**Structures and Interpretation of Computer Program**

**Exercise Chapter 1.3 Name:** Wan Huzaifah bin Wan Azhar

**Exercise 1.3.3 Procedures as General Method**

1. Approximating fixed point

(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))

(display (fixed-point (lambda (x) (+ 1 (/ 1 x))) 1.0))

1. Modifying fixed point

(define (average x y) (/ (+ x y) 2))

(define tolerance 0.00001)

(define (fixed-point f first-guess)

(define (close-enough? v1 v2)

(< (abs (- v1 v2)) tolerance))

(define (try guess)

(newline)

(display guess)

(let ((next (f guess)))

(if (close-enough? guess next)

next

(try next))))

(try first-guess))

(

(fixed-point (lambda (x) (/ (log 1000) (log x))) 10.0)

(newline)

fixed-point (lambda (x) (average x (/ (log 1000) (log x)))) 10.0)

Output:

10.0

2.9999999999999996

6.2877098228681545

3.7570797902002955

5.218748919675316

4.1807977460633134

4.828902657081293

4.386936895811029

4.671722808746095

4.481109436117821

4.605567315585735

4.522955348093164

4.577201597629606

4.541325786357399

4.564940905198754

4.549347961475409

4.5596228442307565

4.552843114094703

4.55731263660315

4.554364381825887

4.556308401465587

4.555026226620339

4.55587174038325

4.555314115211184

4.555681847896976

4.555439330395129

4.555599264136406

4.555493789937456

4.555563347820309

4.555517475527901

4.555547727376273

4.555527776815261

4.555540933824255

10.0

6.5

5.095215099176933

4.668760681281611

4.57585730576714

4.559030116711325

4.55613168520593

4.555637206157649

4.55555298754564

4.555538647701617

Without average damping, the processing took 33 lines while average damping took 10 lines to converge.

1. Infinite continued fraction
2. Recursive

(define (cont-frac n d k)

(define (frac i)

(if (= i k) 0

(/ (n i)

(+ (d i) (frac (+ i 1))))))

(frac 1))

(display (cont-frac (lambda (i) 1.0)

(lambda (i) 1.0)

12))

It took k > 11 to get accurate up to 4 decimal place.

1. Iterative

(define (cont-frac-iter n d k)

(define (frac i result)

(cond ((= i 1)

result)

((= i k)

(frac (- i 1) (/ (n i) (d i))))

(else

(frac (- i 1) (/ (n i) (+ (d i) result))))

))

(frac k (n 1)))

(display (cont-frac-iter (lambda (i) 1.0)

(lambda (i) 1.0)

11))

There is no perfect iterative solution as the algorithm itself must be able to save a particular state and return the final answer. So the best solution is to calculate the lowest term first and go upwards.

1. Approximating e

(define (cont-frac n d k)

(define (frac i)

(if (= i k) 0

(/ (n i)

(+ (d i) (frac (+ i 1))))))

(frac 1))

(define (div-by-three x)

(= 0 (remainder x 3)))

;k is steps

(define (e k)

(cond ((< k 0) 0)

((or (= k 1) (= k 2))

k)

((div-by-three (- k 2))

(+ (e (- k 1)) (e (- k 2)) (e (- k 3))))

(else 1)

))

(define (euler x)

(+ 2 (cont-frac (lambda (i) 1.0)

(lambda (i) (e i)) x)))

(display (euler 1000))

Output:

2.7182818284590455

1. Approximating tan x

(define (cont-frac n d k)

(define (frac i)

(if (= i k)

0

(/ (n i)

(+ (d i) (frac (+ i 1))))))

(frac 1))

(define (tan-cf x k)

(cont-frac

(lambda (i) (if (= 1 i)

x

(\* (expt x 2) -1)))

(lambda (i) (if (= 1 i)

1

(- (\* i 2) 1)))

k))

(display (tan-cf 2.0 20))

Outputs:

-2.185039863261519

Which is the value of tan(2).