JAPLA 研究会資料 2013/1/26

# Ｊは計算ではなく、データ処理の言語である
# －素数を求める2つのプログラムを例として－

西川　利男

## はじめに

Ｊのプログラムは BASIC などと違って、なかなか取り付きにくいと言う。私はそんなことはないと思うが、このような誤解をとくため、私のＪに対する考え方を述べて、
素数を求める2つのプログラムを例にあげて説明したいと思う。

**「Ｊは計算ではなく、データ処理の言語である。」**

ここで、データ処理とは何だろうか。Ｊの用語で言えば

データ＝名詞、処理＝動詞

であると考える。

データとは、英語の辞書を見ると、本来は datum であり、その複数形が data であるとなっている。すなわち、2つ以上の複数の値である。（1つだけの計算ならばプログラムを組むまでもない－電卓で1回やればすむ！）このとき値とは数値のこともあるが、一般にはコンピュータの適当な場所に格納された「もの」であり、これが文字を表すか、数を表すかは問題としない。その場所を示す名前＝名詞で扱われる。

そのような名詞を処理するものが動詞である。動詞つまり処理には次のようにいろいろなものがある。

・名詞の形を変える（加工、修正、削除など）
・複数の名詞を結合する
・名詞を2つに分ける
……
・名詞（この場合は数）の間で計算する

Ｊでは

「プログラムとはデータを変形することである。」
「計算はそのほんの一部でしかない！」

という哲学に基づいている。

そしてそのための動詞が極めて豊富である。これを活用することで、BASIC などではできない非常に効率的なプログラミングが可能となる。これが BASIC などと比べて、取り付きにくいということになるのではなかろうか。

この後、素数を求める2つのプログラム例で上の考え方に基づくプログラミングの実際を示そう。

それには、私のかつての APL の本「基礎からの APL」[1]から例題をとった。なお、Ｊの親である APL も当然ながら、同じ哲学の上に立つ言語である。

[1] 西川利男「基礎からの APL－解説と例題例解」サイエンスハウス(1991).

## 1.与えられた数までの素数を求める－配列と外積による方法 [1] p.101-104

まず、整数Ｎが素数であるかどうかは次の条件により決まる。

・整数Ｎを 1, 2, 3, …, Ｎまでの数で順々に割る。

・そのうち１とＮとだけで割り切れ、それ以外の数では割り切れない。

上の条件が成り立つときは …… 素数

さもないときは …… 合成数

たとえばＮ = 5 としたときは、次のようになる。

ここでＪでは割った余りは、動詞(|)で得られる。

```
５％１＝５あまり０　……　1|5 => 0
５％２＝２あまり１　……　2|5 => 1
５％３＝１あまり２　……　3|5 => 2
５％４＝１あまり１　……　4|5 => 1
５％５＝１あまり０　……　5|5 => 0
```

これはＪではまとめて、次のように行われる。

まず、それぞれの余りは

```
(1 2 3 4 5)| 5 => 0 1 2 1 0
```

これが割れ切れるかどうかは、余りが０か１かを見ればよい。

```
0 = 0 1 2 1 0 => 1 0 0 0 1
```

この数をみるには、合計をとればよい。これにはＪでは次のようにする。

```
+/ 1 0 0 0 1 => 2
```

つまり、余り０の場合は２回起り、5 は素数である。

Ｎ = 6 としたときは、

```
(1 2 3 4 5 6)| 6 => 0 0 0 2 1 0
0 = 0 0 0 2 1 0 => 1 1 1 0 0 1
+/ 1 1 1 0 0 1 => 4
```

つまり、余り０の場合が 1, 2, 3, 4 との４回起り、素数ではない。

それでは、Ｎのいろいろな値、たとえば Ｎ = 1, 2, 3, 4, 5, 6 に対して、それぞれが、素数かどうかを見るには、どうすればよいか?

ここで、Ｊの新しい機能、外積が登場する。余りを得る動詞(|)に、外積の副詞(/)を付け加えて、別の新しい動詞、余り外積(|/)を作り、これを作用させる。結果はつぎのようになる。

```
(1 2 3 4 5 6) |/ (1 2 3 4 5 6)
0 0 0 0 0 0
1 0 1 0 1 0
1 2 0 1 2 0
1 2 3 0 1 2
1 2 3 4 0 1
1 2 3 4 5 0
```

このようなタテヨコの値の並びはＪでは配列と呼ばれる。

この結果は、以下のことを意味する。タテ方向に見ていく。

まず、1列目は１２３４５６のそれぞれを１で割った余りを示す。
続いて、２列目は１２３４５６のそれぞれを２で割った余りを示す。
……
５列目は１２３４５６のそれぞれを５で割った余りを示す。
６列目は１２３４５６のそれぞれを６で割った余りを示す。
Ｊでは１２３４５６のような複数の数値は次のように (i.)で表すのがふつうである。
ここでＪでは整数は０から始まる(0-オリジン)とするので、i. ６ => ０１２３４５となり、これらをすべて１増やす動詞(>:)が必要である。（APL では 1-オリジンであるので不要）

```
   X =. >: i. 6
   X
１２３４５６
```

これを使えば、次のようになる。

```
    X |/ X
００００００
１０１０１０
１２０１２０
１２３０１２
１２３４０１
１２３４５０
```

さて、続いて余りが０になるかどうかを調べる。これは次のように行う。

```
   0 = X |/ X
１１１１１１
０１０１０１
００１００１
０００１００
０００００１
０００００１
```

ここで、０に等しいどうかは配列のそれぞれの要素に作用することに注意。
ここで得た配列をタテに見て、１が2個あれば、素数になる。

```
   +/ 0 = X |/ X
１２２３２４
```

２に等しい(1)、等しくない(0)のフラグは次のようにして得られる。

```
    2 = +/ 0 = X |/ X
０１１０１０
```

Ｘの中から１の場所の値を、動詞(#)により取り出せば、素数が得られる。

```
   (2 = +/ 0 = X |/ X) # X
２３５
```

このようにして、素数が取り出せた。これをプログラムすればよい。

```
   pri =: 3 : 0
X =. >: i. y.
(2 = +/ 0 = X |/ X) # X
)
   pri 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

## 2.与えられた数までの素数を求める－エラトステネスのふるい[1] p.141-2

素数を求めるのに「エラトステネスのふるい」という有名な方法がある。
これは 2, 3, 5,……と素数の目の「ふるい」で次々とふるい落として素数を求めるというやり方である。

その原理をイラストで示すと、次のようになる。[1] p.141

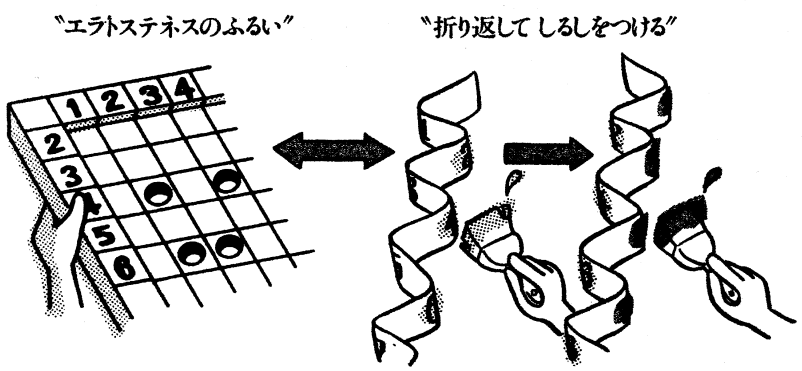


**図 9.1　エラトステネスのふるいの原理とAPLコーディングのアルゴリズム**

1) Adin D. Falkoff, "The Future of APL", APL Quote Quad 16, No. 2, 6 (1985).

この原理のポイントはＪでは、タテヨコの配列の形の変更がごく簡単に行えるという機能を利用している。

これは、最初に述べた「Jは計算ではなく、データの形を変える」という特徴を生かしたプログラミングであり、BASIC などでは考えもつかないし、まず不可能である。

ここでも、例えば 20 までの素数を求めるという具体的例題によって説明しよう。

```
   M =: >: i. 20
  P =: 1{M
  P
2
```

Ｐとして Ｍの１番目の要素、Ｊでは 0－オリジンなので、実際には Ｍの２番目の要素。

これを用いて、10 行 x 2 列の配列に変形する。Ｊでは動詞($)により、簡単に配列の形が変えられる。

```
  M0 =: 10 2 $ M
  M0
1  2
3  4
5  6
7  8
9 10
```

```
11 12
13 14
15 16
17 18
19 20
```

次に、この配列の最後の列だけを落として除く。これは動詞(}:"(1))を用いて次のようにしてできる。ちなみに動詞 (}:) は最後の1行 19 20 を落とす。

```
   M1 =: }:"(1) M0
   M1
 1
 3
 5
 7
 9
11
13
15
17
19
```

今度はこの配列の右に、動詞(,.)により 2 と 9 個の 0 をタテ方向に貼り付ける。

```
   M2 =: M1 ,. (2, 9#0)
   M2
 1 2
 3 0
 5 0
 7 0
 9 0
11 0
13 0
15 0
17 0
19 0
```

最後に動詞(,)により、この配列をほどいて1本のリストにする。

```
   M3 =: , M2
   M3
1 2 3 0 5 0 7 0 9 0 11 0 13 0 15 0 17 0 19 0
```

これまでの操作で何を行ったか、おわかりだろうか。

さきのイラスト「エラトステネスのふるい」の右図―1本のひもを折りたたんで、しるしをつける―に相当する。これは、すなわち「素数のふるい」を行ったこと、に他ならないのである。

　続いて、次の「素数のふるい」の操作を行う。
それには
　M =: M3
　P =: 2{M
 P
3
として、先の操作を繰り返す。

今度はヨコ、3 列の配列に変形する。
　M  =: M3
　M0 =: (7 3)$ M
　M0
1  2  3
0  5  0
7  0  9
0 11  0
13  0 15
0 17  0
19  0  1
　M1 =: }:"(1) M0
　M1
1  2
0  5
7  0
0 11
13  0
0 17
19  0
　M2 =: M1 ,. (3, 6#0)
　M2
1  2 3
0  5 0
7  0 0
0 11 0
13  0 0
0 17 0
19  0 0
　M3 =: , M2
　M3
1 2 3 0 5 0 7 0 0 0 11 0 13 0 0 0 17 0 19 0 0

続いて
```
  M =: M3
 P =: 3{M
 P
0
```
Pが0のときは配列は作れないので、このときはスキップする。これは4が合成数であることに対応する。
さらに
```
  M =: M3
 P =: 4{M
 P
5
```
として、続ける。ふつうは20の平方根(%: 20 => 4.47)から5まで行えば十分である。
以上の操作をまとめてプログラミングすればよい。プログラムを次に示す。

```
prime =: 3 : 0
N =. >: i. y.
M =. N
I =. 1
while. I < (>. %: y.)
 do.
  P =. I{M
  if. P = 0 do. goto_skip. end.
  Q =. >. (#N) % P
  M0 =. (Q, P)$ M, Q#0   NB. added 0's to final M
  MM =.  }:"(1) M0
  M1 =. MM ,. P, (<:Q)#0
  M =. , M1
  label_skip.
  I =. I + 1
 end.
}. M -. 0
)
```

このプログラムでは繰り返しはループ構造で行った。Jではループ構造はなるべく使わないようにと言われているが、必要に応じてどんどん使ったらよい、と私は思っている。ただ、BASICでよくやるような2重、3重のループは使わないようにする。
```
  prime 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

(ノート)
　2つのプログラム pri と prime との実行時間を比較してみた。
2次元の配列を用いた pri では 8000 まで可能、実行時間は t0(sec)であった。9000 では out of memory になった。
　pri 8000
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
　(途中省略)
7793 7817 7823 7829 7841 7853 7867 7873 7877 7879 7883 7901 7907 7919
7927 7933 7937 7949 7951 7963 7993
　t0
2.547
　pri 9000
|out of memory: pri
　ループによる方法では 20000 で OK、実行時間は t1(sec)であった。
　prime 20000
　t1
0.016

```
NB. reshape method 2013/1/4
NB. 西川利男「基礎からの APL」p.141-2

prim =: 3 : 0
0 prim y.
:
N =. >: i. y.
M =. N
I =. 1
while. I < (>. %: y.)
  do.
    P =. I{M
      if. x. = 1 do. wr 'P=', ":P end.
    if. P = 0 do. goto_skip. end.
NB.     Q =. <. (#N) % P
    Q =. >. (#N) % P
      if. x. = 1 do. wr 'P, Q:', ":P, Q end.
    M0 =. (Q, P)$ M, Q#0    NB. added 0's to final M
    MM =.  }:"(1) M0
NB.     wr P, (<:Q)#0
      if. x. = 1 do.  wr '----------' end.
    M1 =. MM ,"(1, 0) P, (<:Q)#0
      if. x. = 1 do.  wr M1 end.
    M =. , M1
      if. x. = 1
        do.
        wr 'go on? y/n'
        if. 'n' = rd 1 do. return. end.
      end.
    label_skip.
    I =. I + 1
  end.
}. M -. 0
)
```

```
NB. Prime Numbers
NB. J program coded from APL
NB. 西川利男「基礎からのAPL」サイエンスハウス(1991)
NB. p.101-104

  X =: >: i. 5
  X
1 2 3 4 5
  X | / X
0 0 0 0 0
1 0 1 0 1
1 2 0 1 2
1 2 3 0 1
1 2 3 4 0
  0 = X | / X
1 1 1 1 1
0 1 0 1 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
  +/ 0 = X | / X
1 2 2 3 2
  2 = +/ 0 = X | / X
0 1 1 0 1
  (2 = +/ 0 = X | / X) # X
2 3 5

  Y =: >: i. 50
  Y
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
  (2 = +/ 0 = Y | / Y) # Y
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
NB. array method
NB. 西川　「基礎からの APL」p.101-104
NB. e.g. pri >: i.100
pri =: 3 : '(2 = +/ 0 = y. | / y.) # y.'


NB. reshape method
============================================================
```

```
   prim 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
   load'f:¥j402¥user¥primes.ijs'
   prim 500
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499
   load'f:¥j402¥user¥primes.ijs'
   prim 600
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 1 2
   load'f:¥j402¥user¥primes.ijs'
   prim 600
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599
   prim 700
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
643 647 653 659 661 673 677 683 691
   prim 800
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
```

643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797

prim 900

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881
883 887

prim 1000

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881
883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997

prim 2000

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881
883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009
1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093
1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201
1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297
1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427
1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499
1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607
1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709
1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823

1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933
1949 1951 1973 1979 1987 1993 1997 1999
  prim 2100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881
883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009
1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093
1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201
1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297
1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427
1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499
1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607
1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709
1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823
1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933
1949 1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039
2053 2063 2069 2081 2083 2087 2089 2099
NB.
==========
==========