1.0
=> 1.1102230246251565e-15     # 1 桁しか精度がない
```

演習 4

$x = a$ における $f(x)$ の導関数は

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

と定義される。つまり、h が十分小さい時

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

と近似される。この近似に基いて、以下の導関数を精度よく計算するにはどうすれば良いか？

1. $f(x) = \sqrt{x}$
2. $f(x) = \sin x$

演習 4-1:結果

$$f'(x) = \frac{\sqrt{x+h} - \sqrt{x}}{h} = \frac{1}{\sqrt{x+h} + \sqrt{x}}$$

とすれば桁落ちを防げる。

例として $h = 10^{-10}$ として $f'(1)$ を計算してみると

工夫なし	0.5000000413701855
工夫あり	0.4999999999875
厳密値	0.5

となる。

演習 4-2:結果

$$f'(x) = \frac{\sin(x+h) - \sin x}{h} = \frac{2\cos(x+\frac{h}{2})\sin\frac{h}{2}}{h}$$

とすれば良い。

例として $h = 10^{-10}$ として $f'(\frac{\pi}{3})$ を計算してみると

工夫なし	0.5000000413701855
工夫あり	0.4999999995669875
厳密値	0.5

となる。

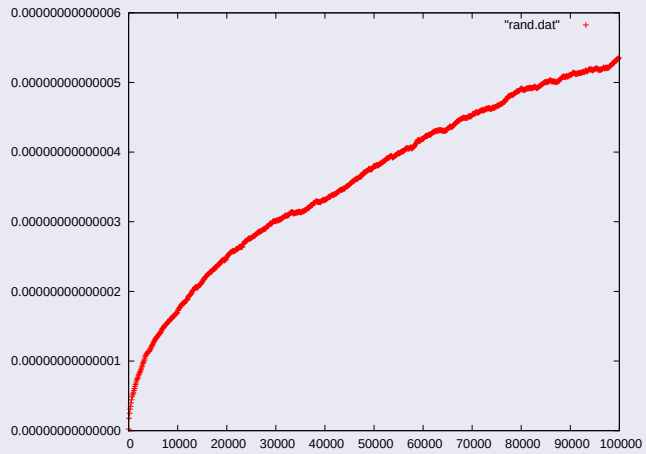
丸め誤差の累積

誤差の累積については以下のような有名な事実がある。

一度の演算で $\pm\varepsilon$ の誤差が発生するとする。ε と $-\varepsilon$ の出方がランダムであるならば n 回の演算によって、平均的に

$$\varepsilon\sqrt{n}$$

程度誤差が累積する。



実際には誤差幅は一定ではないし、正・負の誤差の出方もランダムではないからこんなに単純ではない。

常微分方程式の数値解法 (続き)

次は右辺に $x(t)$ を含む

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(t, x(t))$$

という形の微分方程式を考える。具体例として以下の方程式を取り上げる。

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -x$$

陽的解法と陰的解法

前進オイラー法では

$$x_{i+1} = x_i + f(t_i, x_i)\Delta t$$

となり、i ステップ目の結果から直接 $i+1$ ステップ目の値を求める事が出来る。このような解法を陽的解法という。

後退オイラー法では

$$x_{i+1} = x_i + f(t_{i+1}, x_{i+1})\Delta t$$

となるが、両辺に x_{i+1} を含むのでこのまま利用することは出来ない。このような解法を陰的解法という。

後退オイラー法の例

後退オイラー法では

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -x$$

を

$$x_{i+1} = x_i - x_{i+1}\Delta t$$

と近似する。これを解くには、まず x_{i+1} について解いて

$$x_{i+1} = \frac{x_i}{1 + \Delta t}$$

とすれば良い。

演習 5

常微分方程式

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -x$$

を初期条件 $x(0) = 1$ の元で区間 $[0, 1]$ について解け。
厳密解は $x = e^{-t}$ である。
(この方法では中点法が使えないので、前進・後退オイラー法と台形公式のみでよい。)

演習 5:近似式

前進オイラー

$$x_{i+1} = (1 - \Delta t)x_i$$

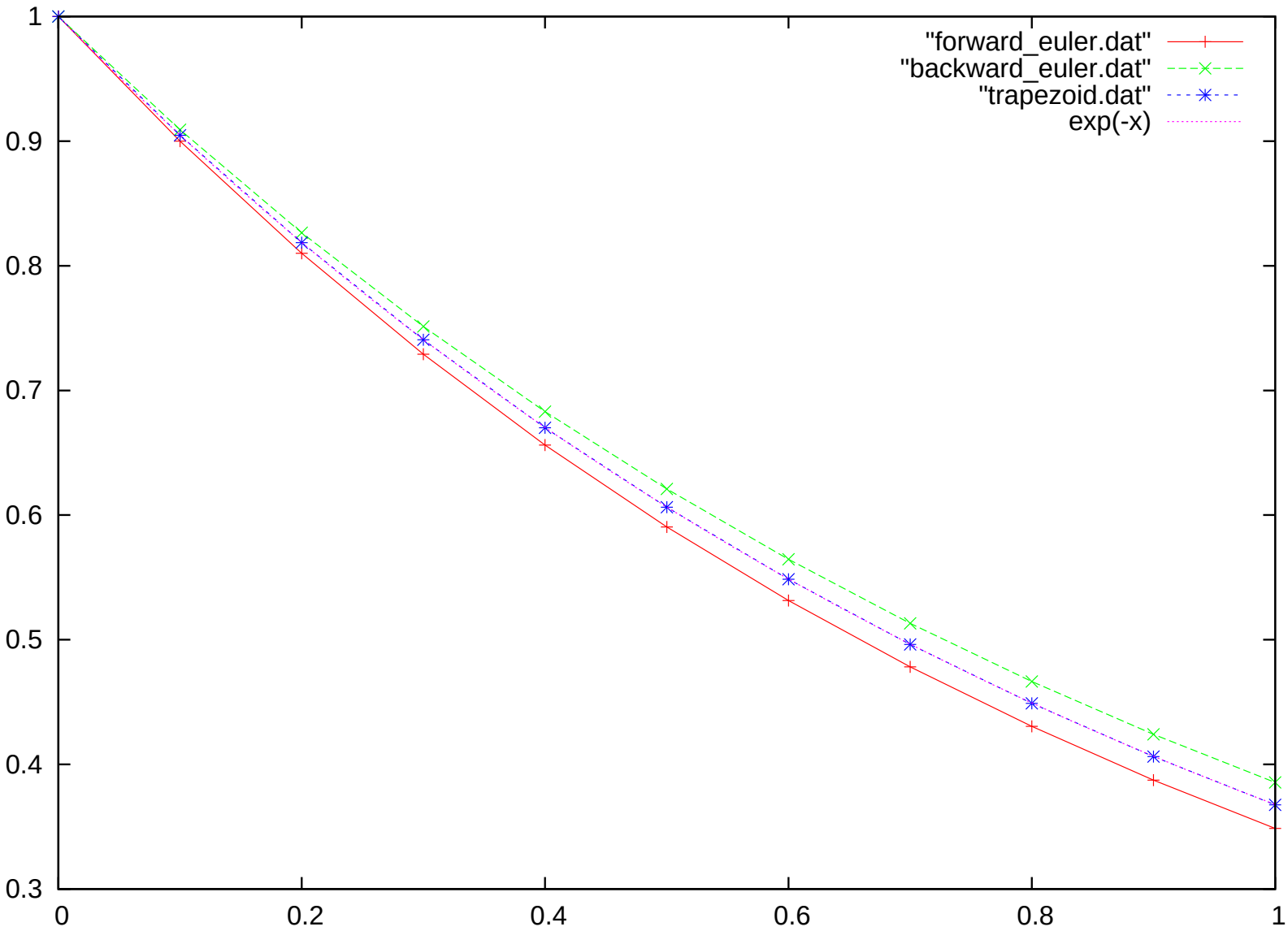
後退オイラー

$$x_{i+1} = \frac{x_i}{1 + \Delta t}$$

台形公式

$$x_{i+1} = \frac{2 - \Delta t}{2 + \Delta t}x_i$$

演習 5:数値解のグラフ


"forward_euler.dat"
"backward_euler.dat"
"trapezoid.dat"
exp(-x)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
0.2
0.4
0.6
0.8
1

代数的に解けない場合

陰的解法を用いる場合、殆どの場合は方程式を代数的に解けない。

例

常微分方程式 $\dfrac{\mathrm{d}x}{\mathrm{d}t} = \sin x$ を後退オイラー法で近似すると

$$x_{i+1} = x_i + \sin x_{i+1} \Delta t$$

となるが、これを x_{i+1} について解くことは出来ない。

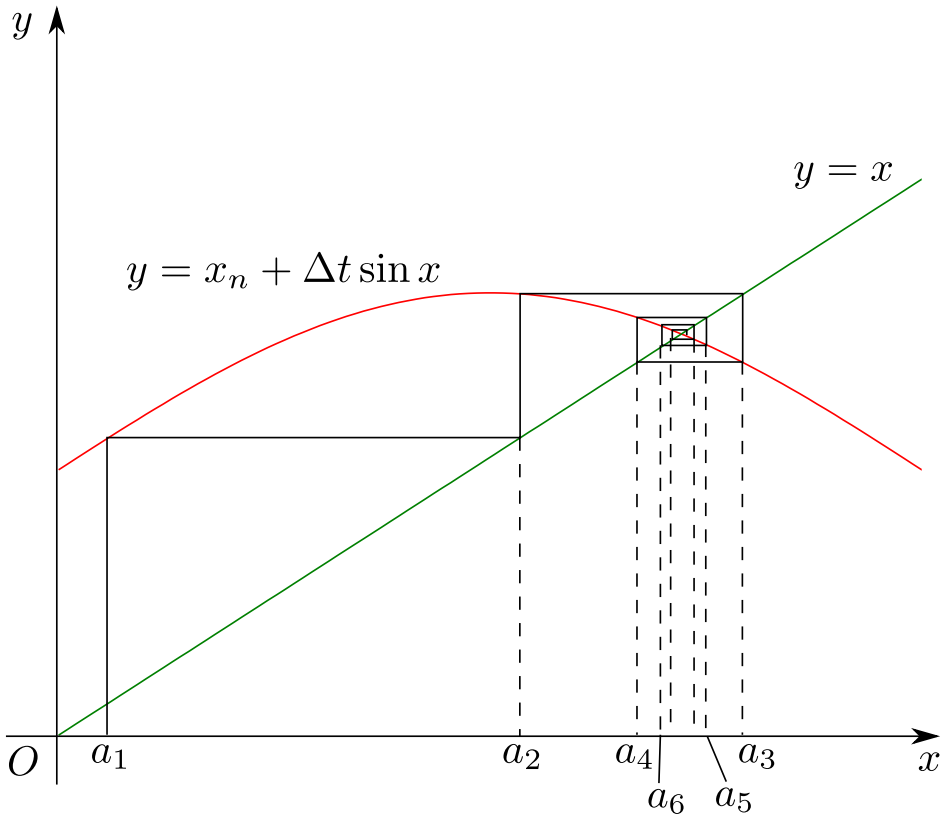
このような場合には**求根アルゴリズム**を用いて方程式を数値的に解くことになる。
求根アルゴリズムにはニュートン法などの様々なアルゴリズムが存在するが、今回は最も簡単な方法を紹介する。

反復法による求根

$x = f(x)$ という方程式の根 $x = \alpha$ を求める場合には、適当な初期値 $x = a_1$ から初めて

$$a_{n+1} = f(a_n)$$

を適当な項数、計算すれば良い。(誤差 $|a_n - \alpha|$ は一回反復する毎におおよそ $|f'(\alpha)|$ 倍になるが、常微分方程式の場合には Δt が掛かっているので十分である事が多い。)



演習 6

1. **反復法により**

$$x = a \sin x + b$$

を解くプログラムを作成せよ。

2. **常微分方程式**

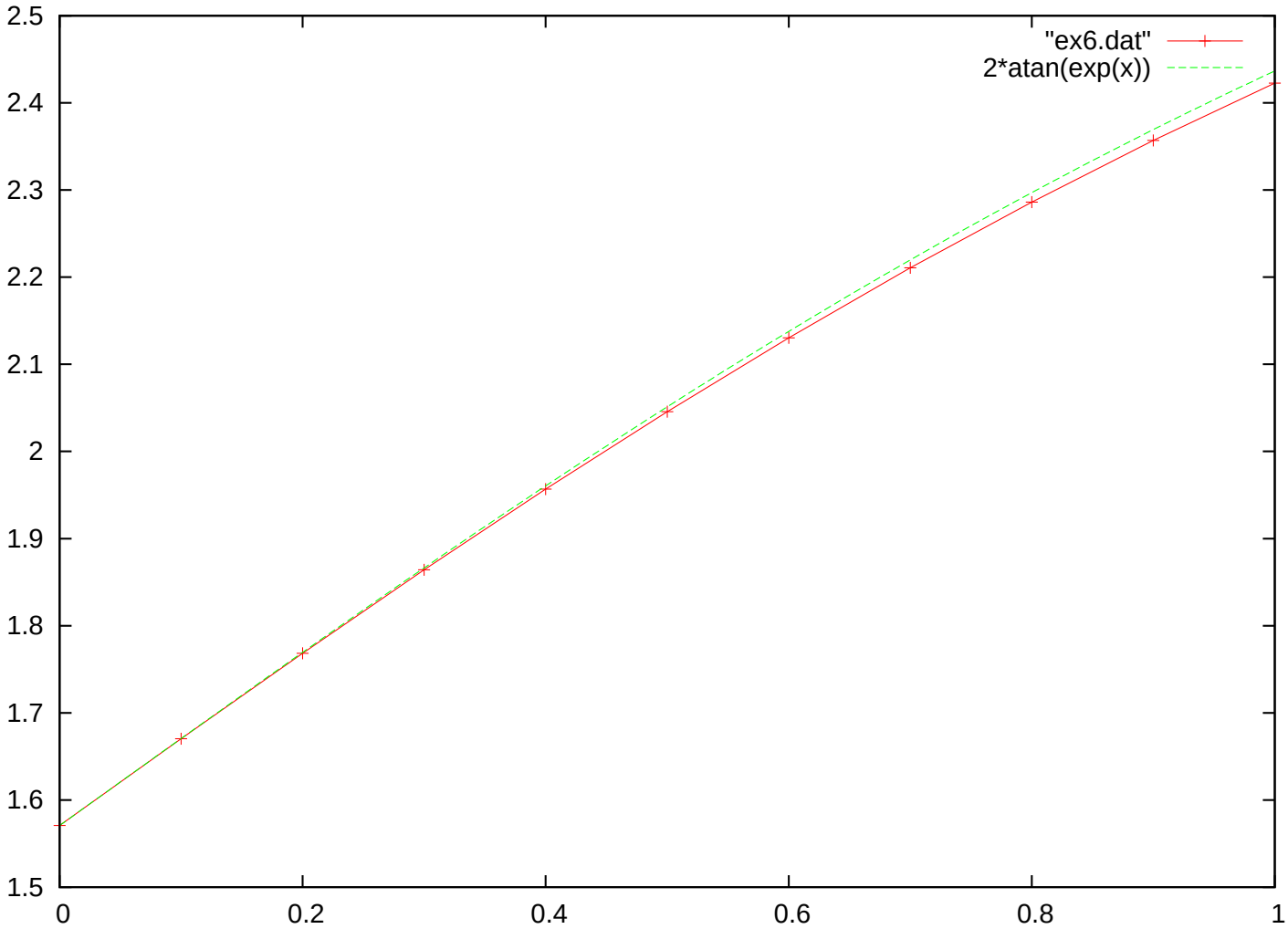
$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sin x$$

を陰的オイラー法により解け。初期値は $x(0) = \frac{\pi}{2}$ とせよ。厳密解は $x(t) = 2 \tan^{-1} e^{x}$ である。

演習 6:プログラム例

```
solve(a, b, x): {
    for (i in 1..6)
        x = a*sin(x) + b
    x
}

N:  10      # 分割数
x:  PI/2    # 初期値
dt: 1.0/N

printf("%f %f\n", 0.0, x)
for(i in 0..N-1) {
    t1: i*dt            # t_i の値
    t2: (i+1)*dt        # t_{i+1}の値

    x = solve(dt, x, x)

    printf("%f %f\n", t2, x)
}
```

演習 6 :結果



高階常微分方程式の数値解法

$\dfrac{\mathrm{d}^2x}{\mathrm{d}t^2}, \dfrac{\mathrm{d}^3x}{\mathrm{d}t^3}, \cdots$ などの高階な導関数を含む場合には新たな未知関数を導入して**連立１階常微分方程式**に直す。

例えば

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = f(t)$$

に対しては $v = \dfrac{\mathrm{d}x}{\mathrm{d}t}$ と置いて

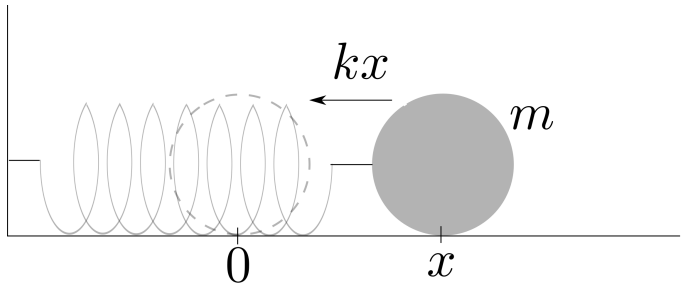
$$\begin{cases} \dfrac{\mathrm{d}v}{\mathrm{d}t} = f(t) \\ \dfrac{\mathrm{d}x}{\mathrm{d}t} = v \end{cases}$$

を解けば良い。

例題:調和振動子のシミュレーション

質量 $m(\mathrm{kg})$ のおもりをバネ定数 $k(\mathrm{N/m})$ のバネにつなぐ。この時自然長からのバネの伸びを $x(\mathrm{m})$ とすると、ニュートンの運動方程式は

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -kx$$



つまり

$$\begin{cases} \dfrac{\mathrm{d}v}{\mathrm{d}t} = -\alpha x \\ \dfrac{\mathrm{d}x}{\mathrm{d}t} = v \end{cases} \qquad \left(\alpha = \frac{k}{m}\right)$$

となる。

調和振動子:前進オイラー法

```
N:    (分割数)
tmin: (計算区間の左端)
tmax: (計算区間の右端)
x:    (初期値)
v:    (初期値)
a:  k/m
dt: (tmax-tmin)/N

printf("%f %f %f\n", tmin, x, v)
for(i in 0..N-1) {
    t2: tmin+(i+1)*dt       # t_{i+1}の値

    # 前進オイラー法による更新
    v1: v
    v += -a*x*dt
    x += v1*dt

    printf("%f %f %f\n", t2, x, v)
}
```

調和振動子:後退オイラー法

$$\begin{cases} v_{i+1} = v_i - \alpha x_{i+1} \Delta t \\ x_{i+1} = x_i + v_{i+1} \Delta t \end{cases}$$

を v_{i+1} について解くと

$$\begin{cases} v_{i+1} = \dfrac{v_i - \alpha x_i \Delta t}{1 + \alpha \Delta t^2} \\ x_{i+1} = x_i + v_{i+1} \Delta t \end{cases}$$

となる。

調和振動子:後退オイラー法

```
N:    (分割数)
tmin: (計算区間の左端)
tmax: (計算区間の右端)
x:    (初期値)
v:    (初期値)
a:    k/m
dt: (tmax-tmin)/N

printf("%f %f %f\n", tmin, x, v)
for(i in 0..N-1) {
    t2: tmin+(i+1)*dt       # t_{i+1}の値

    # 後退オイラー法による更新
    v = (v - a*x*dt)/(1 + a*dt^2)
    x += v*dt

    printf("%f %f %f\n", t2, x, v)
}
```

演習 7

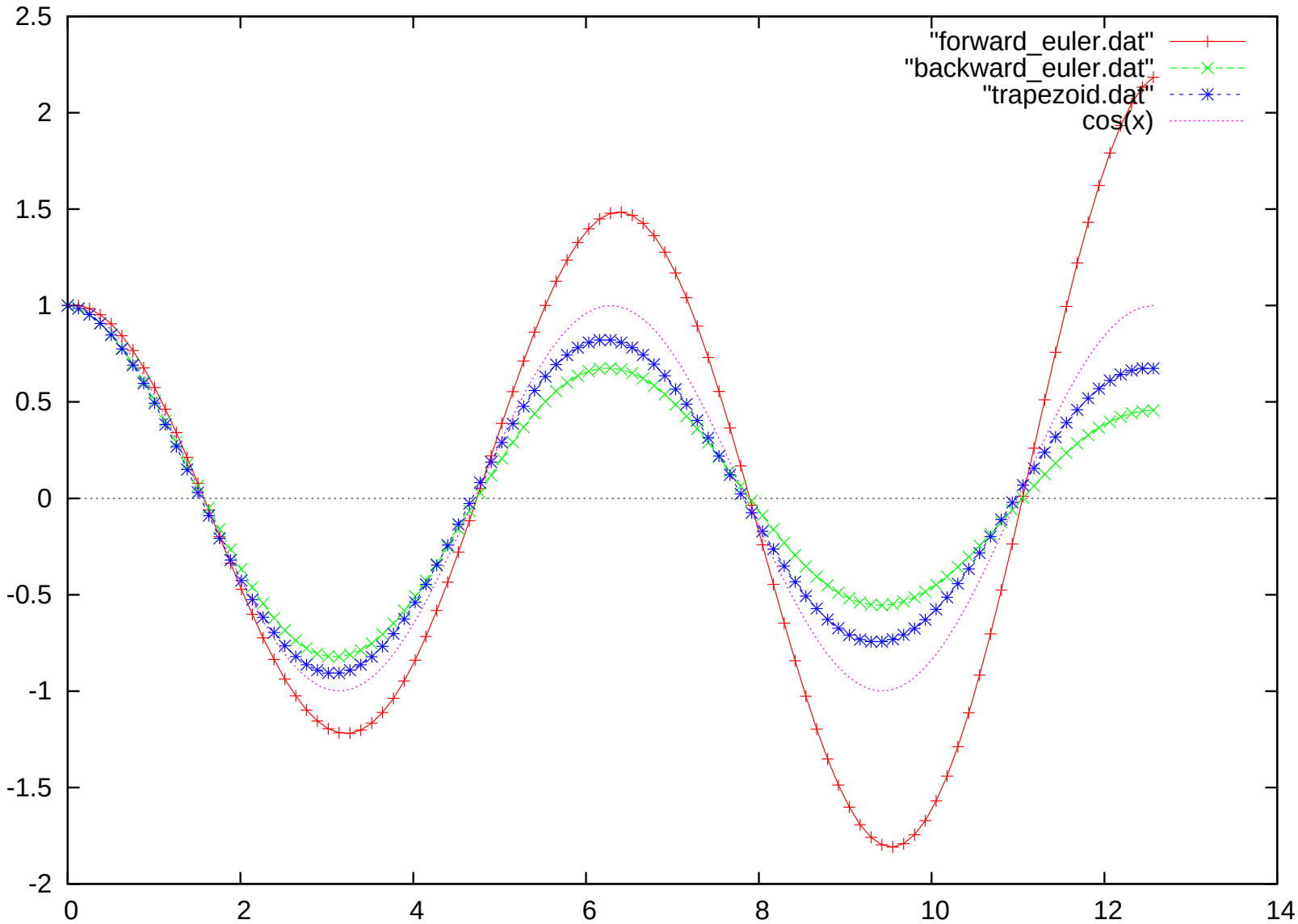
調和振動子のシミュレーションを実装せよ。パラメータは適当なもので良い。例えば $N = 100, m = 1, k = 1$ など。

このシミュレーションの厳密解は

$$x = A \sin\left(\sqrt{\frac{k}{m}} t + \phi\right)$$

(ただし $A = \sqrt{x_0^2 + \frac{m v_0^2}{k}}, \sin\phi = \frac{x_0}{A}, \cos\phi = \frac{v_0 \sqrt{\frac{m}{k}}}{A}$) である。

演習 7:結果


"forward_euler.dat"
"backward_euler.dat"
"trapezoid.dat"
cos(x)
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
2
4
6
8
10
12
14

ルンゲ=クッタ法

まず簡単な例で説明する。
右辺に $x(t)$ を含む場合に中点法を用いると

$$x_{i+1} = x_i + f\left(t_i + \frac{\Delta t}{2}, x\left(t_i + \frac{\Delta t}{2}\right)\right)$$

となるが、ここままだと計算出来ないので $x\left(t_i + \frac{\Delta t}{2}\right)$ の部分を更に前進オイラー法で近似する。すると

$$x\left(t_i + \frac{\Delta t}{2}\right) = x_i + \frac{\Delta t}{2} f(t_i, x_i)$$

となるので、整理すれば

$$k_1 = f(t_i, x_i)$$
$$x_{i+1} = x_i + f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} k_1\right)$$

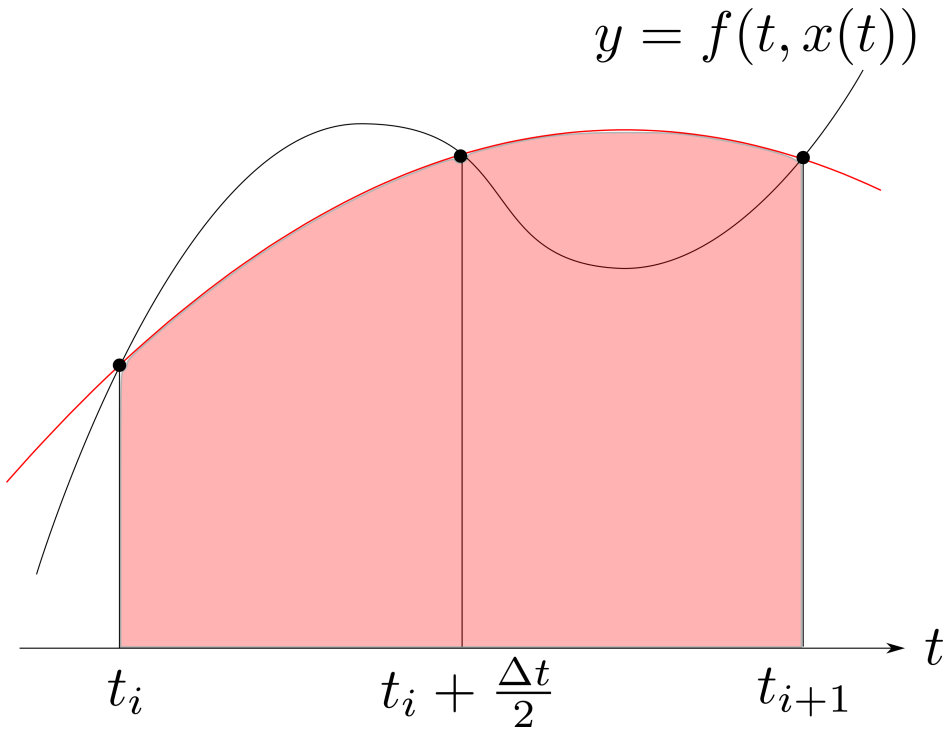
となる。これは 2 次の公式となる。

古典的ルンゲ=クッタ法

図の様に、$y = f(t, x(t))$ を放物線によって近似すると

$$\int_{t_i}^{t_{i+1}} f(t, x(t))\mathrm{d}t \approx \frac{\Delta t}{6}(f(t_i, x_i) + 4f(t_{i+1/2}, x_{i+1/2}) + f(t_{i+1}, x_{i+1}))$$

となる。これは 4 次の精度となる。



古典的ルンゲ=クッタ法

シンプソン法の $f(t_{i+1/2}, x_{i+1/2})$ や $f(t_{i+1}, x_{i+1})$ を上手く近似すれば (ここが難しい所だが)4 次の公式が作れる。
例えば

$$k_1 = f\left(t_i, x_i\right)$$
$$k_2 = f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} k_1\right)$$
$$k_3 = f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} k_2\right)$$
$$k_4 = f\left(t_i + \Delta t, x_i + \Delta t k_3\right)$$

のように各点の値を近似して求め

$$x_{i+1} = x_i + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

とすると 4 次の精度を保った公式となる (中点での値は k_2 と k_3 の平均値にしている)。

古典的ルンゲ=クッタ法

以下の公式を**古典的ルンゲ=クッタ法**と言い、4 次の精度である。

$$
\begin{aligned}
k_1 &= f\left(t_i, x_i\right) \\
k_2 &= f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} k_1\right) \\
k_3 &= f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} k_2\right) \\
k_4 &= f\left(t_i + \Delta t, x_i + \Delta t k_3\right)
\end{aligned}
$$

$$
x_{i+1} = x_i + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)
$$

(以上の説明はあくまで直感的な物であるので注意。厳密な係数の計算は難しいのでここでは省略する。)

ルンゲ=クッタ法

今の例のように

- 区間 $[t_i, t_{i+1}]$ 内にいくつか分点を取り
- 各分点での値を、近似して求め
- それらの重み付き和によって積分を近似する

方法を総称してルンゲ=クッタ法という。一般には

$$
\begin{aligned}
k_1 &= f(t_i + c_1\Delta t, x_i + \Delta t(a_{11}k_1 + a_{12}k_2 + \cdots + a_{1s}k_s)) \\
k_2 &= f(t_i + c_2\Delta t, x_i + \Delta t(a_{21}k_1 + a_{22}k_2 + \cdots + a_{2s}k_s)) \\
&\vdots \\
k_s &= f(t_i + c_s\Delta t, x_i + \Delta t(a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{ss}k_s)) \\
x_{i+1} &= x_i + \Delta t(w_1k_1 + w_2k_2 + \cdots + w_sk_s)
\end{aligned}
$$

という形で表される。

調和振動子:ルンゲ=クッタ法

```
N:    (分割数)
tmin: (計算区間の左端)
tmax: (計算区間の右端)
x:    (初期値)
v:    (初期値)
a:    k/m
dt: (tmax-tmin)/N

printf("%f %f %f\n", tmin, x, v)
for(i in 0..N-1) {
    t2: tmin+(i+1)*dt       # t_{i+1}の値

    vk1 = -a*x
    xk1 = v
    vk2 = -a*(x+xk1*dt/2)
    xk2 = v+vk1*dt/2
    vk3 = -a*(x+xk2*dt/2)
    xk3 = v+vk2*dt/2
    vk4 = -a*(x+xk3*dt)
    xk4 = v+vk3*dt

    v += (vk1+2*vk2+2*vk3+vk4)*dt/6
    x += (xk1+2*xk2+2*xk3+xk4)*dt/6

    printf("%f %f %f\n", t2, x, v)
}
```

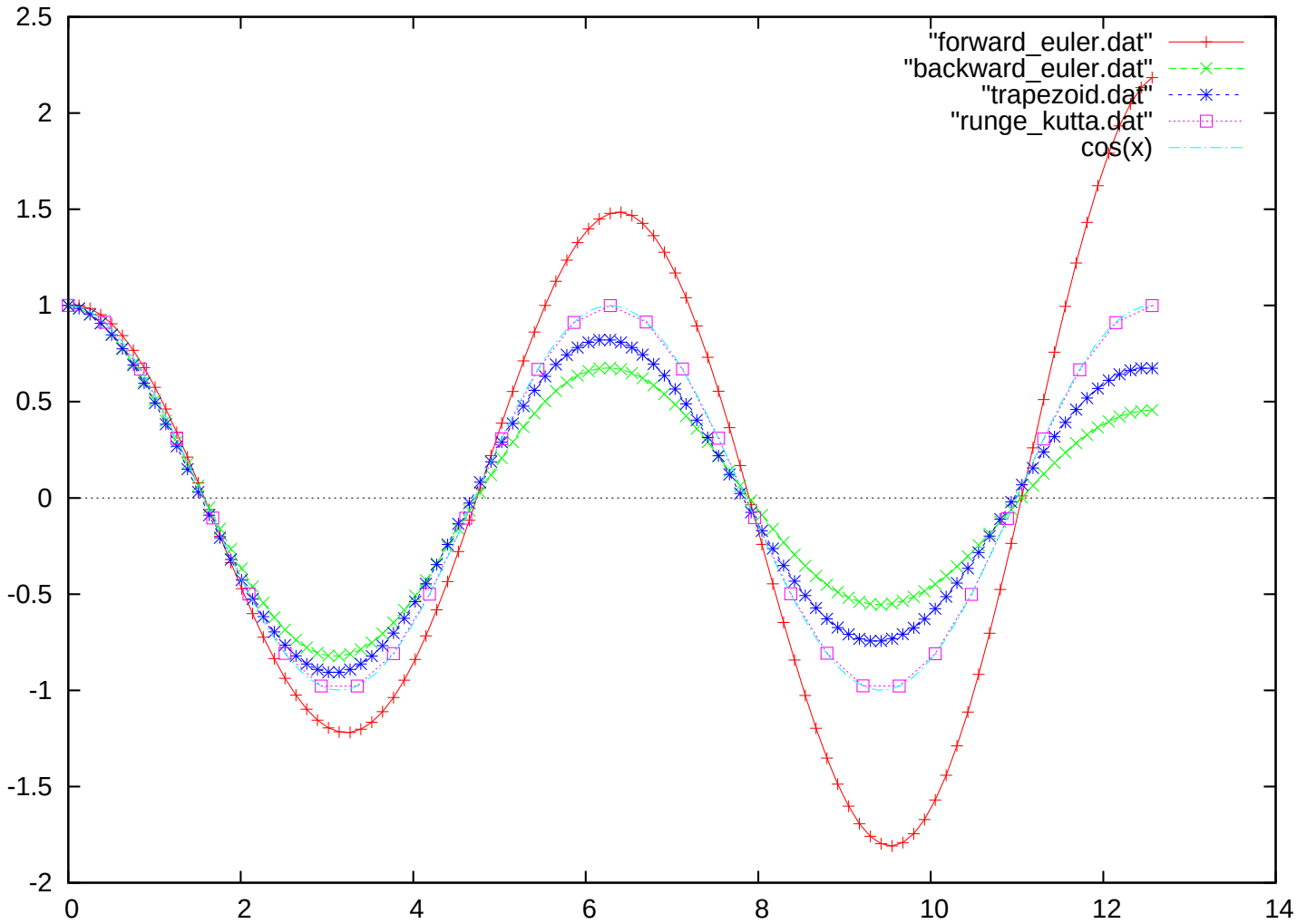
調和振動子:ルンゲ=クッタ法

ベクトルを用いると綺麗に書ける。

```
N:    (分割数)
tmin: (計算区間の左端)
tmax: (計算区間の右端)
x: (x0, v0)
a:    k/m
dt: (tmax-tmin)/N

f((x, v)): (v, -a*x)

printf("%f %f %f\n", tmin, x[0], x[1])
for(i in 0..N-1) {
    t2: tmin+(i+1)*dt       # t_{i+1}の値

    k1: f(x)
    k2: f(x + k1*dt/2)
    k3: f(x + k2*dt/2)
    k4: f(x + k3*dt)

    x += (k1+2*k2+2*k3+k4)*dt/6

    printf("%f %f %f\n", t2, x[0], x[1])
}
```

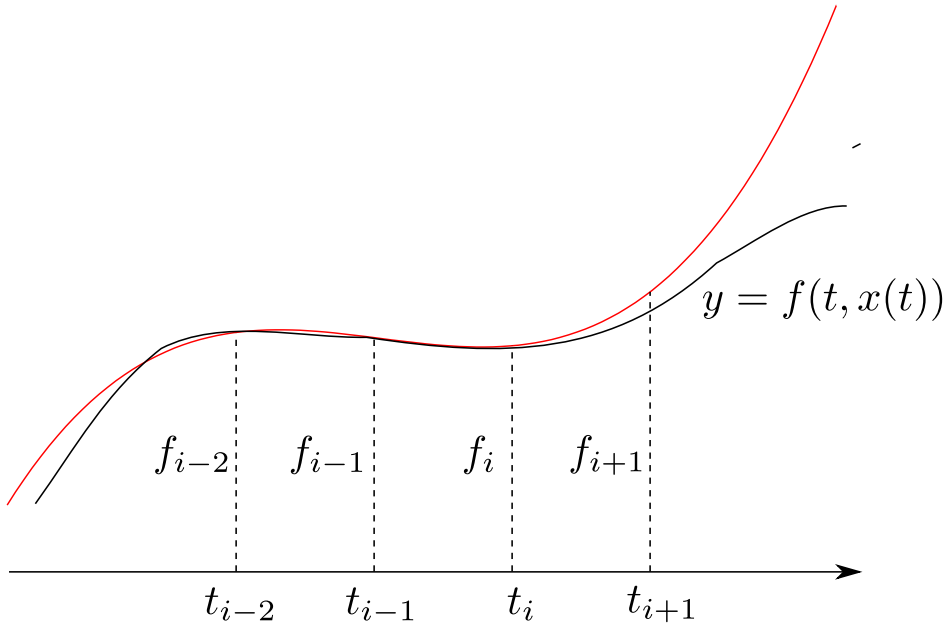
調和振動子のシミュレーション結果

ルンゲクッタ法はステップ数 30、他は 100



線形多段階法

t_i での情報のみから t_{i+1} の情報を求める方法を**一段法**という。
一方、複数の時刻の情報を用いる方法を**多段法**という。



- 多段法では段数を増やせば近似精度をいくらでも向上させる事が出来る。
- 一ステップあたりの計算量も少ない。

アダムス法

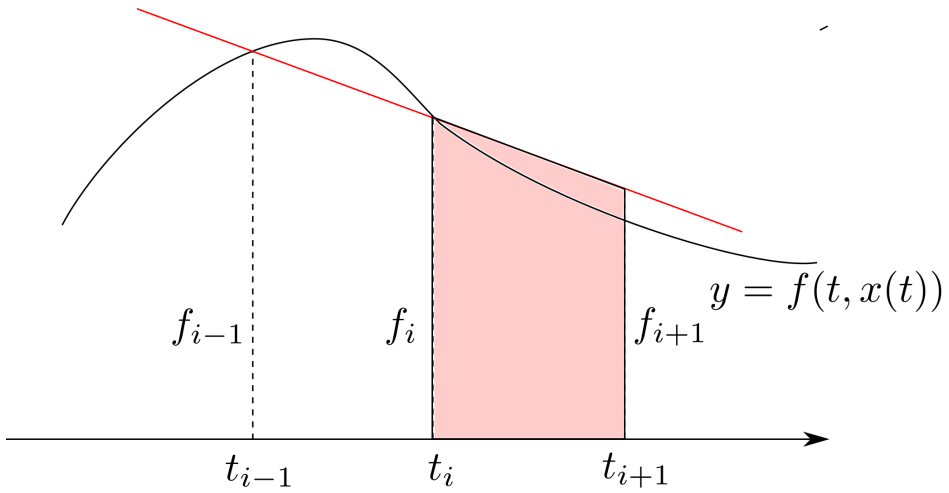
2 点 $(t_{i-1}, f_{i-1}), (t_i, f_i)$ を通る直線の方程式は

$$y = \frac{f_i - f_{i-1}}{\Delta t}(t - t_i) + f_i$$

となる。この直線で $y = f(t, x(t))$ を近似すれば

$$\int_{t_i}^{t_{i+1}} f(t, x(t))\mathrm{d}t \approx \int_{t_i}^{t_{i+1}} \left\{ \frac{f_i - f_{i-1}}{\Delta t}(t - t_i) + f_i \right\} \mathrm{d}t = \frac{3f_i - f_{i-1}}{2}\Delta t$$

となる。



アダムス法

以上の様に、過去の n 点を多項式補間して積分する方法を n 段のアダムス=バッシュフォース公式という。

アダムス=バッシュフォース公式

$$x_{i+1} = x_i + \frac{3f_i - f_{i-1}}{2}\Delta t$$

$$x_{i+1} = x_i + \frac{23f_i - 16f_{i-1} + 5f_{i-2}}{12}\Delta t$$

$$x_{i+1} = x_i + \frac{55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}}{24}\Delta t$$

ただし、x_1, x_2, x_3 の計算には何らかの一段法を用いる。

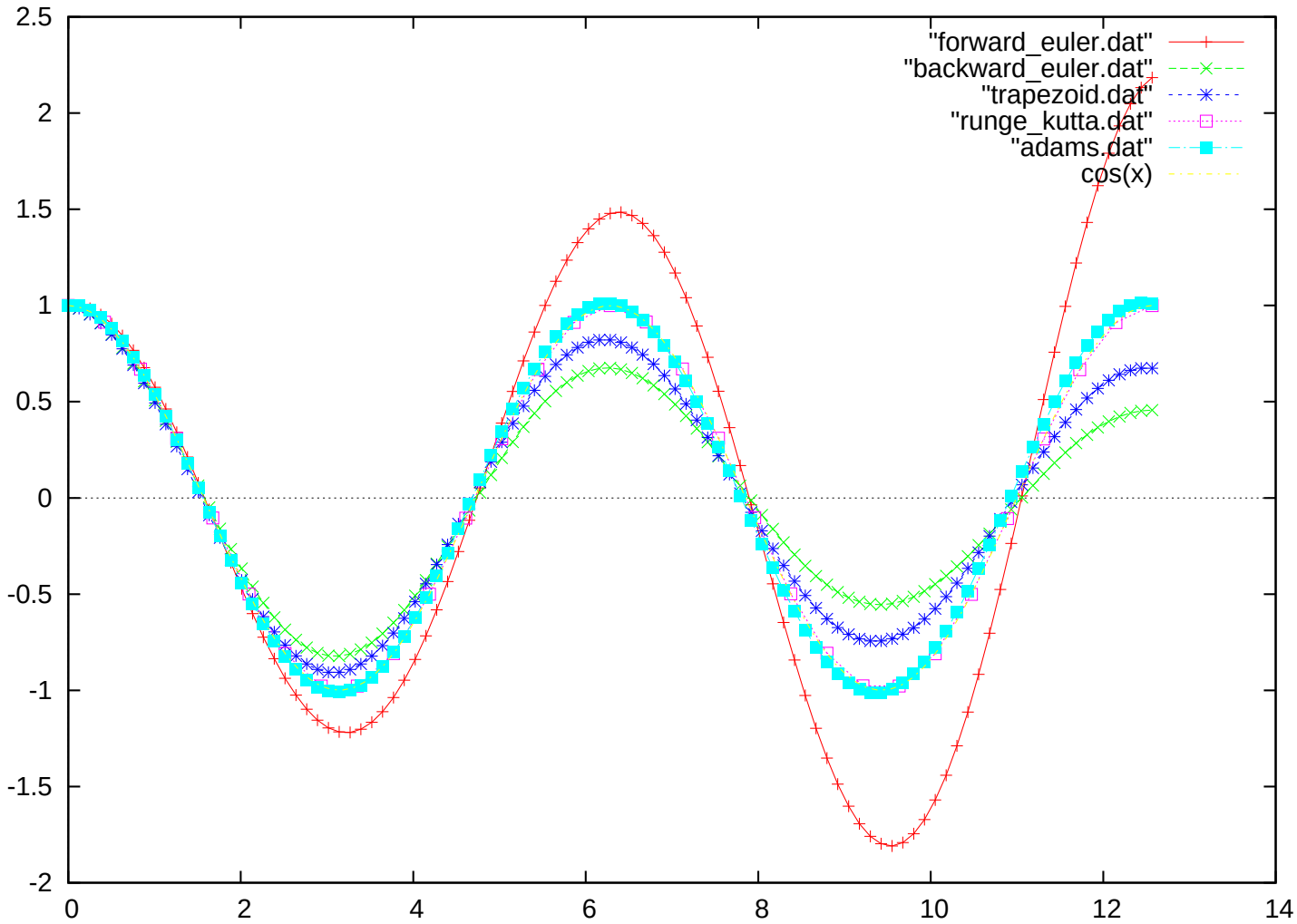
以上は陽的公式になっているが、点 (t_{i+1}, f_{i+1}) も用いて補間すれば陰的公式 (アダムス=モールトン公式) が得られる。

調和振動子:アダムス=バッシュフォース公式

```
N:    (分割数)
tmin: (計算区間の左端)
tmax: (計算区間の右端)
a:    k/m
dt: (tmax-tmin)/N

f((x, v)): (v, -a*x)

x: (x0, v0) # 初期値
printf("%f %f %f\n", tmin, x[0], x[1])

f0: f(x)
x += f0 * dt  # 1 ステップだけ一段法で計算
f1: f(x)
printf("%f %f %f\n", tmin+dt, x[0], x[1])

for(i in 1..N-1) {
    x += (3*f1-f0)/2*dt
    f0, f1 = f1, f(x)
    printf("%f %f %f\n", tmin + (i+1)*dt, x[0], x[1])
}
```

調和振動子:アダムス=バッシュフォース公式

アダムス=バッシュフォース公式のステップ数は 100



アニメーションの作成

調和振動子のシミュレーション結果をアニメーション化してみる。質点の座標は既に得られたので、基本的には好きなソフトウェアで表示すれば良い。ここでは手軽な gnuplot **を利用する。**

まず、数値シミュレーションにより以下の様なデータが得られる。

```
# t の値                x の値                 v の値
0.0                     1.0                    0.0
0.05                    0.9987502604166667     -0.04997916666666668
0.1                     0.995004165581665      -0.09983341144748266
0.15000000000000002     0.9887710787807205     -0.1494381247070911 7
0.2                     0.9800665794836232     -0.1986693205089470 3
...
```

絵を描く

gnuplot で以下のようなコマンドを実行すると、簡単な絵が書ける (質点とバネのつもり)。
X の部分と 0001 の部分を書き換えながら実行するスクリプトを作って、連番画像 (anime-0001.png, anime-0002.png, ...) を生成する。

```
# 出力の設定
set terminal png
set nokey
# 描画範囲の設定
set xrange [-2:2]
set yrange [-1:1]
set size ratio 0.5  # 縦横比
set zeroaxis
# 絵を描く
set parametric
set trange [0:1]
set output "anime-0001.png"
plot X + 0.1*cos(2*pi*t), 0.1*sin(2*pi*t), -2*(1-t) + X*t, 0
```

絵を描く

例えば Ruby なら次のようなスクリプト (plot.rb とする) を作り

```
puts <<-EOS
set terminal png
set nokey
set xrange [-2:2]
set yrange [-1:1]
set size ratio 0.5
set zeroaxis
set parametric
set trange [0:1]
EOS

step = 0
open("oscillator.dat").each do |line|
    t,x,v = line.split.map(&:to_f)

    puts "set output \"anime-#{sprintf("%04d", step)}.png\""
    puts "plot #{x} + 0.1*cos(2*pi*t), 0.1*sin(2*pi*t), -2*(1-t) + #{x}*t, 0"

    step += 1
end
```

次のように実行する

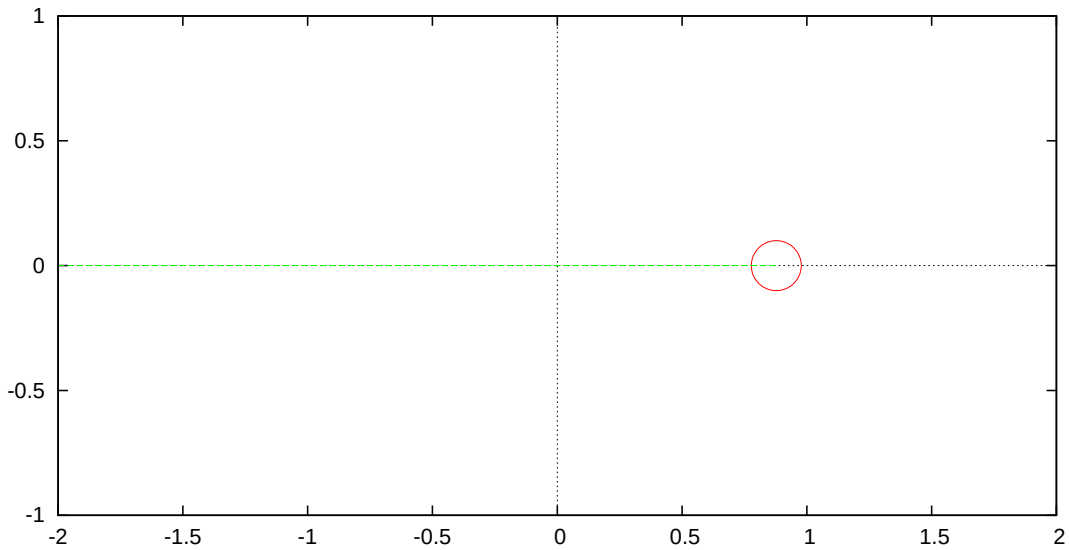
```
% ruby plot.rb | gnuplot
```

動画にする

ここでは ffmpeg を用いる。フレームレートが 10 ならば以下の様にする。

```
ffmpeg -r 10 -i anime-%04d.png anime.mp4
```

以上で動画が作成出来る。



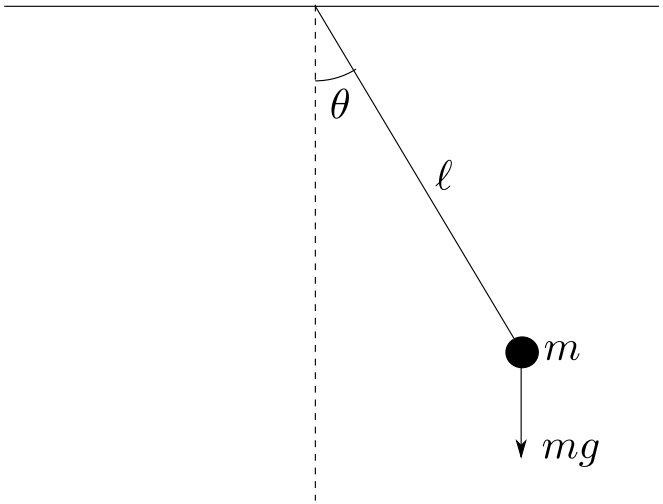
演習 8

単振子のシミュレーションを作成せよ。

右図の単振子の運動方程式は

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = \frac{g}{\ell}\sin\theta$$

となる。



1 次元波動方程式のシミュレーション

図のような弦の振動を表す方程式は

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

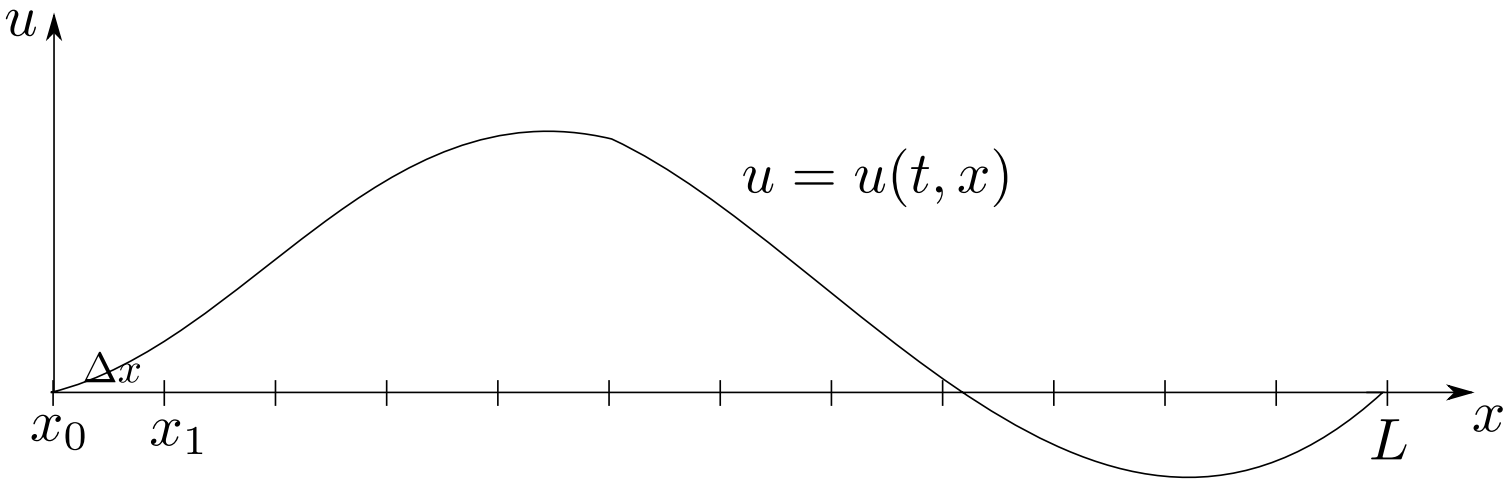
という偏微分方程式となる。これを初期条件

$$u(0,x) = \phi(x), \quad \frac{\partial u}{\partial t}(0,x) = \varphi(x)$$

をディリクレ境界条件

$$u(t,0) = u(t,L) = 0$$

の元で解くことを考える。



補助変数の導入

(あまり一般的なやり方では無いが)

$$v = \frac{\partial u}{\partial t}$$

という補助変数を導入すると、解くべき方程式は

$$\frac{\partial v}{\partial t} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad \frac{\partial u}{\partial t} = v$$
$$u(0,x) = \phi(x), \quad v(0,x) = \varphi(x)$$
$$u(t,0) = u(t,L) = 0$$

となる。この時間方向の離散化は

$$v_{i+1} = v_i + c^2 \frac{\partial^2 u}{\partial x^2} \Delta t$$
$$u_{i+1} = u_i + v_{i+1} \Delta t$$

という形にする。

中心差分

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(t, x+\Delta x) + u(t, x-\Delta x) - 2u(t,x)}{\Delta x^2}$$

とする (中心差分法)。これは

$$u(t, x+\Delta x) = u(t,x) + \frac{\partial u}{\partial x}\Delta x + \frac{1}{2}\frac{\partial^2 u}{\partial x^2}\Delta x^2 + \cdots$$
$$u(t, x-\Delta x) = u(t,x) - \frac{\partial u}{\partial x}\Delta x + \frac{1}{2}\frac{\partial^2 u}{\partial x^2}\Delta x^2 - \cdots$$

というテイラー展開より

$$u(t, x+\Delta x) + u(t, x-\Delta x) - u(t,x) = \frac{\partial^2 u}{\partial x^2}\Delta x^2 + \cdots$$

となる事を利用している。

1 次元波動方程式の差分方程式

以上をまとめると、解くべき差分方程式は

$$
\begin{aligned}
v_{i+1}^{j} &= v_i^j + \frac{c^2\Delta t}{\Delta x^2}(u_i^{j+1} + u_i^{j-1} - 2u_i^j) \\
u_{i+1}^{j} &= u_i^j + v_{i+1}^j \Delta t \\
u_0^j &= \phi(x^j), \quad v_0^j = \varphi(x^j) \\
u_i^0 &= 0, \quad u_i^M = 0
\end{aligned}
$$

となる。ただし i は時刻 j は空間に対応しており、空間方向の分割数を M としている。詳細は省略するが、この解が安定する為には

$$
\frac{c^2\Delta t^2}{\Delta x^2} \leqq 1
$$

である事が必要となる。