sp-13. 数値微分と数値積分

(Scheme プログラミング)

https://www.kkaneko.jp/cc/scheme/index.html

金子邦彦

アウトライン

13-1 数値微分と数値積分

13-2 パソコン演習

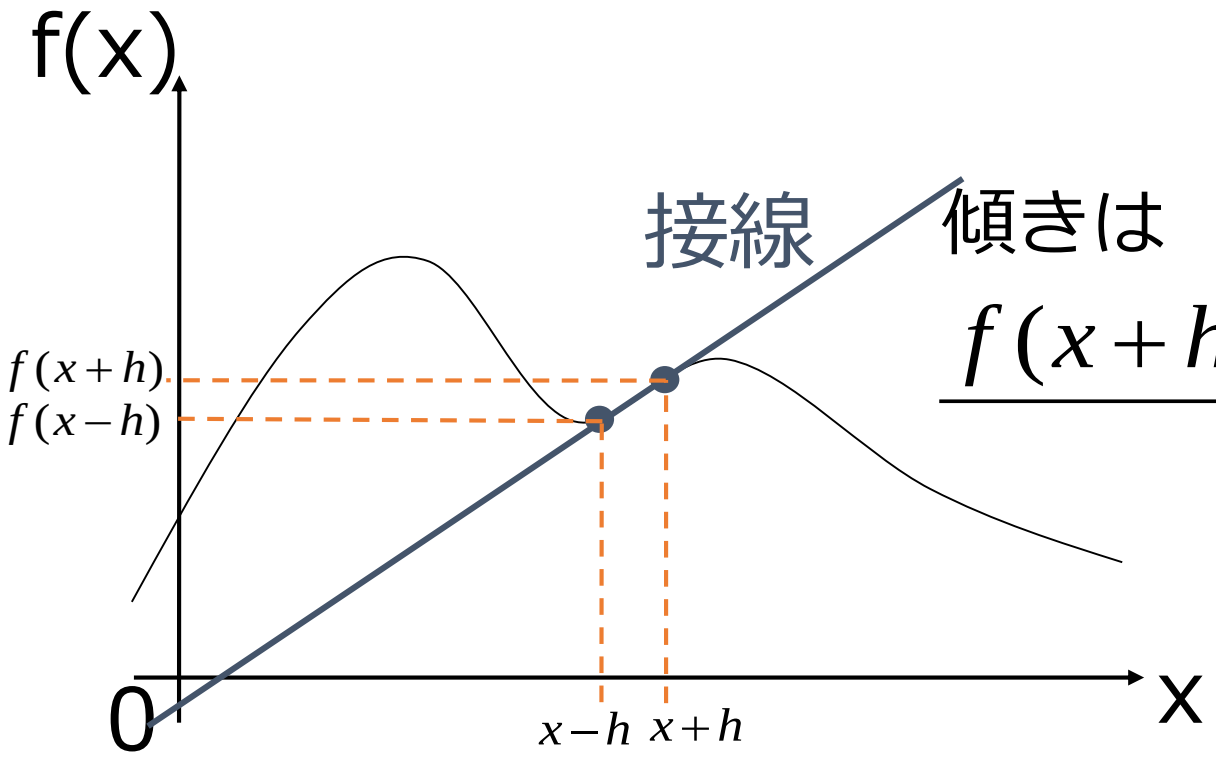
13-3 課題

13-1 数値微分と数値積分

内容

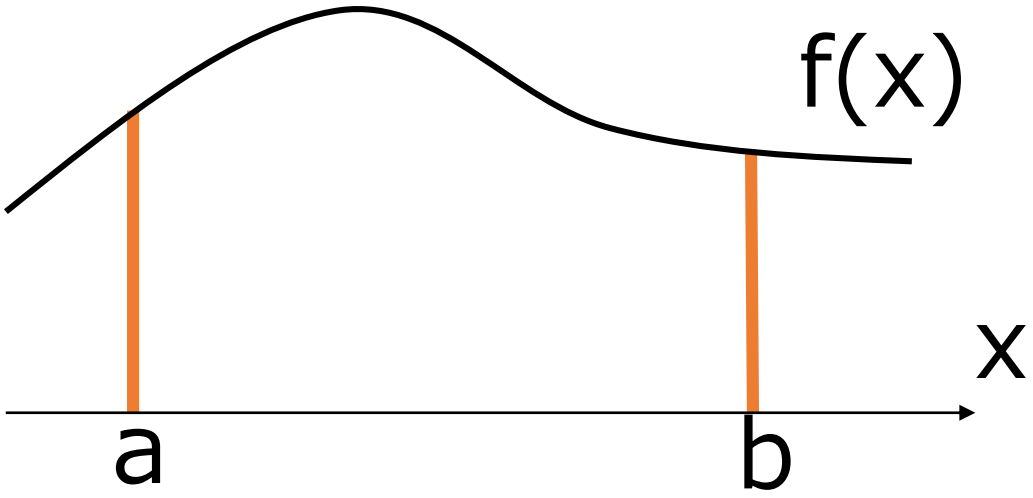
1. 接線の傾き d/dx
2. 台形則による数値積分 trapezoid

接線の傾き



h を 0 に近づけることで、接線の傾き（＝f '(x) ）の「近似値」が求まる

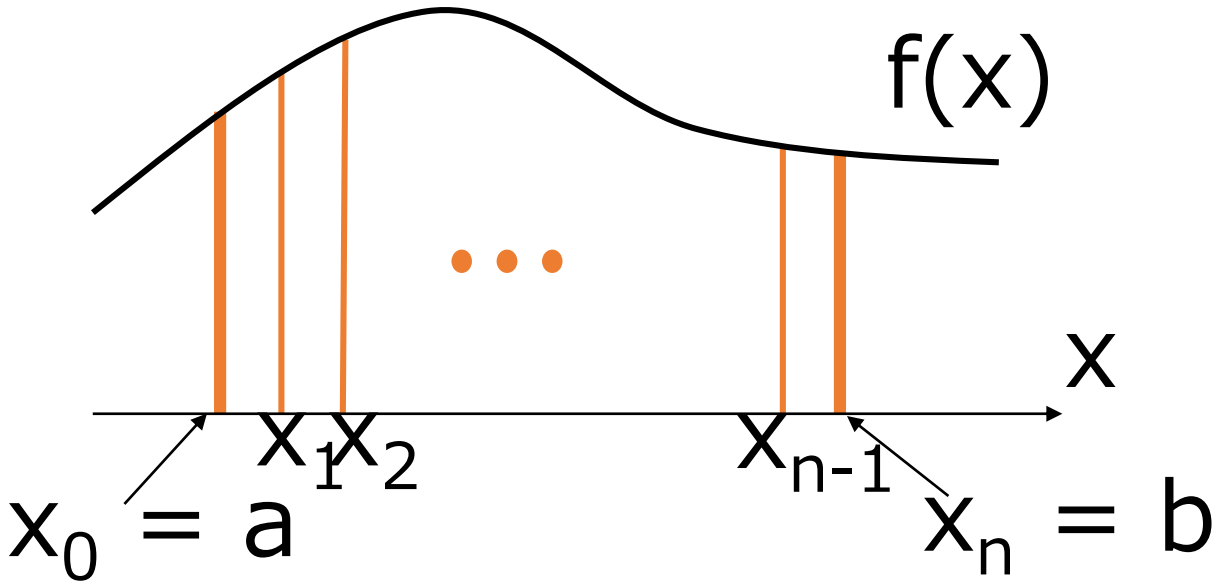
定積分



$$\text{定積分：} I = \int_a^b f(x)dx$$

区間 [a, b] で，連続関数 f, x 軸, x=a, x=b で囲まれた面積

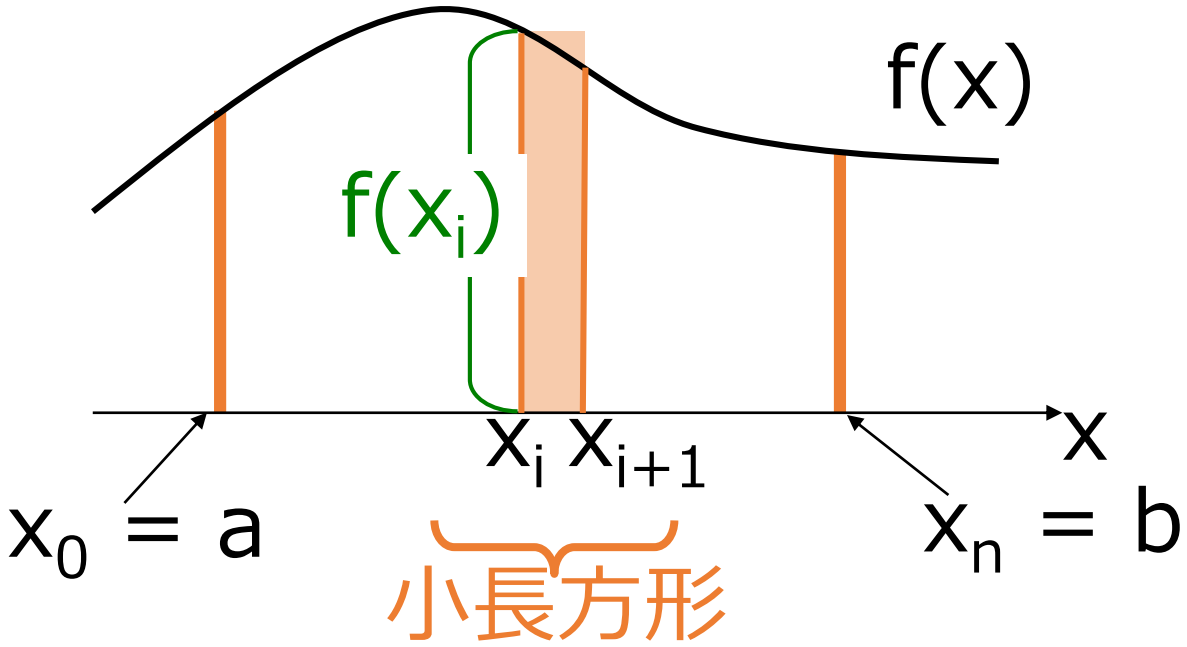
区間 [a, b] の小区間への分割



- n 個の等間隔な小区間に分割
  - 幅： $h = (b-a) / n$
  - 小区間： $[x_0, x_1], [x_1, x_2], \ldots, [x_{n-1}, x_n]$

    但し， $x_0 = a$, $x_i = x_0 + i \times h$

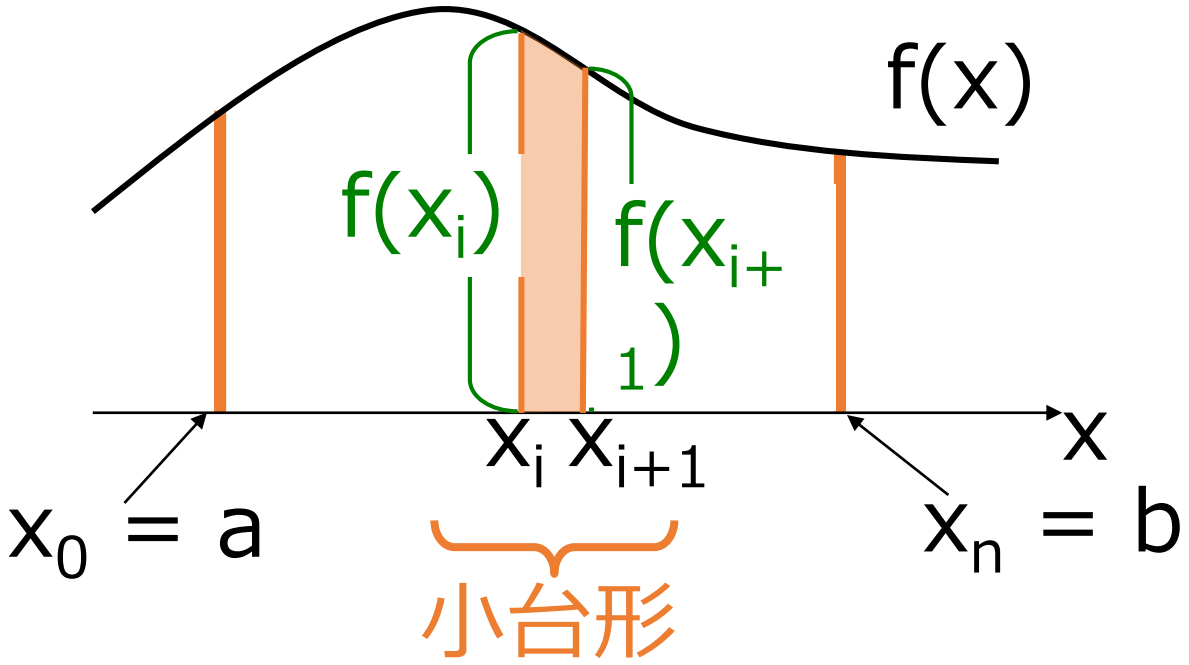
小長方形



- 小長方形の面積は

$$f(x_i)h$$

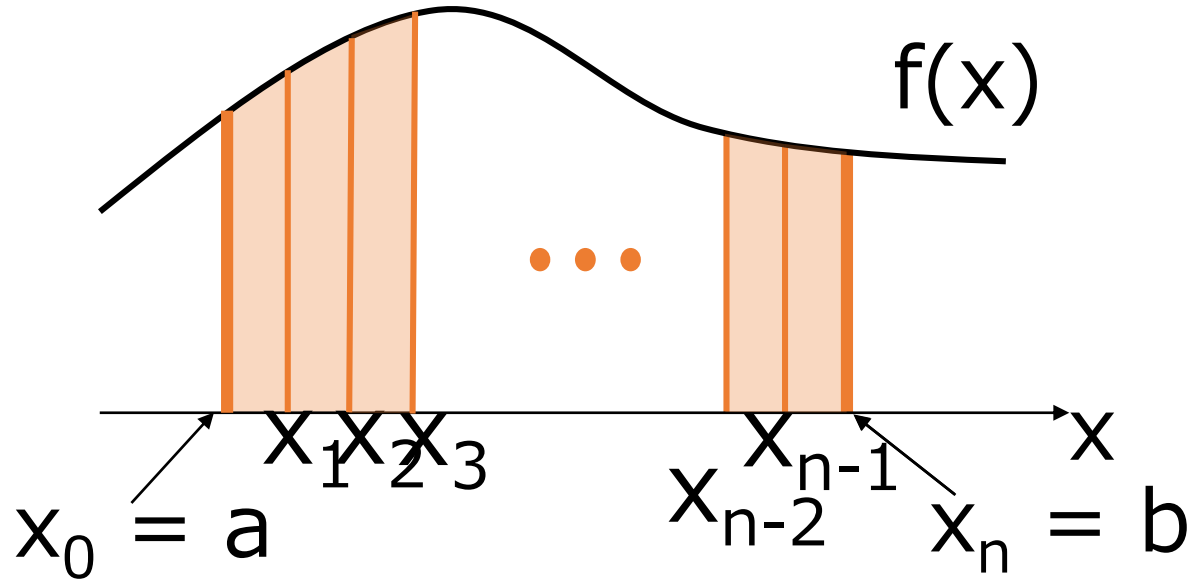
小台形



- 小台形の面積は

$$\frac{h}{2}(f(x_i) + f(x_{i+1}))$$

台形則 trapezoidal rule



- 小台形の面積の和は

$$S_n = \sum_{i=0}^{n-1} \frac{h}{2}(f(x_i) + f(x_{i+1}))$$

- 定積分 I を，この和 S_n で近似 ⇒ 台形則という

台形則による数値積分

- 区間［a，b］を n 等分 (1区間の幅h=(b-a)/n)
- n 個の台形を考え，その面積の和 S_n で，定積分 I を近似
  - f(x) が連続関数のときは，n を無限大に近づければ，S_n は I に近づく

$$I = \lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{h}{2}(f(x_i) + f(x_{i+1})) = \lim_{n\to\infty} S_n$$

- 但し，単純に「n を大きくすればよい」とは言えない
  - n を大きくすると　⇒　計算時間の問題，丸め誤差の問題が発生

数値積分の意味

- 式で書いてある関数の積分
  ⇒　ごく限られた関数しか定積分できない

- 「数値積分」が重要になる

台形則

- 両端 $x_0 = a$ と $x_n = b$ を除いて，$f(x_i)$ は2度現れる

$$S_n = \sum_{i=0}^{n-1} \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right)$$

$$= \frac{h}{2}\left(f(x_0) + 2\underbrace{\left(f(x_1) + \cdots + f(x_{n-1})\right)}_{\text{2回現れる部分}} + f(x_n)\right)$$

$$= h \cdot \left(\frac{1}{2} f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2} f(x_n)\right)$$

$$= h \cdot \left(\frac{1}{2} f(a) + \sum_{i=1}^{n-1} f(a+ih) + \frac{1}{2} f(b)\right)$$

但し　h = (b-a) / n

13-2 パソコン演習

パソコン演習の進め方

- 資料を見ながら，「例題」を行ってみる
- 各自，「課題」に挑戦する
- 自分のペースで先に進んで構いません

DrScheme の使用

- DrScheme の起動
  プログラム　→ PLT Scheme → DrScheme
- 今日の演習では「Advanced Student」に設定
  Language
  → Choose Language
  → Advanced Student
  → OK
  →「Execute ボタン」

例題１．接線の傾き

- 接線の傾きを求める関数 d/dx を作り，実行する
  - 数値 x, h と関数 f から，x における f の傾き（= f' (x)）の近似値を求める
  - d/dx は高階関数

「例題１．接線の傾き」の手順

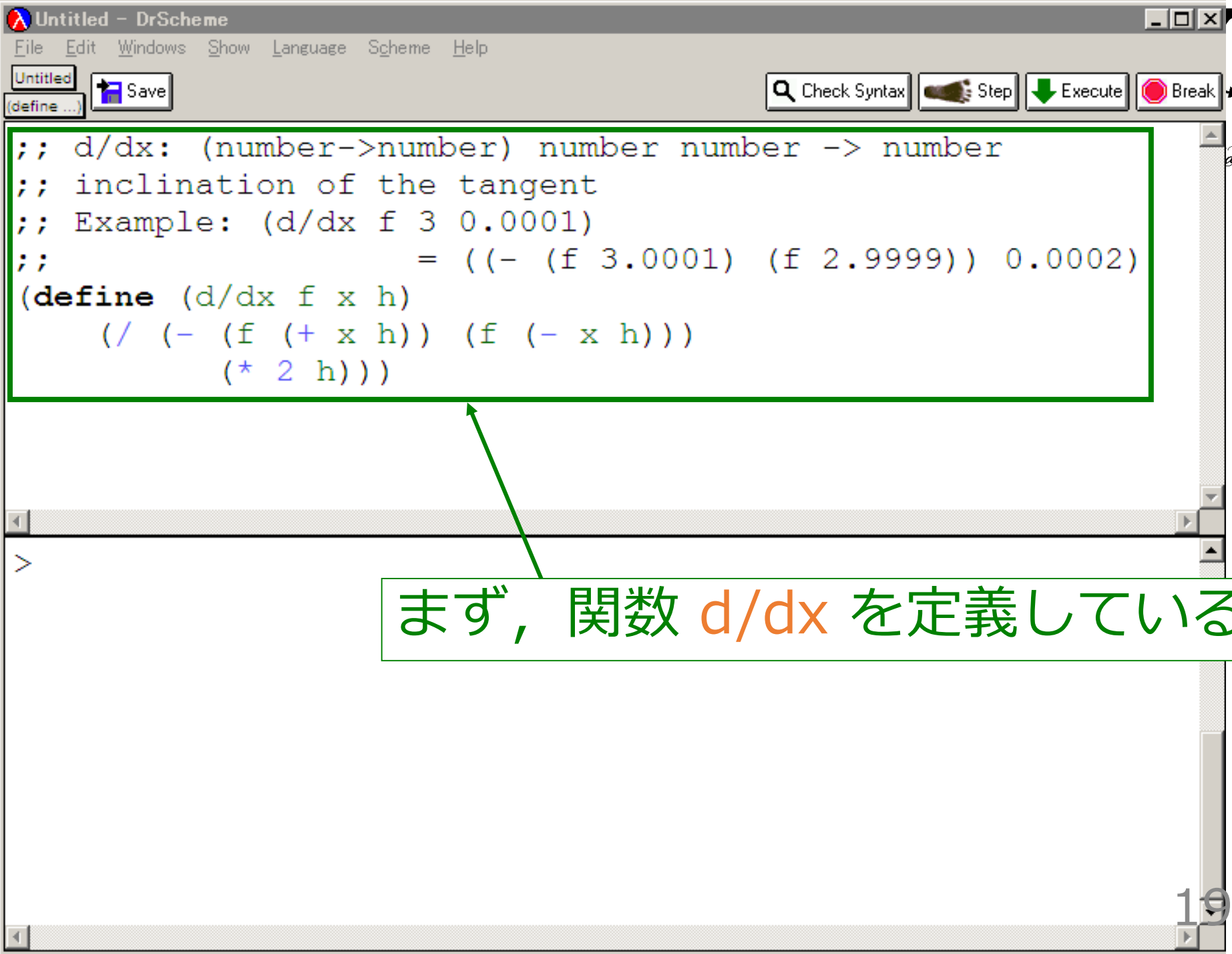
1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

```
;; d/dx: (number->number) number number -> number
;; inclination of the tangent
;; Example: (d/dx f 3 0.0001)
;;                  = ((- (f 3.0001) (f 2.9999)) 0.0002)
(define (d/dx f x h)
   (/ (- (f (+ x h)) (f (- x h)))
         (* 2 h)))
(define (f2 x)
   (- (* x x) 2))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(d/dx f2 3 0.0001)
```

☆　次は，例題２に進んでください


Untitled - DrScheme
File Edit Windows Show Language Scheme Help
Untitled
(define ...)
Save
Check Syntax
Step
Execute
Break
;; d/dx: (number->number) number number -> number
;; inclination of the tangent
;; Example: (d/dx f 3 0.0001)
;;   = ((- (f 3.0001) (f 2.9999)) 0.0002)
(define (d/dx f x h)
(/ (- (f (+ x h)) (f (- x h)))
(* 2 h)))
>
まず，関数 d/dx を定義している

Untitled - DrScheme

File Edit Windows Show Language Scheme Help

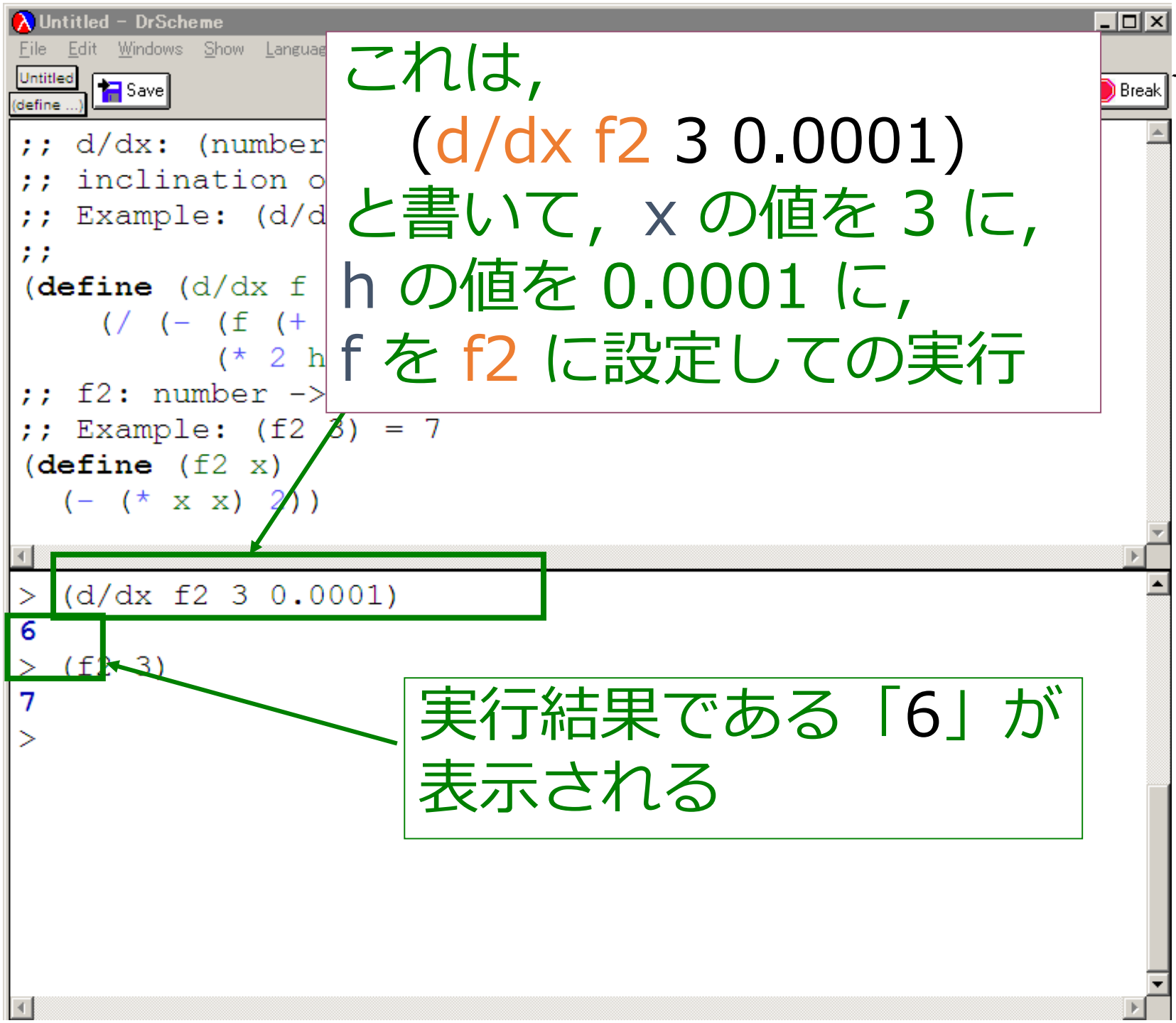
Untitled (define ...) Save Check Syntax Step Execute Break

```
;; d/dx: (number->number) number number -> number
;; inclination of the tangent
;; Example: (d/dx f 3 0.0001)
;;                    = ((- (f 3.0001) (f 2.9999)) 0.0002)
(define (d/dx f x h)
    (/ (- (f (+ x h)) (f (- x h)))
          (* 2 h)))
;; f2: number -> number
;; Example: (f2 3) = 7
(define (f2 x)
  (- (* x x) 2))
```

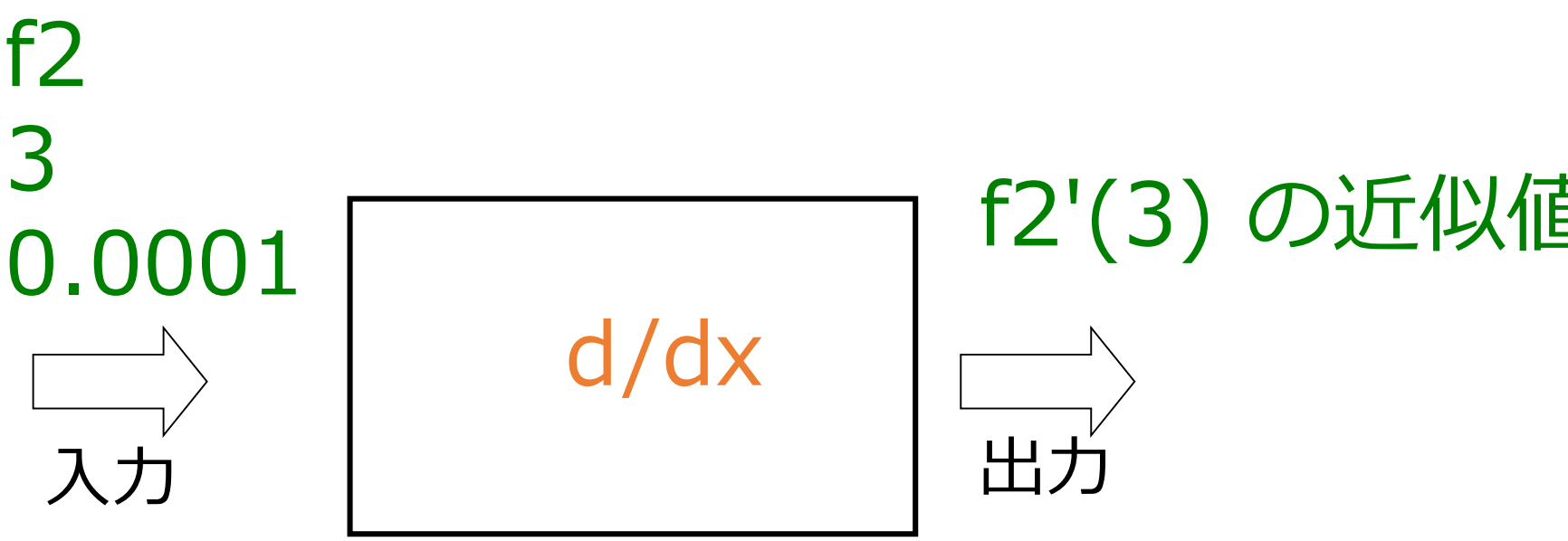
>

次に関数 f2 を定義している

```
(define (f2 x)
    (- (* x x) 2))
```


Untitled - DrScheme
File Edit Windows Show Languag
Untitled
(define ...)
Save
Break
;; d/dx: (number
;; inclination o
;; Example: (d/d
;;
(define (d/dx f
(/ (- (f (+
(* 2 h
;; f2: number ->
;; Example: (f2 3) = 7
(define (f2 x)
(- (* x x) 2))
> (d/dx f2 3 0.0001)
6
> (f2 3)
7
>
これは,
(d/dx f2 3 0.0001)
と書いて，x の値を 3 に,
h の値を 0.0001 に,
f を f2 に設定しての実行
実行結果である「6」が
表示される

入力と出力



入力は２つの数値と関数

出力は数値

d/dx は，関数を入力とするような関数（つまり高階関数）

d/dx 関数


```
;; d/dx: (number->number) number number ->
  number
;; inclination of the tangent
;; Example: (d/dx f 3 0.0001)
;;                  = ((- (f 3.0001) (f 2.9999)) 0.0002)
(define (d/dx f x h)
    (/ (- (f (+ x h)) (f (- x h)))
          (* 2 h)))
```

関数が,
「d/dx」の入力になっている

(dx/d f2 3 0.0001) から 6 が得られる過程の概略

(dx/d f2 3 0.0001) 最初の式

```
= (/ (- (f2 (+ 3 0.0001)) (f2 (- 3 0.0001)))
     (* 2 0.0001))
= (/ (- (- (* (+ 3 0.0001) (+ 3 0.0001)) 2)
          (- (* (- 3 0.0001) (- 3 0.0001)) 2)
       (* 2 0.0001)))
= …
```

コンピュータ内部での計算

= 6 実行結果

但し，f2 は

```
(define (f2 x)
    (- (* x x) 2))
```

(dx/d f2 3 0.0001) から 6 が得られる過程の概略

(dx/d f2 3 0.0001)

= (/ (- (f2 (+ 3 0.0001)) (f2 (- 3 0.0001)))
(* 2 0.0001))

= (/ (- (- (* (+ 3 0.0001) (+ 3 0.0001)) 2)

これは,

```
(define (d/dx  f x h)
    (/   (-  (f  (+ x h))   (f  (- x h)))
         (* 2 h)))
```

の x を 3 で, h を 0.0001 で, f を f2 で置き換えたもの

(dx/d f2 3 0.0001) から 6 が得られる過程の概略


```
(dx/d f2 3 0.0001)
= (/ (- (f2 (+ 3 0.0001)) (f2 (- 3 0.0001)))
     (* 2 0.0001))
= (/ (- (- (* (+ 3 0.0001) (+ 3 0.0001)) 2)
        (- (* (- 3 0.0001) (- 3 0.0001)) 2)
     (* 2 0.0001)))
= …
```

これは,

```
(define (f2 x)
    (- (* x x) 2))
```

の x を (+ 3 0.0001) や (- 3 0.0001) で置き換えたもの

例題２．台形則による数値積分

- 台形則による数値積分の関数 trapezoid を作り，実行する
  - 但し，

    積分したい関数　$f(x) = e^{-x^2}$

    積分区間:　　[a, b]

    区間数:　　n

「例題２．台形則による数値積分」の手順

1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

```
(define (trapezoid-iter f a h k)
  (cond
    [(= k 1) (f (+ a h))]
    [else (+ (f (+ a (* h k)))
       (trapezoid-iter f a h (- k 1)))]))
;; trapezoid: number number number -> number
;; to compute the area under the graph of f between a and b
(define (trapezoid f a b n)
  (* (/ (- b a) n)
     (+ (trapezoid-iter f
                        a
                        (/ (- b a) n)
                        (- n 1))
        (/ (f a) 2)
        (/ (f b) 2))))
(define (f2 x)
   (exp (- (* x x))))
```

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(trapezoid f2 0 1 1000)
```

☆　次は，課題に進んでください

Untitled – DrScheme

File Edit Windows Show Language Scheme Help

Untitled (define ...) Save Check Syntax Step Execute Break

```
(define (trapezoid-iter f a h k)
  (cond
    [(= k 1) (f (+ a h))]
    [else (+ (f (+ a (* h k)))
             (trapezoid-iter f a h (- k 1)))]))
;; trapezoid: number number number -> number
;; to compute the area under the graph of f between a and b
(define (trapezoid f a b n)
  (* (/ (- b a) n)
     (+ (trapezoid-iter f
                        a
                        (/ (- b a) n)
                        (- n 1))
        (/ (f a) 2)
        (/ (f b) 2))))
```

```
>
```

まず，関数 trapezoid-iter, trapezoid を定義している

File Edit Windows Show Language Scheme Help

Untitled (define ...) Save Check Syntax Step Execute Break

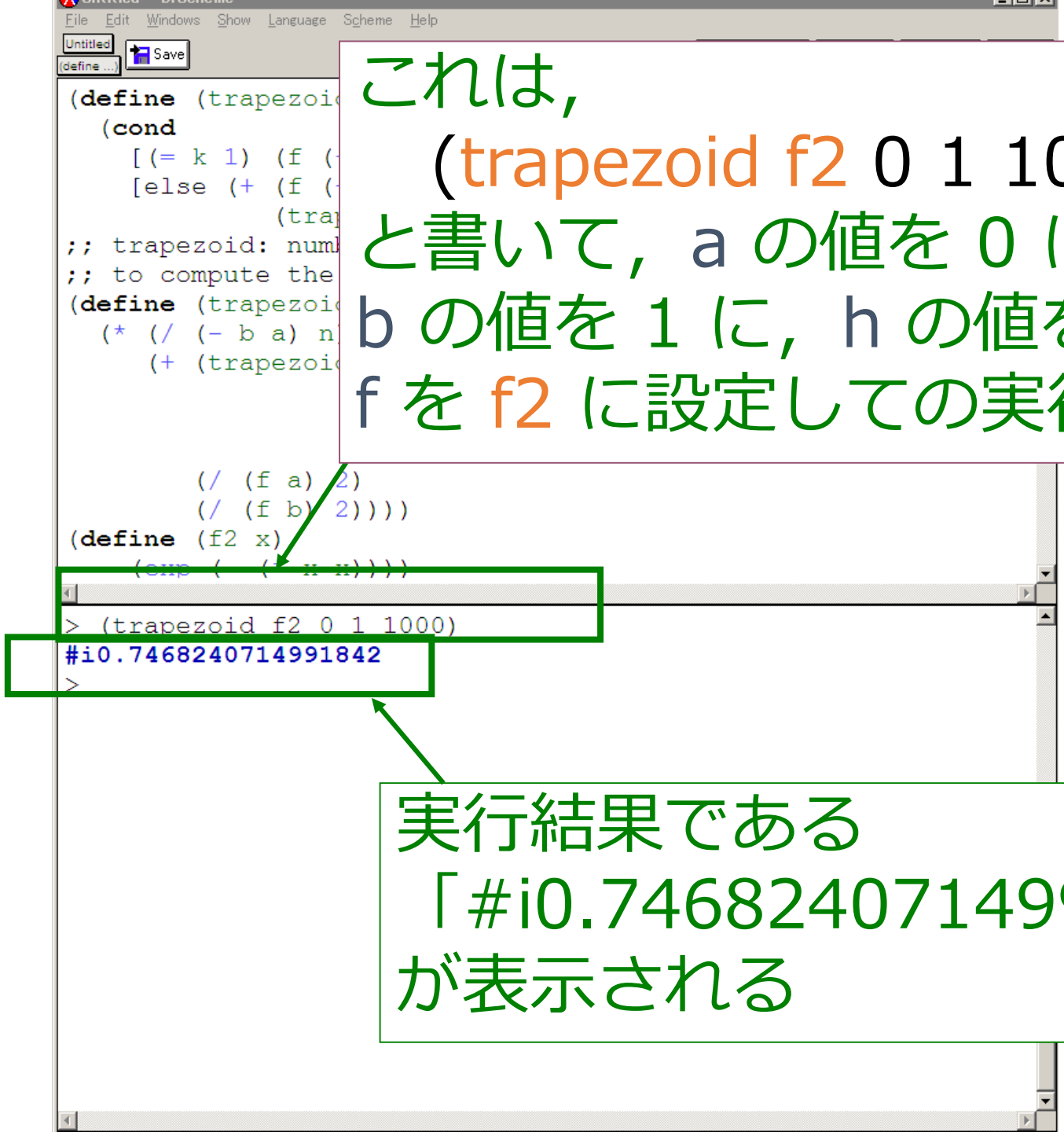
```
(define (trapezoid-iter f a h k)
  (cond
    [(= k 1) (f (+ a h))]
    [else (+ (f (+ a (* h k)))
             (trapezoid-iter f a h (- k 1)))]))
;; trapezoid: number number number -> number
;; to compute the area under the graph of f between a and b
(define (trapezoid f a b n)
  (* (/ (- b a) n)
     (+ (trapezoid-iter f
                        a
                        (/ (- b a) n)
                        (- n 1))
        (/ (f a) 2)
        (/ (f b) 2))))
(define (f2 x)
    (exp (- (* x x))))
```

```
>
```

次に関数 f2 を定義している

```
(define (f2 x)
    (exp (- (* x x))))
```

これは，数値積分したい関数


File Edit Windows Show Language Scheme Help
Untitled
Save
(define (trapezoi
(cond
[(= k 1) (f (
[else (+ (f (
(tra
;; trapezoid: numb
;; to compute the
(define (trapezoi
(* (/ (- b a) n
(+ (trapezoi
(/ (f a) 2)
(/ (f b) 2))))
(define (f2 x)
> (trapezoid f2 0 1 1000)
#i0.7468240714991842
>
これは,
(trapezoid f2 0 1 1000)
と書いて，a の値を 0 に,
b の値を 1 に，h の値を 1000 に,
f を f2 に設定しての実行
実行結果である
「#i0.7468240714991842」
が表示される


```
(define (trapezoid-iter f a h k)
  (cond
    [(= k 1) (f (+ a h))]
    [else (+ (f (+ a (* h k)))
       (trapezoid-iter f a h (- k 1)))]))
;; trapezoid: number number number -> number
;; to compute the area under the graph of f
  between a and b
(define (trapezoid f a b n)
  (* (/ (- b a) n)
     (+ (trapezoid-iter f
                        a
                         (/ (- b a) n)
                         (- n 1))
        (/ (f a) 2)
        (/ (f b) 2))))
```

例題３．台形則の計算の精度

- 例題２での台形則の計算の精度が，分割幅を小さくするほど高精度になることを確かめる．

  - 但し，

    積分したい関数　$f(x) = e^{-x^2}$

    積分区間　0 から 1

    分割数　10, 20, 40, 80, 160, 320, 640, 1280 の８通り

例題３．「台形則の計算の精度」の手順 (1/2)

1. 次を「定義用ウインドウ」で，実行しなさい
   - 入力した後に，Execute ボタンを押す

```
(define (trapezoid-iter f a h k)
  (cond
    [(= k 1) (f (+ a h))]
    [else (+ (f (+ a (* h k)))
       (trapezoid-iter f a h (- k 1)))]))
;; trapezoid: number number number -> number
;; to compute the area under the graph of f between a and b
(define (trapezoid f a b n)
  (* (/ (- b a) n)
     (+ (trapezoid-iter f
                        a
                        (/ (- b a) n)
                        (- n 1))
        (/ (f a) 2)
        (/ (f b) 2))))
(define (f2 x)
   (exp (- (* x x))))
```

これは，例題２と同じ

例題３．「台形則の計算の精度」の手順（2/2）

2. その後，次を「実行用ウインドウ」で実行しなさい

```
(trapezoid 0 1 10)
(trapezoid 0 1 20)
(trapezoid 0 1 40)
(trapezoid 0 1 80)
(trapezoid 0 1 160)
(trapezoid 0 1 320)
(trapezoid 0 1 640)
(trapezoid 0 1 1280)
```

☆　次は，課題に進んでください

数値積分の精度

- 分割幅を小さくするほど高精度

おわりに

- 近似値を求める手法
  - 数値微分，数値積分
  - 十分に役に立つ
  - 「必ず誤差が生じる」ことを意識しながら使うことが必要

13-3 課題

課題 1

- f(x) = x cos x について，f'(0), f'(0.2), f'(0.4), f'(0.6) の値を報告しなさい
  - 関数 d/dx （授業の例題１）を使いなさい
  - h = 0.1 としなさい

演習２

- f(x) = x cos x について，h = 0.1, 0.01, 0.001 と変えて関数 d/dx を実行し，結果を比べなさい
  - h と誤差の関係が分かるグラフを作成せよ

演習３

- 台形則を使って，次を計算せよ

$$\log_e 2 = \int_0^1 \frac{x}{1+x}\,dx$$

- 計算結果を，以下と比較せよ

```
> (log 2)
#i0.6931471805599453
```

DrScheme での (log 2) の実行結果

演習 4

- 演習 3 について，区間数 n を，n = 4, 8, 16, 32, 64, 128 と変えて計算を行い，刻み幅と誤差の関係を調べよ
  - 区間数 n と誤差の関係を書いたグラフを作成せよ
  - この場合，おおよそ次の関係が成り立っていることを確認せよ

$$Sn - I \approx \frac{c}{n^2}$$ （c は定数）

演習４．関数のグラフ

- 関数 d/dx （例題１）を使って，関数のグラフをグラフィックスで表示させなさい
  - プログラムは次ページ以降にある．このプログラムを実行させ，実行結果を報告しなさい
  - 但し，実行結果の報告では，必ず 関数 f2 を別のものに書き換えて実行しなさい


```
;; d/dx: (number->number) number number -> number
;; inclination of the tangent
;; Example: (d/dx f 3 0.0001)
;;                  = ((- (f 3.0001) (f 2.9999)) 0.0002)
(define (d/dx f x h)
  (/ (- (f (+ x h)) (f (- x h)))
     (* 2 h)))
;; samples: (number->number) number number number ->
   list of 'posn' structure
(define (samples f a h k)
  (cond
    [(= k 1) (cons (make-posn (+ a h)
                              (f (+ a h)))
                   empty)]
    [else (cons (make-posn (+ a (* h k))
                           (f (+ a (* h k))))
                (samples f a h (- k 1)))]))
```


```
; window size
(define WX 500)
(define WY 500)
; grid color, axis color and
(define GRID_COLOR 'blue)
(define AXIS_COLOR 'red)
(define GRAPH_COLOR 'red)
; draw-a-line
(define (sessen x px py d)
  (+ (* d (- x px)) py))
(define (draw-a-sessen from to px py d x0 y0 sx sy)
  (draw-solid-line (make-posn (+ (* from sx) x0) (+ (* (sessen from
   px py d) sy) y0))
                   (make-posn (+ (* to   sx) x0) (+ (* (sessen to   px py d)
   sy) y0)) GRAPH_COLOR))
(define (draw-sessen f px x0 y0 sx sy)
  (draw-a-sessen (/ (- x0) sx) (/ (- WX x0) sx) px (f px) (d/dx f px
   0.00001) x0 y0 sx sy))
```


```
; draw-polyline
(define (draw-polyline a-poly)
  (cond
    [(empty? (rest a-poly)) true]
    [else (and
           (draw-solid-line (first a-poly)
                            (first (rest a-poly)) GRAPH_COLOR)
           (draw-polyline (rest a-poly)))]))
; draw-h-lines
(define (draw-h-lines start skip stop width)
  (cond
    [(>= start stop)
         (draw-solid-line (make-posn 0 stop) (make-posn
   width stop) GRID_COLOR)]
    [else
     (and
      (draw-solid-line (make-posn 0 start) (make-posn width
   start) GRID_COLOR)
      (draw-h-lines (+ start skip) skip stop width))]))
```


```
; draw-v-lines
(define (draw-v-lines start skip stop height)
  (cond
    [(>= start stop)
        (draw-solid-line (make-posn stop 0) (make-posn stop height)
   GRID_COLOR)]
    [else
     (and
      (draw-solid-line (make-posn start 0) (make-posn start height)
   GRID_COLOR)
      (draw-v-lines (+ start skip) skip stop height))]))
; (x0, y0) is the origin. sx and sy represent one grid size
(define (draw-grid x0 y0 sx sy)
  (and
   (draw-v-lines (- x0 (* (quotient x0 sx) sx)) sx (+ (* (quotient (- WX
   x0) sx) sx) x0) WY)
   (draw-h-lines (- y0 (* (quotient y0 sy) sy)) sy (+ (* (quotient (- WY
   y0) sy) sy) y0) WX)
   (draw-solid-line (make-posn 0 y0) (make-posn WX y0) AXIS_COLOR)
   (draw-solid-line (make-posn x0 0) (make-posn x0 WY) AXIS_COLOR)))
```


```
; (x0, y0) is the origin. sx and sy represent one grid size
(define (scaling a-list x0 y0 sx sy)
  (cond
    [(empty? a-list) empty]
    [else (cons (make-posn (+ (* (posn-x (first a-list)) sx) x0)
                           (+ (* (posn-y (first a-list)) sy) y0))
                (scaling (rest a-list) x0 y0 sx sy))]))
;
; f2: number -> number
(define (f2 x)
    (- (* x x) 2))
; (X0, Y0) reprerents the origin of the graph.
; SX and SY represent the size of one grid
(define X0 (/ WX 2))
(define Y0 (/ WY 2))
(define SX 50)
(define SY 50)
(define PX 0.5)
(start WX WY)
(draw-grid X0 Y0 SX SY)
(draw-polyline (scaling (samples f2 (/ (- X0) SX) (/ 1 SX) WX) X0 Y0 SX SY))
(draw-sessen f2 PX X0 Y0 SX SY)
```