アルゴリズムとデータ構造入門
1.手続きによる抽象の構築
1.3.2 高階手続きによる抽象化

奥 乃 博
大学院情報学研究科知能情報学専攻
知能メディア講座 音声メディア分野
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/08/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

12月8日3階大会議室で中間試験
TAのページがオープン，質問箱もあります
http://winnie.kuis.kyoto-u.ac.jp/~fukubaya/AlgDsWiki/

1

11月17日・本日のメニュー

高階手続きによる抽象化

1. 1.3.2 Constructing Procedures Using `Lambda`（復習）
2. 1.3.3 Procedures as General Methods
3. 1.3.4 Procedures as Returned Values

2

1.3.2 Local Variables with let

```
(define (f x y)
  (define (f-helper a b)
    (+ (* x (square a))
       (* y b)
       (* a b) ))
  (f-helper
    (+ 1 (* x y))
    (- 1 y) ))
```

```
(define (f x y)
  (let ((a (+ 1 (* x y)))
        (b (- 1 y)) )
    (+ (* x (square a))
       (* y b)
       (* a b) )))
```

```
(define (f x y)
  ((lambda (a b)
     (+ (* x (square a))
        (* y b)
        (* a b) ))
   (+ 1 (* x y))
   (- 1 y) ))
```

```
(let ((<v1> <e1>)
      (<v2> <e2>)
      …
      (<vn> <en>) )
  <body> )
```

シンタックス・シュガー

14

let* は、順番に変数を評価

```
(let ((x 5))
  (let* ((x 3)
         (y (+ x 2)) )
    (* x y) ))
```

Substition model

x=3, y=5

15

λ式に展開して考える

```
((lambda (x)
   ((lambda (x)
      ((lambda (y)
         (* x y) )
       (+ x 2) )
     3 )
 5 )
```

17

11月17日・本日のメニュー

高階手続きによる抽象化

1. 1.3.2 Constructing Procedures Using `Lambda'
2. **1.3.3 Procedures as General Methods**
3. 1.3.4 Procedures as Returned Values

18

1.3.3 Procedures as General Methods

Finding roots of equations by the half-interval method（区間二分法）

```
(define (search f neg-point pos-point)
  (let ((midpoint (average neg-point pos-point)))
    (if (close-enough? neg-point pos-point)
        midpoint
        (let ((test-value (f midpoint)))
          (cond ((positive? test-value)
                 (search f neg-point midpoint))
                ((negative? test-value)
                 (search f midpoint pos-point))
                (else midpoint))))))
```

19

Finding roots of equations by the half-interval method

```
(define (close-enough? x y)
  (< (abs (- x y)) 0.001))

(define (half-interval-method f a b)
  (let ((a-value (f a))
        (b-value (f b)))
    (cond ((and (negative? a-value) (positive? b-value))
           (search f a b))
          ((and (negative? b-value) (positive? a-value))
           (search f b a))
          (else
           (error "Values are not of opposite sign" a b))
    )))
```

2点の値の符号が異なるかのチェックを行う

L:開始時の区間長、T:誤差許容度、

ステップ数: $\Theta(\log(L/T))$

20

Finding fixed points of functions（不動点）

```
(define tolerance 0.00001)
(define (fixed-point f first-guess)
  (define (close-enough? v1 v2)
    (< (abs (- v1 v2)) tolerance))
  (define (try guess)
    (let ((next (f guess)))
      (if (close-enough? guess next)
          next
          (try next))))
  (try first-guess))
```

抽象化すると

xが不動点 $x = f(x)$ $f(x), f(f(x)), f(f(f(x))), \cdots$

Finding fixed points of functions（不動点）

```
(fixed-point cos 1.0)
(fixed-point
  (lambda (y) (+ (sin y) (cos y)))
  1.0 )
```

$y = \cos y$

$y = \sin y + \cos y$

$y * y = x$ より $y = \frac{x}{y}$ と書くと、

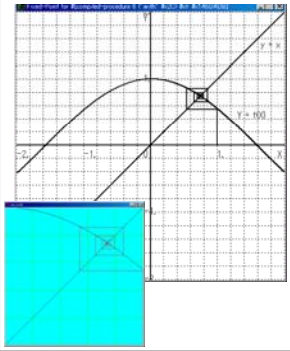
次の関数の不動点探索となる $y \mapsto \frac{x}{y}$

```
(define (sqrt x)
  (fixed-point (lambda (y) (/ x y))
               1.0))
```

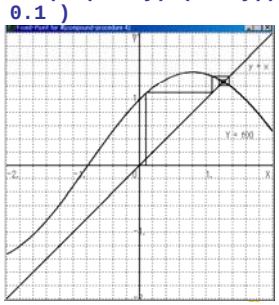
22

Finding fixed points of functions(不動点)

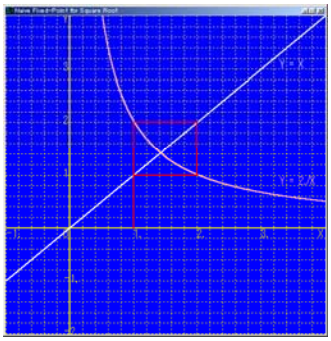
```
(fixed-point cos 1.0)
```



```
(fixed-point
  (lambda (y)
    (+ (sin y) (cos y)))
0.1 )
```



Naïve Fixed Point (sqrt 2)



$$y \mapsto \frac{x}{y}$$

急いては事を仕損じる

アイデア倒れ

27

Average damping（平均緩和法）

One way to control such ocillations:

Redefine a new function

$$y \mapsto \frac{1}{2}\left(y + \frac{x}{y}\right)$$

```
(define (sqrt x)
  (fixed-point
      (lambda (y) (average y (/ x y)))
      1.0) )
```

Average damping（平均緩和法）

28

11月17日・本日のメニュー

高階手続きによる抽象化

1. 1.3.2 Constructing Procedures Using `Lambda'
2. 1.3.3 Procedures as General Methods
3. **Intermission**
4. 1.3.4 Procedures as Returned Values

30

WHAT IS THIS INSTRUMENT?

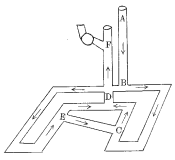
計算尺（slide rule, slipstick）
対数による積の計算
乗算→対数→加算
累乗→対数→乗算
2^{30} はいくら
2^{10} →対数→10log2 →3.01
$2^{10} \fallingdotseq 10^3$ →1K
$2^{30} \fallingdotseq 10^9$ →1G
音楽のピッチ
音の知覚

31

Slide Rule

鉱山用通気計算尺
G,R,R,I,Mine Ventilation Slide Rule

一資源試型計算尺使用法説明書



ヘンミ計算尺株式会社
東京都千代田区神田駿河台４－４
－２６３１（代）
（６９０６Ｎ）

- アポロ13
- ミクロの決死隊
- タイタニック

32

Scale Label	Value Relative to C/D
A	X^2
B	X^2
C	X
CF	X x pi
CF/M	X x $\log_e 10$
CI	1 / X
CIF	1 / (pi x X)
D	X
DF	X x pi
DF/M	X x $\log_e 10$
DFM	X x $\log_e 10$
DI	1 / X
DIF	1 / (pi x X)
E	e^x
H	-
H1, H2	Square root of $(1+X^2)$
HC	-

K	X^3
KZ	X x 360
L	log X
Lg	log X
Ln	ln X
LL0	$e^{0.001x}$
LL1 or ZZ1	$e^{0.01x}$
LL2 or ZZ2	$e^{0.1x}$
LL3 or ZZ3	e^x
LL00 or LL/0	e^{-x}
LL01 or LL/1	$e^{-0.1x}$
LL02 or LL/2	$e^{-0.01x}$
LL03 or LL/3	$e^{-0.001x}$
M	log X
P	Square root of $(1-X^2)$
p%	-
P1, P2	Square root of $(1-X^2)$

R	1 / X
R1, R2	Square root of X
S	sin X
Sh1, Sh2	sinh X
Sq1, Sq2	Square root of X
SRT	sin X, tan X
ST	sin, tan X
T	tan X, cotan X
T1, T3	tan X, cotan X
T2	tan X, cotan X
Th	tanh X
V	Volts
W1, W2	Square root of X
Z	X
ZZ1, ZZ2, ZZ3	e^x

大きな数・小さな数

deca	da	$\times 10^1$
hecto	h	$\times 10^2$
kilo	K	$\times 10^3$
mega	M	$\times 10^6$
giga	G	$\times 10^9$
tera	T	$\times 10^{12}$
peta	P	$\times 10^{15}$
exa	E	$\times 10^{18}$
zetta	Z	$\times 10^{21}$
yotta	Y	$\times 10^{24}$

deci	d	$\times 10^{-1}$
centi	c	$\times 10^{-2}$
milli	m	$\times 10^{-3}$
micro	μ	$\times 10^{-6}$
nano	n	$\times 10^{-9}$
pico	p	$\times 10^{-12}$
femto	f	$\times 10^{-15}$
atto	a	$\times 10^{-18}$
zepto	z	$\times 10^{-21}$
yocto	y	$\times 10^{-24}$

34

10^1	ten *or* decad
10^2	hundred *or* hecatontad
10^3	thousand *or* chiliad
10^4	myriad
10^5	lac *or* lakh
10^6	million
10^7	crore
10^8	myriamyriad
10^9	milliard *or* billion
10^{12}	trillion
10^{15}	quadrillion
10^{18}	quintillion

10^{21}	sextillion
10^{24}	septillion
10^{33}	decillion
10^{63}	vigintillion
10^{303}	centillion
10^{100}	googol
10^{googol}	googolplex
10^N	*N* plex
10^{-N}	*N* minex

35

88plex	無量大数	20plex	垓	4minex	絲 (糸)
80plex	不可思議	16plex	京	5minex	忽
72plex	那由他	12plex	兆	6minex	微
64plex	阿僧祇	8plex	億	7minex	纖 (繊)
56plex	恒河砂	4plex	萬(万)	8minex	沙
48plex	極	3plex	千	9minex	塵
44plex	載	2plex	百	10minex	埃
40plex	正	1plex	十	11minex	渺
36plex	澗	0plex	一	12minex	漠
32plex	溝	1minex	分	13minex	模糊
28plex	穣	2minex	厘	14minex	逡巡
24plex	杼 (禾偏)	3minex	毫(毛)	15minex	須臾

36

1minex	分	13minex	模糊
2minex	厘	14minex	逡巡 シュンジュン
3minex	毫 (毛) モウ	15minex	須臾シュユ
4minex	絲 (糸) シ	16minex	瞬息シュンソク
5minex	忽 コツ	17minex	弾指ダンシ
6minex	微 ビ	18minex	殺那
7minex	纖 (繊)セン	19minex	六徳 リットク
8minex	沙シャ	20minex	虚
9minex	塵ジン	21minex	空
10minex	埃アイ	22minex	清
11minex	渺ビョウ	23minex	浄
12minex	漠バク		

37

Rules of Thumb

1. **Rule of 72 : $r \times y = 72$**
   For y years with an interest rate of r % per year $\Rightarrow$ roughly double.
2. ***π seconds is a nanocentury.***
3. **1 *year* $= 3.155 \times 10^7$ *seconds***

38

emacs/meadow/mule

1. すべての入力は評価される
   - 単純な文字 ⇒ 自分が返される(self-evaluating)
2. コマンドは連想型
   - f(orward), b(ackward), e(nd), a(初め)
   - p(revious), n(ext)
   - d(delete), k(ill)
   - control-key(C-): 文字単位のコマンド
   - meta-key(M-) : 単語単位のコマンド(モードに依存)
   - control-meta-key(C-M-):S式単位のコマンド
3. Incremental search(C-s):逐次探索
4. C-x は拡張コマンド C-xC-s (save), C-xC-f (find)
5. ファイルの属性(ext)によりモード自動設定
6. タブで自動字下げ, 閉じ括弧で対応する開き括弧が点滅
7. M-x apropos で関連情報を検索するのがよい.

39

11月17日・本日のメニュー

高階手続きによる抽象化

1. 1.3.2 Constructing Procedures Using `Lambda'
2. 1.3.3 Procedures as General Methods
3. **Intermission**
4. **1.3.4 Procedures as Returned Values**

40

平均緩和法を不動点手続きから見直す

```
(define (sqrt x)
  (fixed-point (lambda (y) (average y (/ x y)))
               1.0))
```

$y \mapsto \frac{x}{y}$

平均緩和法を不動点手続きの観点から眺めると

$y \mapsto \frac{1}{2}\left(y + \frac{x}{y}\right)$

```
(define (average-damp f)
  (lambda (x) (average x (f x))))

((average-damp square) 10)

(define (sqrt x)
  (fixed-point
    (average-damp (lambda (y) (/ x y)))
    1.0))
(define (cube-root x)
  (fixed-point
    (average-damp (lambda (y) (/ x (square y))))
    1.0))
```

$y \mapsto \frac{x}{y^2}$

$y \mapsto \frac{1}{2}\left(y + \frac{x}{y^2}\right)$

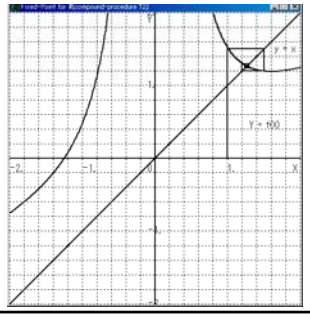
average-damp で統一的に捉えることが可能

41

Cubic-root の実行過程

```
(define (cube-root x)
  (fixed-point
    (average-damp (lambda (y) (/ x (square y))))
    1.0 ))
```

$$y \mapsto \frac{1}{2}\left(y + \frac{x}{y^2}\right)$$



ニュートン法を不動点手続きから見直す

```
(define (deriv g)
  (lambda (x)(/ (- (g (+ x dx)) (g x)) dx)) )
(define dx 0.00001)

(define (cube x) (* x x x))
((deriv cube) 5)
```

ニュートン法

$$y = x - \frac{g(x)}{g'(x)}$$

```
(define (newton-transform g)
  (lambda (x)(- x (/ (g x) ((deriv g) x)))) )

(define (newtons-method g guess)
  (fixed-point (newton-transform g) guess) )

(define (sqrt x)
  (newtons-method (lambda (y) (- (square y) x))
                  1.0))
```

更なる抽象化・first-class procedures

```
(define (fixed-point-of-transform g transform
    guess)
  (fixed-point (transform g) guess) )
```

1st method: 平均緩和法

```
(define (sqrt x)
  (fixed-point-of-transform
     (lambda (y) (/ x y))
     average-damp
     1.0 ))
```

2nd method:ニュートン法

```
(define (sqrt x)
  (fixed-point-of-transform
     (lambda (y) (- (square y) x))
     newton-transform
     1.0 ))
```

手続きの構築で何ら差別がない

First-class citizen （第1級市民）

第1級市民の“権利と特権”

- 変数で名前をつけることができる.
- 手続きへ引数として渡すことができる.
- 手続きを結果として返すことができる.
- データ構造の中に含めることができる.

Microsoft Longhorn will make RAW 'first class citizen.'
The Inquirer, Wed. Jun-8, 2005

45

手続き(関数)への演算: 導関数

```
(define dx 0.0001)
(define (ddx f x)
    (/ (- (f (+ x dx)) (f x)) dx) )
(ddx square 3) ⇒ 6.00010000001205
```

数値微分

我々はもっとスマート！ 導関数 いう考え方を採用

```
(define (deriv f)
   (lambda (x)
      (/ (- (f (+ x dx)) (f x)) dx) ))
((deriv square) 3) ⇒ 6.00010000001205
((deriv (deriv square)) 3)⇒ 1.99999998
(define (new-ddx f x)
    ((deriv f) x) )
```

46

手続き(関数)の合成: 高階導関数

この考え方を発展させ、高階導関数が構築できる

```
(define (compose f g)
  (lambda (x)
     (f (g x)) ))
(define 2nd-deriv (compose deriv deriv))
   ((2nd-deriv square) 3) ⇒ 1.9999999878
```

もちろん手続きの合成も

```
  ((compose square sqrt) 7) ⇒ 7.0
  ((2nd-deriv cos) pi) ⇒ 0.999999993922529
(define 3rd-deriv (compose deriv 2nd-deriv))
  ((3rd-deriv sin) pi) ⇒ 0.999999960615838
  ((4th-deriv cos) pi) ⇒ 1.11022302462516
```

47

Let's Play JMC with your num.

```
(define (jmc n)
  (if (> n 100)
      (- n 10)
      (jmc (jmc (+ n 11)))
  ))
```

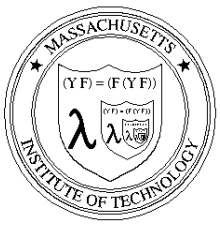
- 各自、次の式を求めよ

(jmc (modulo 学籍番号 100))

49

50

補足: Fixed Point




```
(define (jmc n)
   (if (> n 100)
       (- n 10)
       (jmc (jmc (+ n 11))) ))
```

(fixed-point jmc 1) ⇒ ?

(Y F) = (F (Y F)) Y operator
(不動点となる手続きを作成)

```
(Y jmc) = (F (Y jmc))
        = (lambda (n)
            (if (> n 100) (- n 10) ?) )
```

51

Fixed Point Operator F

```
(define (Y F)
  (lambda (s)
    (F (lambda (x) (lambda (x) ((s s) x)))
       (lambda (s) (F (lambda (x) ((s s) x))))
    )))
```

再帰呼び出しに無名手続きを使いたい

(Y F) = (F (Y F))

詳しくは、Church numeralの項で説明。

52

宿題:11月25日正午締切

- 不動点の考え方を習得すること
- 宿題は、次の2題:
- Ex.1.35, 1.41, 1.42, 1.43.
- 実行例を添付

DON'T PANIC!

54