アルゴリズム・データ構造 I 第6回

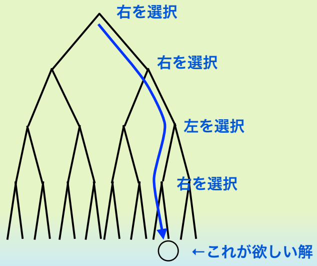
宝探しをしよう—2分法とニュートン法による方程式の解法

名城大学理工学部情報工学科

山本修身

２分探索法とは（復習）

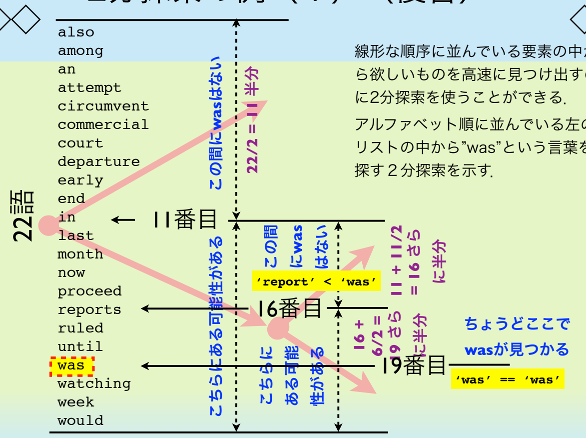
- 以下のような木構造（これを2分木という）があるとする．この木構造において，それぞれのレベルで2つうちのどちらかを選択すると，最終的に欲しい解に辿り着くことができる場合，この探索方法を**2分探索法（バイナリーサーチ：binary search）**と呼ぶ．



2分探索の例（１）（復習）

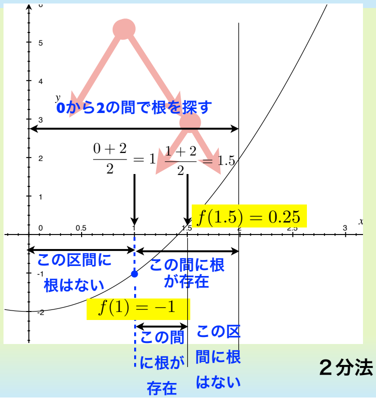
線形な順序に並んでいる要素の中から欲しいものを高速に見つけ出すのに2分探索を使うことができる．

アルファベット順に並んでいる左のリストの中から"was"という言葉を探す２分探索を示す．



２分探索の例（２）（復習）

- ２の平方根を計算してみる． $f(x) = x^2 - 2$ の根を求めれば良い



始めの状態：[0.0--2.0]
↓
次の状態：[1.0--2.0]
↓
その次の状態：[1.0-1.5]
↓
：

真の答えに近づく

近似解

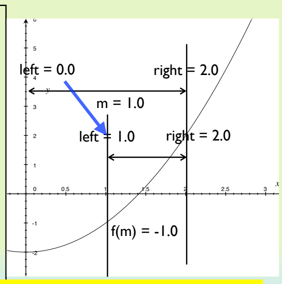
２分法（bisection method）

実際に２の平方根を計算する（復習）

２分探索によって２の平方根を計算してみる．f(x) = x^2 - 2は x>0で単調に増加する．確実に√2が入っている区間[0, 2]を初期区間にする

```
function binary(){
   var EPS = 1.0e-6
   var left = 0.0
   var right = 2.0
   while (right - left > EPS){
      var m = (left + right) / 2
      var v = m * m - 2
      if (v < 0) left = m
      else right = m
   }
   puts(left + ", " + right)
}
binary()
```

```
1.4142131805419922, 1.4142141342163086
```



ニュートン法について（復習）

ニュートン法は探索アルゴリズムではなく，数値を近似するためのアルゴリズムである．しかし，区分２分法で方程式の根を求めるなどの場合には，区分２分法と同じように動く．また，区間という概念を必要としないので，2次元以上の問題を解くこともできる．

ニュートン法はある近似値xからそれよりも良い近似値x'を計算する方法である．

$x' = x + \varepsilon$ とおく

$$f(x') = f(x+\varepsilon) = f(x) + f'(x)\varepsilon + o(\varepsilon) = 0$$

↓

$$\varepsilon = -f(x)/f'(x) \longrightarrow \boxed{x' = x - f(x)/f'(x)}$$

◇◇ニュートン法で√2を計算してみる (1)◇◇ (復習)

- √2を計算するということは，f(x) = x^2 - 2 の根を求めることである．すなわち，

$$\boxed{\begin{aligned} f(x) &= x^2 - 2 \\ f'(x) &= 2x \end{aligned}}$$

反復公式

$$\boxed{x' = x - \frac{x^2 - 2}{2x} = \frac{2x^2 - x^2 + 2}{2x} = \frac{x^2 + 2}{2x}}$$

$$x' = x - f(x)/f'(x)$$

◇◇ニュートン法で√2を計算してみる (2)◇◇ (復習)

実際のニュートン法のプログラムは以下のとおり．

ニュートン法の収束は非常に高速である（2次収束）

区分２分法の収束は1次収束

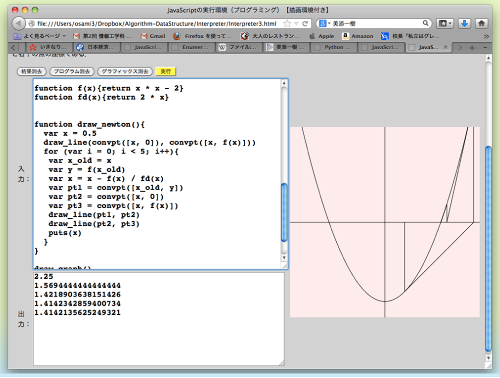
```
function newton2(){
   var x = 2.0
   for (var i = 0; i < 10; i++){
      x = (x * x + 2) / x / 2
      puts(i + " " + x)
   }
}

newton2()
```

4回目で収束している

```
0 1.5
1 1.4166666666666667
2 1.4142156862745099
3 1.4142135623746899
4 1.414213562373095
5 1.414213562373095
6 1.414213562373095
7 1.414213562373095
8 1.414213562373095
9 1.414213562373095
```


◇◇ Newton法の近似過程 ◇◇

- 前のスライドのプログラムを実行したところ



◇◇ 反復法を用いた連立一次方程式の解法 (1) ◇◇

以下のような連立1次方程式を解く． $\overline{x} = A^{-1}b$

$$Ax = b$$

$A = D + U + L$ として，おけば，$Dx = b - (U + L)x$

となるので， $x = D^{-1}(b - (U + L)x)$ 従って，

$F(x) = D^{-1}(b - (U + L)x)$ と定義すれば，

反復 $x_{n+1} = F(x_n)$ によって解を求めることができる．

ただし，収束するには条件 $\left|\det D^{-1}(U + L)\right| < 1$ が成り立たないといけない．また，$D^{-1}(U+L)$の絶対値最大の固有値の絶対値が1以下であれば収束する．この方法は**ヤコビ法**と呼ばれるものである．

ただし，Dは対角行列，UとLはそれぞれ上と下三角行列

◇◇ 反復法を用いた連立一次方程式の解法 (2) ◇◇

- 以下のような方程式を解いてみる．

$$\begin{pmatrix} 1 & 2 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

これに反復法を適用してみる．

```
for (i = 0; i < 10; i++)
    puts(i * 5 + " " + jacobi(i * 5))
```

```
function jacobi(n){
    var a = 1, b = 2, c = 1, d = 4
    var e = 2, f = -3
    function F(p){
        var [x, y] = p
        return [(e - b * y) / a, (f - c * x) / d]
    }
    var p = [0, 0]
    for (var i = 0; i < n; i++){
        p = F(p)
    }
    return p
}
```

```
0 0,0
5 5.75,-2.0625
10 6.78125,-2.421875
15 6.9609375,-2.486328125
20 6.9931640625,-2.49755859375
25 6.998779296875,-2.49957275390625
30 6.999786376953125,-2.4999237060546875
35 6.999961853027344,-2.4999866485595703
40 6.999993324279785,-2.499997615814209
45 6.9999988079071045,-2.4999995827674866
```


◇◇ 反復法を用いた連立一次方程式の解法 (3) ◇◇

- 以下のような連立1次方程式を解いてみる．

$$\begin{pmatrix} 6 & 6 & -7 \\ 4 & 7 & 3 \\ 3 & -9 & 10 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -3 \\ 4 \\ 9 \end{pmatrix}$$

$$Ax = b$$

$$A = D + L + U$$

$$x' = D^{-1}(b - (L + U)x)$$

この式を用いて繰り返す

反復法を用いた連立一次方程式の解法 (3)

- プログラムは以下のようになる.

```
function jacobi(n){
    var A = [[6, 6, -7], [4, 7, 3], [3, -9, 10]]
    var b = [-3, 4, 9]
    var p = [0, 0, 0]
    var eps = 1.0e-12
    function res(){
        var ss = 0.0
        for (var i = 0; i < 3; i++){
            var s = b[i]
            for (var j = 0; j < 3; j++)
                s -= A[i][j] * p[j]
            ss += s * s
        }
        return ss
    }
        for (var k = 0; k < n; k++){
            if (res() < eps){
                puts("k = " + k)
                return p
            }
            var q = []
            for (var i = 0; i < 3; i++){
                var s = b[i]
                for (var j = 0; j < 3; j++)
                    if (i != j) s -= A[i][j]*p[j]
                q[i] = s / A[i][i]
            }
            p = q
        }
        return null
    }
```

```
k = 226
0.4075472847467725,
0.0037735089093031376,
0.7811320421977082
```

反復法を用いた連立一次方程式の解法 (4)

- 数式処理システム Maximaで解けば以下のようになる.

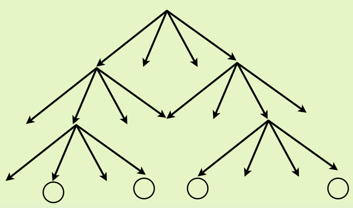
http://maxima.sourceforge.net/

```
(%i3) A: matrix([6, 6, -7], [4, 7, 3], [3, -9, 10]);
                                 [ 6   6   - 7 ]
                                 [             ]
(%o3)                            [ 4   7    3  ]
                                 [             ]
                                 [ 3  - 9  10  ]
(%i4) invert(A);
                              [  97     1    67   ]
                              [  ---   ---   ---  ]
                              [  795   265   795  ]
                              [                   ]
                              [   31   27     46  ]
(%o4)                         [ - ---  ---  - --- ]
                              [   795  265    795 ]
                              [                   ]
                              [   19   24     6   ]
                              [ - ---  ---   ---  ]
                              [   265  265   265  ]
```

```
(%i6) b: matrix([-3], [4], [9]);
                                 [ - 3 ]
                                 [     ]
(%o6)                            [  4  ]
                                 [     ]
                                 [  9  ]
(%i7) invert(A) . b;
                                 [ 108 ]
                                 [ --- ]
                                 [ 265 ]
                                 [     ]
                                 [  1  ]
(%o7)                            [ --- ]
                                 [ 265 ]
                                 [     ]
                                 [ 207 ]
                                 [ --- ]
                                 [ 265 ]
```

バイナリーサーチの変形版 1/4円を描画する (1) $f(x,y) = x^2 + y^2 - 1$

- バイナリーサーチは２分木を根から辿るとき，各ノードでかならず左右どちらかを選択する.
- 左右のどちらも選択する必要があるようなケースについて考える．さらに２分木ではなく，ここでは４分木について考える.

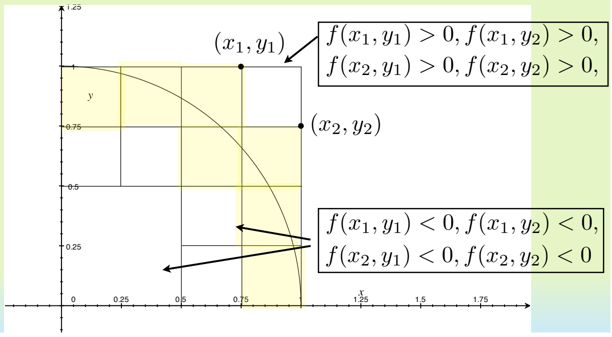


欲しい答えもいくつか存在する

バイナリーサーチの変形版 1/4円を描画する (2)

領域を1/4ずつに分けて，端点での値がすべて正または負の時はそれ以上描画しない.

$$f(x,y) = x^2 + y^2 - 1$$



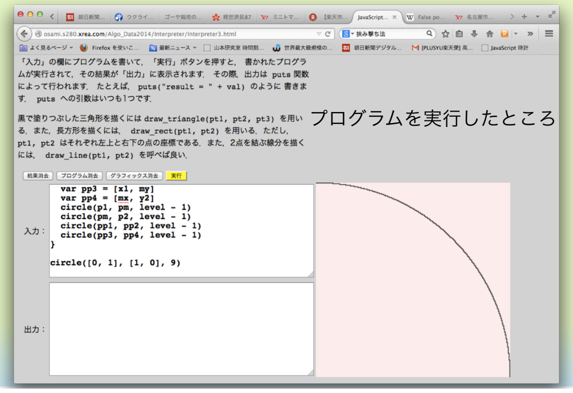
バイナリーサーチの変形版 1/4円を描画する (3)

関数circle(p1, p2) によってp1が左上でp2が右下の矩形を処理する

```
function f(x, y){
  return x * x + y * y - 1
}

function circle(p1, p2, level){
  if (level == 0){
    draw_rect(p1, p2)
    return
  }
  var x1 = p1[0]; var y1 = p1[1]
  var x2 = p2[0]; var y2 = p2[1]
  var c1 = f(x1, y1)
  var c2 = f(x1, y2)
  var c3 = f(x2, y1)
  var c4 = f(x2, y2)
  if (c1 > 0 && c2 > 0 && c3 > 0 && c4 > 0 ||
      c1 < 0 && c2 < 0 && c3 < 0 && c4 < 0)
        return
```

```
  var mx = (x1 + x2) / 2
  var my = (y1 + y2) / 2
  var pm = [mx, my]
  var pp1 = [mx, y1]
  var pp2 = [x2, my]
  var pp3 = [x1, my]
  var pp4 = [mx, y2]
  circle(p1, pm, level - 1)
  circle(pm, p2, level - 1)
  circle(pp1, pp2, level - 1)
  circle(pp3, pp4, level - 1)
}

circle([0, 1], [1, 0], 9)
```

バイナリーサーチの変形版 1/4円を描画する (4)



プログラムを実行したところ

ニュートン・ラフソン法 (1)

2変数以上の連立方程式を求める場合，ニュートン・ラフソン法を用いることができる．ここでは2次元の場合について解説する．

解きたい連立方程式

$$f_1(x,y)=0$$
$$f_2(x,y)=0$$

$$\boxed{\begin{aligned} f_1(x+\varepsilon, y+\delta) &= f_1(x,y) + \frac{\partial f_1(x,y)}{\partial x}\varepsilon + \frac{\partial f_1(x,y)}{\partial y}\delta \\ f_2(x+\varepsilon, y+\delta) &= f_2(x,y) + \frac{\partial f_2(x,y)}{\partial x}\varepsilon + \frac{\partial f_2(x,y)}{\partial y}\delta \end{aligned}}$$

$f_1(x+\varepsilon, y+\delta)=0, f_2(x+\varepsilon, y+\delta)=0$ と仮定する

ニュートン・ラフソン法 (2)

- 線形近似してから解く方法は1次元のときと同じ考え方

ヤコビ行列

$$\begin{pmatrix} \frac{\partial f_1(x,y)}{\partial x} & \frac{\partial f_1(x,y)}{\partial y} \\ \frac{\partial f_2(x,y)}{\partial x} & \frac{\partial f_2(x,y)}{\partial y} \end{pmatrix} \begin{pmatrix} \varepsilon \\ \delta \end{pmatrix} = -\begin{pmatrix} f_1(x,y) \\ f_2(x,y) \end{pmatrix}$$

これを解いて

$$\begin{pmatrix} \varepsilon \\ \delta \end{pmatrix} = -\begin{pmatrix} \frac{\partial f_1(x,y)}{\partial x} & \frac{\partial f_1(x,y)}{\partial y} \\ \frac{\partial f_2(x,y)}{\partial x} & \frac{\partial f_2(x,y)}{\partial y} \end{pmatrix}^{-1} \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \end{pmatrix}$$

反復公式は以下のとおり：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} \frac{\partial f_1(x,y)}{\partial x} & \frac{\partial f_1(x,y)}{\partial y} \\ \frac{\partial f_2(x,y)}{\partial x} & \frac{\partial f_2(x,y)}{\partial y} \end{pmatrix}^{-1} \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \end{pmatrix}$$

ニュートン・ラフソン法 (3)

以下のような方程式を考える．この方程式は少々工夫すると解くことができる．

$$\begin{cases} x^2+y^2 = 1 \\ xy = 1/3 \end{cases}$$

$$(x+y)^2 = x^2+y^2+2xy = 1+2/3 = 5/3$$

$$x+y = \pm\sqrt{5/3}$$

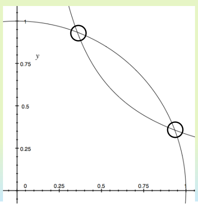
x, yは以下の方程式の2つの解となる

$$t^2 \pm \sqrt{5/3}t + 1/3 = 0$$

$$x, y = \frac{\sqrt{5/3} \pm \sqrt{1/3}}{2}$$

(x, y) = (0.934, 0.357)

(x, y) = (0.357, 0.934)



ニュートン・ラフソン法 (4)

この方程式を何も考えずにニュートン・ラフソン法で解いてみる．

$$\begin{cases} x^2+y^2 = 1 \\ xy = 1/3 \end{cases} \qquad \begin{aligned} f_1(x,y) &= x^2+y^2-1 \\ f_2(x,y) &= xy-1/3 \end{aligned}$$

$$J = \begin{pmatrix} 2x & 2y \\ y & x \end{pmatrix}$$

$$\boxed{\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} 2x & 2y \\ y & x \end{pmatrix}^{-1} \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \end{pmatrix}}$$

ニュートン・ラフソン法 (5)

- 実際にプログラムにすると以下のようになる．

$$\begin{cases} x^2+y^2 = 1 \\ xy = 1/3 \end{cases} \qquad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} 2x & 2y \\ y & x \end{pmatrix}^{-1} \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \end{pmatrix}$$

```
function f1(x, y){ return x * x + y * y - 1}
function f2(x, y){ return x * y - 1.0/3}

function work(n, x0, y0){
  var x = x0
  var y = y0
  for (var i = 0; i < n; i++){
    puts(x + " " + y)
    var det = 2 * (x * x - y * y)
    var a = x / det
    var b = -2 * y / det
    var c = -y / det
    var d = 2 * x / det
    var xx = x - a * f1(x, y) - b * f2(x, y)
    var yy = y - c * f1(x, y) - d * f2(x, y)
    x = xx
    y = yy
  }
}
work(12, 0.1, 0.2)
```

```
0.1 0.2
0.6055555555555556 2.322222222222222
0.3965490789697234 1.3519697909438335
0.3493498519397422 1.0015034291987086
0.3553401820150935 0.9369804950634335
0.3568145194377309 6 0.9341806098109103
0.35682208966479523 0.9341723590711898
0.3568220897730898 0.9341723589627157
0.3568220897730 9 0.934172358962715 8
0.3568220897730899 0.9341723589627157
0.35682208977308993 0.9341723589627158
0.35682208977308993 0.9341723589627157
```

$$\begin{pmatrix} 2x & 2y \\ y & x \end{pmatrix}^{-1} = \frac{1}{2(x^2-y^2)} \begin{pmatrix} x & -2y \\ -y & 2x \end{pmatrix}$$