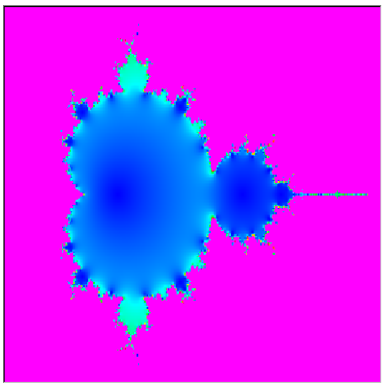
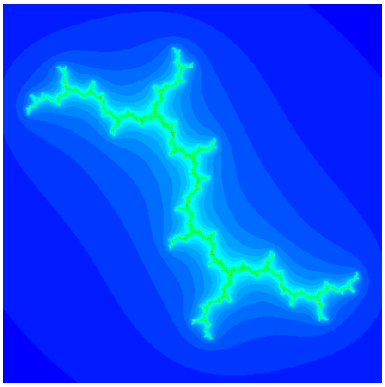
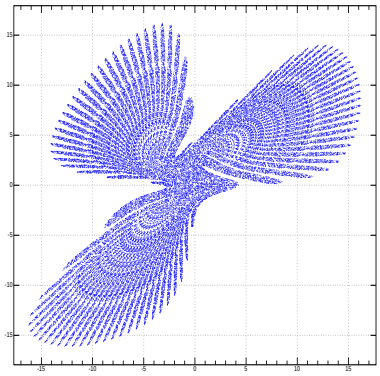
J Quick Reference

(Ver.2.0)







J Research Group

Japan APL Association

1998/2010

本稿は　TEX　で組版している。

- TEX　の組み版記号と J のキーボード記号の衝突が生じるので J のスクリプトの多くは TEX の verbatim 環境を用いて記述した。[*1]
- TEX　の欧文フォントは斜体を多く用いる。重要な箇所は verbatim を用いて立体にしたが、斜体も多く残っている。
- J の配列の区切りはスペースである。 明示的に TEX　のスペース記号　␣　で記述した箇所もあるが普通にキーボードのスペースキーで空白を入れればよい。
- 本文中のブルーで表記した箇所は参考である。

.

[*1] そのまま表示する

(K.E.Iverson 1920-2004 Canada)
(issue:www.vector.org.uk)

この論文の主題は単一の一貫性のある言語の中に、プログラミング言語の中に見いだされる実行可能性と一般性の利点が数学的表記法が持つ利点と効果的に組合わせ得ると言うことです

K.I.Iverson　思考の道具としての表記法
(ACM チューリング賞講演集 共立出版より*)*

初版の序ー平成時代の新言語

人と人とが意志を伝えあうためには、まず言葉が必要である。コンピュータに仕事をさせる場合でも、何らかの方法でコンピュータとのコミュニケーションを図る必要がある。コンピュータが理解できる言葉は、端的にいえば、０と１の羅列からなる「機械語」である。この機械語を人間がみると、まるで暗号のようなもので、これではコンピュータと会話することはまず不可能である。そこで、人間が理解しやすい言語がいろいろと考案されてきた。最もポピュラーな言語としては *BASIC* があるが、昨今ではより高水準の *C* や *PASCAL* といった言語のほうが多用されつつある。特に科学技術計算向きの言語として、４０年ほど前に開発されたのが *APL* という言語であるが、当時としては大型計算機でないと利用することができないことや、見馴れない特殊記号を用いていたために、広く普及するには至らなかった。

ところが１０年ほど前に、*APL* の創始者であるアイバーソン (*K.E.Iverson*) を中心として、*APL* の泣きどころであった特殊記号を排し、通常のアスキー記号にピリオド「.」やコロン「:」を付加することによって、記号の種類を３倍に拡大し、*APL* の機能をさらに強力にした「関数型言語」を開発し、これを言語 *J* と名づけた。コンピュータの飛躍的な進歩は、パソコンレベルでも *APL* や *J* が使える時代になったのである。ともあれ *J* 言語は、パソコンを「超」高級電卓として利用できることが大きなメリットといえる。つまり、コンピュータに不慣れなタイプでも、すぐに自分でプログラムを作ることができる。そこで、この *J* 言語なるものの簡単な紹介を試みてみたい。とにかく、「馬には乗ってみよ、人には添うてみよ」というように、是非、チャレンジしてみて頂きたい。

本稿は、*J* 言語の記号と基本概念を、機能・用途別に分類し、簡単な説明を加えた「クイック・リファレンス」である。Ｊ言語の特徴と概要を知る上で便利なものであり、また *APL* を知らない読者でも初めて *J* 言語を学ぶときにも役立つものと思う。

Ｊ言語のキーワードの日本語表現については、Ｊ言語研究会で議論・統合したものを用いた。対応する *APL* の説明や用例は、「日本語ＡＰＬクイック・リファレンス *(*日本ＩＢＭ社*)*」と対比し、参照した。

第 2 版にあったて

この冊子の改訂中に *NHK* ラヂオ *Buisiness English* のフレーズが聞こえた。「*Algebra* は数学の時間ではなく情報の時間で教えた方が良いと言うことが論議されている。」

超高級関数電卓 *J* は格好のツールであり、*Quick Reference* は時宜を得た教材、自習 *Text* になる。間もなく間違いなく。

J はドイツ製で世界標準の会計経理の *Graphics* や最もポピュラーな *CPU* の計算 *Check* など先端分野でも活躍しており、輪は広範に世界へ広がっている。最近は専門書でも数式とツールを使った計算結果のみでアルゴリズムを丁寧に解説することが少なくなってきている。更にはプログラム特許では理論が完全にブラックボックス化されれしまう。理論のプロセスを確認し、ブラックボックス化されないためにも簡単に高度なプログラムが作れる言語が必要とされる。

この *Reference* が *J* 言語を始められる人の助けとなれば幸甚である。

第0章

目次

第0章

J 言語超入門

J 言語超入門

1. **インストール**
   第 *4* 章　*4.1* で解説
2. **実行方法**
   - 対話型（インタプリッタ）言語であり、入力するとそのまま、直ちに結果が得られる。*C* や *FORTRAN* のようなコンパイラ言語ではない。複雑な処理や大きな一連の処理はプログラムを組んで行う。
   - *J* の内部は *C* と *J* のマクロで作成されており、コンパイラ言語の *C* に速度で大きく遅れることはない
   - 実行は *ijx* の画面で行う。超高級関数電卓として用いるときも *ijx* の画面を用いる。
   - スクリプト（プログラム）を書くときは *ijs* の画面を用いる。*File→ newijs*
   - *ijx* で予め動かして確認できたスクリプトを *ijs* に *copy* すると便利。殆どデバッグは不要。
   - *Ctrl + W* で *ijs* のスクリプトは *ijx* にロードされる。
   - *Ctrl + Shift + ↓↑ (*履歴の呼び出し*)*
   - *Ctrl + D (*過去の実行関数の呼び出し*)*
3. **数学用語との比較** *J* の名詞の形の呼称と数学用語との対応を以下に示す。*J* は数値計算以外にも事務計算や文字列の処理などにも利用できる汎用言語であるから、必ずしも数学用語とは一致しない。

J 言語	数学用語	*J* での表示例
アトム	スカラー	2
リスト	ベクトル	2 4 6
テーブル	マトリクス (行列)	0 1 2 3 4 5 6 7 8
レポート	多次元配列	0 1 2 3 4 5 6 7 8 9 10 11

4. **品詞**
   J 言語では自然言語に習ってデータ · 処理手順は次のような品詞で呼ばれる。（他のプログラミング言語の呼称より洗練されている）
   データ （数値、文字列) ··· 名詞
   処理 ··· 動詞、副詞、接続詞

J の祖である *APL* に対しては次のように対応する。*(*「代名詞」や「代動詞」という概念もあるが、特に名詞、動詞と区別しなくともよい*)*

（*J* 言語）	（*APL*）	(他の言語)
名詞	定数	変数
動詞	関数	関数
副詞	作用素	
接続詞	—	

J のシステムで予め定められた処理はキーボード記号 *(ASCII* 文字）を用い、これをプリミティブと呼ぶ。（詳細は後述）

5. **数値**（名詞）
   - 名詞に型の指定は必要ない。名詞の種類には数値、文字、ボックスがある。
   - 数値はそのまま入力した通りで整数、小数の区別の必要はない。
   - 数値は全て明記する必要がある。「*.5*」のように「*0*」を省略することはできず「*0.5*」のように入力しなければならない。
   - 負の数にはアンダーバーをつけて**「_5」**のように表わす。*APL* の場合のオーバーバーに対応するもので符号の「_」と差を計算するマイナス「-」とは異なる点に注意。*(5-_3* と記述でき数学の弱点を改善している*)*
   - *J* は次のような表記法を取り入れており、浮動小数点や指定精度に制約されないで計算できる。

分数表記	*3r12*	$\left(\frac{3}{12}\right)$
無理数の表記	*1x1*	e
パイ	*1p1*	π
整数拡張表記	*123x*	浮動小数点を用いない

   - 数値に名前を付けた定義には　**=:や=.** を用い次のように行う
     （*APL* は ←,　*FORTRAN,BASIC* では*=)*

```
A =: 1 3 5
even =.2 4 6
```

   - 複数のデータをまとめて用いる配列（リストやテーブル）の場合は、数値の間にスペース「␣」を入れなければならない。
     *[*2*]*

   - 配列の大きさや形は$の左引数で指定する。

```
2 3 $ 5 2 3 4 1 6
5 2 3
4 1 6
```

   - *0* から始まる整数列は「*i.*」を用いて生成する。

```
   i.10
0 1 2 3 4 5 6 7 8 9

i. 2 3  NB. 配列
```

*[*2*]* ␣はキーボードの半角スペース

```
0 1 2
3 4 5
```

6. **文字列**（名詞）

   文字列は前後をシングル･クオート**「'」**で囲んで定義すれば良く、他の言語のような型の指定は一切不要である。

   ```
   JAPLA ='JAPAN APL ASSOCIAION'
   ```

7. **ボックス**

   複数の数値や文字を四角で囲んだものをボックス *Box* として定義できる。*(*数値や文字列をデータとして用いる。関数をボックスに入れて並列処理を行うこともできる）

   ボックスと単独の数値とは直ちに演算はできない。（ボックスの中やボックスと単独の数値を演算するため *L:0* などが用意されている）

```
   1 3 5;'JAPLA'
+-----+-----+
|1 3 5|JAPLA|
+-----+-----+
```

```
   1 2 3;4 5 6
+-----+-----+
|1 2 3|4 5 6|
+-----+-----+
  +/(L:0) 1 2 3;4 5 6
+-+--+
|6|15|
+-+--+
```

8. **プリミティブ** *(*動詞、副詞、接続詞*)*

   - *J* 言語で用いられる、**「+」「-:」**等の記号をプリミティブといい、動詞、副詞、接続詞などがある。数学＝ギリシャ文字を使うという伝統を *K.E.Iverson* が *APL* で継承し、*J* では記号をキーボード記号に移した。
   - プリミティブの用法は固定されているが *plus=:+* や *square=:*:* のように任意の名前をつけて使ってもよい。
     このとき、**plus や square** は動詞と呼ばれる。
     *(*厳密には名詞、動詞にテンポラリーな名前を付けたものを代名詞、代動詞と呼ぶ。*)*
   - *tasu=:+* など各国の言語で自由に定義しても良い。
   - *unicode* もサポートされており日本語などの多バイト文字はデータ、表示、コメントに使用できるが、名詞や動詞の定義（名前）や処理手順の記述に用いることは出来ない。
     [*3]
   - プリミティブにはキーボード記号そのまま**（ベア）**、後ろに**ピリオド** *(.)* をつけたもの、さらに**コロン (:)** を付したものと 3 種類あり、機能はそれぞれ異なる。 更に単項と両項（次の項）があり、*1* つの記号に概ね *6* の機能が割り付けられている。

     ```
     3+3             NB. plus(dyad)
     ```

[*3] Shift-jis は用いることが出来ない

```
6
   12 +. 48       NB. GCD(dyad)
12
   0 0 1 +. 0 1 1 NB. or(dyad)
0 1 1
   +: 5           NB. double(monad)
10
```

- （最近は{:: fetch や p.. Polynomial Derivative の様に *2* 個付したものも少しではあるが見受けられる。）

9. **単項と両項**（動詞の定義）

ほとんどの動詞には、*(*右引数だけの*)* **単項**と、*(*左右に引数をとる*)* **両項**の 2 種類がある。この点は *C* や *FORTRAN* と大きく異なる。

キーボードに制約があるため、一つの記号を単項と両項とで用いる場合が殆どあり、その機能は異なるので使い分けに注意が必要である。

```
   ^. 100
4.60517 （自然対数 e 単項）
   10 ^. 100
2     （常用対数　両項）
```

10. **副詞**

副詞は *(*右から*)* 動詞を修飾して新たな動詞を作る。最も縦横に活躍する副詞はスラッシュ「/」である。

```
   +/ 1 2 3 4 5
15        （「1+2+3+4+5 = 15」 動詞「+」を各要素の間に入れて演算する）
   */ 1 2 3 4 5
120           （「1*2*3*4*5 = 120」動詞「*」を挿入）
```

11. **スペース**

- プリミティブと数値の間には必ずしもスペースをとる必要はない。（あった方が見やすい。）

```
   3*3
   3 * 3
9
```

- **定義した動詞と数値の間**には必ずスペースが必要である。
- ドットやコロンが独立した記号として使われる場合には、前のプリミティブとの間にスペース「␣」が必要となる。

  明示的定義を宣言する際の「 name=: 3 : 0」では、コロンの前のスペースは絶対必要。さらに 内積の「+/ . *」の場合もドットの前のスペースは必ず挿入しなければならない。

12. 実行の優先順位

J の実行順序は *(APL* と同様に）原則は右から左である。

```
  5%4*3+2-1
0.3125      (5%(4*(3+(2-1))) = 0.3125 のように演算する)
```

動詞などの全てのプリミティブの**優先順位は平等**であり、**右から左へ**と記述順に実行される。
*(*数学の優先順位は無視される。*K.E.Iverson* 言語 *APL* や *J* は数学を拡張しており、また多くのユーザー定義関数も用いられるので、優先順位を決めると実行が極めて煩瑣になる）

13. **明示型とタシット型** *(*動詞の定義法）
J の動詞の定義には明示型 *(Explicit* 型）と関数型（*Tacit* 型）がある。（*J* は当初から *Tacit* 型をサポートし、後に *APL* が追従した）
明示型定義は、右から順に記述順に演算する。
`reg=: 3 :  'x %. 1,.y'` *NB.* 線形回帰
タシット型の関数の定義では、フォーク *Fork* やフック *Hook* を作るので、その実行順序はやや複雑になる。

```
   5(+%-)3      NB. 5 と 3 の平均    Folk
4             (「(5+3)%(5-3) = 4」のように演算する)
```

`reg=: [ %. 1&,.@]` *NB.* 線形回帰

関数型　タシット定義

関数型定義の特色

（第 *6* 章参照）
J が当初から取り入れたエレガントな関数の定義方法で、その中心は次の通りである。

- *Folk* と *Hook*
- *bond* と *Atop*
- 簡易反復法

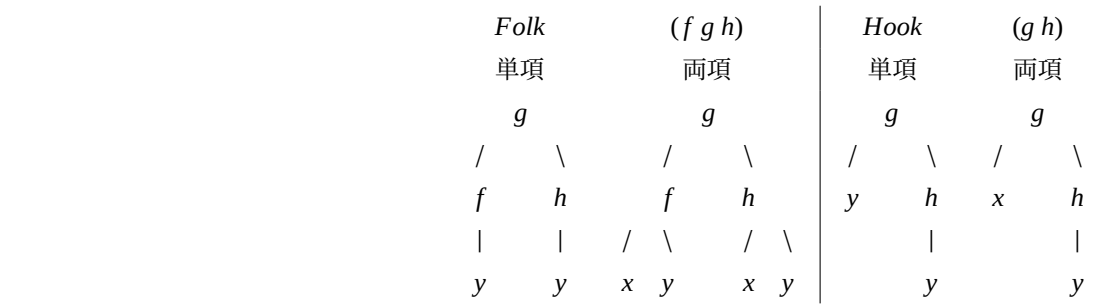
◇ 関数型定義では引数は記述しなくとも良い。
◇ 両項の場合は [ ] で左右の引数を区別する
◇ 実行順序は *Folk* 等を用いるため右から左とは限らない

```
DAT=: 20 ?. 100
mean=: +/%#             NB. mean
dev=: - mean            NB. hook  (x - x bar)
var=: # %~ +/@:*:@dev   NB. variance
sd=: %:&var             NB. Standard deviation
```

フォークとフック

タシット型の動詞を連結する場合には、右から連続した *3* つの動詞 *(*フォーク*)* が最優先される。*2* つの動詞の場合はフックになる。*(3* 個以上ある場合は右から *3* 個づつとり、最後が *3* 個なら *Folk,2* 個なら *Hook* になる。)

```
    (+/%#)a=: 2 3 4 5
3           (「(+/ a)%(#a) = 3」のように演算する)
    5(+%-)3
4           (「(5+3)%(5-3) = 4」のように演算する)
    (*-)3
_9                (「3*(-3) = _9」のように演算する)
```



*(*フォークやフックにおける ***g*** は必ず両項動詞でなければならない*)*

キャップド フォーク

```
  Capped Fork [: g h
```

Tacit 型の動詞で *Folk* や *Hook* を構成しないように**右から左へ**に実行したいときに用いる。

*(*キャップドフォークの場合 ***g*** は必ず単項動詞でなければならない*)*

接続詞 ボンドとアトップ

J では、動詞と動詞、あるいは動詞と名詞を連結して新たな動詞を作ることができ、その連結の為に接続詞を導入した。その代表が、次のボンド「&」とアトップ「@」である。

Bond	「&」	*Atop*	「@」
単項	両項	単項	両項
u	*u*	*u*	*u*
|	/ \	|	|
v	*v* *v*	*v*	*v*
|	| |	|	/ \
y	*x* *y*	*y*	*x* *y*

単項の場合に限り**「&」と「@」**は同じ機能で、**「u{v(y)}」**と演算する

両項の場合は機能が異なる

「x (u&v) y」＝「v(x)u v(y)」
「x (u@v) y」＝「u(x v y)」

動詞と名詞（数値など）との結合には**「m&u」、「u&n」**のように&を用いる。

```
   DAT=: 20 ?. 100
   mean=: +/%#             NB. mean
   dev=: - mean            NB. hook  (x - x bar)
   var=: # %~ +/@:*:@dev   NB. variance
   sd=: %:&var             NB. Standard deviation
   sd DAT
27.7767
   var DAT
771.547

   ([: %: var) DAT
27.7767
   (sd=: %:@var) DAT
27.7767
```

定義ファイルのインクルード他

インクルードファイルの指定

J ではインクルードとしてファイルの指定は特に必要としない。
特に必要な都度プログラムを記述したファイルの最初に記述すれば読み込まれる。

- よく使うファイルのロード法
  ijx に直接書いてもよい。*ijs* の冒頭に記述しておくとその都度読み込まれる
  `require 'plot numeric trig'`*(*一度だけ）
  `load 'plot numeric trig'`（ファイルロード時毎回）

[*4]

- 各自の定義ファイル（例えば *my_util.ijs*）を作成しておくと便利である。

```
mean=: +/%#
mav=: +/\%[
reg=: %. 1,.&@]
wr=: 1!:2&2
```

```
require jpath '~classes/util/my_util.ijs'
```

WINDOWS の機能 *(Files→Open* または *Run　→　Files*）で読み込み、*ctrl* + *W* でロードする

- *profile.ijs* は指定しなくとも起動時に最初に読み込まれる。
  更に *profile.ijs* に記述しておくと常に読み込まれる。

ロケール

J では名詞（変数）や動詞（関数）の名前が重複したときは**警告なしに**後からロードしたり、定義した方が優先する。

このような名前の衝突を避けるにはロケールの機能を用いると安全である。

次のように *my_util.ijs* に *nakano* をロケールとして登録する

```
coclass 'nakano'
```

*reg_nakano_*
*mean_nakano_*
は同じ名の別の関数として定義でき、ロケールにある名詞や関数名と衝突しない。[*5]
[*6]

ロケール機能を用いると *J* でオブジェクト指向プログラミングを実現できる。
また、ファイルに
coclass 'koenji' と指定されている場合は
*reg_koenji_*とロケール機能により同じ関数を用いることができる

[*4] numeric に入っている関数 *clean* は数値計算の塵取りに便利である
[*5] この状態で reg mean を単独で使うことは出来ない
[*6] xxx_main.ijs や xxx_main.ijp として別個にファイルを作ると分かりやすい

第1章

J のプリミティブ

F1 又は *Help* → *Vocaburary* で *J* 自身の *primitive* の解説を見ることができる。

1.1 数値計算の概要

1.1.1 基本的な数学演算

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
+ *Conjugate* 共役	+	実数の範囲内ではそのまま `   + 0.4 _5 0` `0.4 _5 0` `   + 3j4` `3j_4`**（共役複素数）**	*Plus* 加算 *X+Y*	左引数と右引数との和 左右のサイズは同じでなければならない。片方がスカラの時はスカラは拡張される。 `   5 + 1 2 3` `6 7 8` `   1 2 3 + 4 5 6` `5 7 9`
- *Negate* 逆符号	-	右引数の符号を反対にする `   - _5 0 1` `5 0 _1` `   - 3j4` `_3j_4` **複素数では実部と虚部の符号をそれぞれ逆にする**	*Minus* 減算 *X-Y*	**左引数から右引数を引く** `   8 _3 - 6 _5` `2 2` `   3.5j0.5 - 1j1` `2.5j_0.5` **複素数どうしの引算も可能**

*
Signum
符号
** Y* | × | 実数の場合、正には *1* を、*0* には *0* を、負には *_1* をそれぞれ付与。
複素数では大きさと偏角を与える

```
   * _5 0 4
_1 0 1
   * 3j4
0.6j0.8
   *. 3j4
5 0.927295
   *. 0.6j0.8
1 0.927295
```

**.(Length/Angle)* では大きさが *1* で、同じ偏角の複素数 | *Times*
乗算
*X * Y* | **左引数と右引数との積**

```
   0 1 * 3 4
0 4
   3.5j0.5 * 1j1
3j4
```

複素数どうしの掛算も可能 |
| *%*
Reciprocal
逆数
% Y | ÷ | 逆数、すなわち `% y` は `1 % y` と同じ。
複素数の逆数は大きさが逆数で、偏角が反対符号になる

```
  %0.25
4
  % 3j4
0.12j_0.16
  3j4 * % 3j4
1
   *. 3j4
5 0.927295
   *. % 3j4
0.2 _0.927295
```

| `Devide`
除算
`X % Y` | 左引数を右引数で割る

```
   0 8 % 1 0.4
0 20
   3j4 % 1j1
3.5j0.5
   1j1 * 3j4 % 1j1
3j4
```

複素数どうしの割算も可能 |

| *Magnitude* 絶対値 *|Y*	|	**符号を取り去って絶対値にする** `   | 6 _5` `6 5` `   |3j4` `5` **複素数に対してはノルム** *xjy* のとき $\sqrt{x^2+y^2}$	*Residue* 剰余 *X | Y*	右引数を左引数で割ったときの余り `   3|  i.7` `0 1 2 0 1 2 0` `   3 |  -i.7` `0 2 1 0 2 1 0` `   _3 |  i.7` `0 _2 _1 0 _2 _1 0` `   _3  | -i.7` `0 _1 _2 0 _1 _2 0`
`*:` *Square* 平方 `*: Y`		右引数の要素を平方する `   *: i.5` `0 1 4 9 16` `  ^&2 i.5` `0 1 4 9 16`	*(Not-And)* *(否定論理積)* *X *: Y*	左右の引数が共に *1* のときに *0*、そうでなければ *1* `   0 0 1 1 *: 0 1 0 1` `1 1 1 0`
`%:` *Square -root* 平方根 `%: Y`		右引数の要素の平方根をとる `   %: 4 9 16` `2 3 4` `   %: _4` `0j2` **負の値のときは虚根になる**	*Root* ベキ乗根 `X %: Y`	*Y* の *X* 乗根で、`Y ^ % X` に等しい `   2 3 4 %: 4 27 256` `2 3 4` $^2\sqrt{4}, ^3\sqrt{27}, ^4\sqrt{256}$ `   _1 %: 4` `0.25` `-1` **乗根は逆数に等しい**
`^` *Exponen -tial* *e* のベキ乗 `^ Y`	*	e の *Y* 乗 `   ^ 0 1 2` `1 2.71828 7.38906` e^0, e^1, e^2	*Power* ベキ乗 `X ^ Y`	*X* を *Y* 乗するに等しい `   2 3 4 ^ 1 2 3` `2 9 64` $2^1, 3^2, 4^3$

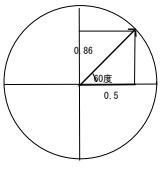
^ *Natural-Log* 自然対数 ^. Y	⊗	底が e の *Y* の対数 *(自然対数)* ^. 1 2 1 0.693147 $log_e 1, log_e 2$	*Log* 対数 X ^. Y	*X* を底とする *Y* の対数に等しい 10 ^. 125 100 2.09691 2 $log_{10} 125,\ log_{10} 100$
>: *Increment* *1* 増 >: Y	≥	**Y に 1 を加える** >: i.5 1 2 3 4 5 (i.5)+1 1 2 3 4 5	*(Larger or equal)* *(大きいか等しい)* X >: Y	左引数が右引数より大きいか等しければ *1*、そうでなければ *0* 20 30 40 >: 40 30 20 0 1 1
<: *Decrement* *1* 減 <: Y	≤	**Y から 1 を減じる** <: i.5　NB. y-1 _1 0 1 2 3 -. i.5　NB. 1-y 1 0 _1 _2 _3 *1* から *Y* を引く。*Complement*	*(Lesser or equal)* *(小さいか等しい)* X <: Y	左引数が右引数より小さい か等しければ *1*、そうでなければ *0* 20 30 40 <: 40 30 20 1 1 0
>. *Ceiling* 天井値 >. Y	⌈	*Y* より大きな最小の整数 >. 0.4 _0.3 _3.4 1 0 _3　　**(切上げ)** >./ 1 2 3 4 5 5　**(最大値の取り出し)**	*Larger of* 最大値 X >. Y	左引数と右引数の大きい方の値を選ぶ 4 0.5 >. 1 6 4 6
<. *Floor* 床値 <. Y	⌊	*Y* を超えない最大の整数 <. 0.4 _0.3 _3.4 0 _1 _4　　**(切捨て)** <./ 1 2 3 4 5 1　**(最小値の取り出し)**	*Lesser of* 最小値 X <. Y	左引数と右引数の小さい方の値を選ぶ 4 0.5 <. 1 6 1 0.5

+. *Real/ Imagi-nary* 実部／虚部 +. Y		複素数の実部と虚部を生成 +. 1j2, 2j3, 3j4 1 2 2 3 3 4	*GCD* 最大公約数 最大公約数 *(Or)* *(論理和)* *X* +. *Y* *1.2* 参照	GCD 24 +. 60 12 12 24 36 +. 60 72 120 12 24 12 Or 引数が論理数の時。 左右の引数が共に 0 のときに 0、 そうでなければ 1 0 0 1 1 +. 0 1 0 1 0 1 1 1
*. *Length /angle* 長 さ/偏角 *. *Y*		複素数の長さと偏角を与える *. 3 3 0 *. 3j4 5 0.927295	*LCM* 最小公倍数 最小公倍数 *(And)* *(論理積)* *X* *. *Y* *1.2* 参照	LCM 24 *. 60 120 And 引数が論理数の時 左右の引数が共に 1 のときに 1、 そうでなければ 0 0 0 1 1 *. 0 1 0 1 0 0 0 1

短縮 Shorthands

記号	機能	説明	用法	例	結果
+:	*Double*	*2* 倍	+: Y	+: i.5	0 2 4 6 8
-:	*Halve*	*2* 分の *1*	-: Y	-: 0 2 4	0 1 2
-.	*Complement*	$1-y$	-. Y	-. 2 5 6	_1 _4 _5
*:	*Square*	*2* 乗 y^2	*: Y	*: 2 4 6	4 16 36
%:	*Squre Root*	平方根 $\sqrt{y}$	%: Y	%: 4 16 36	2 4 6

1.1.2　数の表記法

用法	結果と例	用法	結果と例
_ *(under-bar)* *Negative sign* マイナス符号	数学のマイナス記号 *J*　アンダーバー *APL*　オーバーバー `   -3.14` `_3.14` `   3-_1 NB.` **括弧不要** `4`	`_9:↔9:` 数 動詞	`_9:⋯ _1:  0: 1: ⋯ 2: 9:` `_9` **から** 9 までの数を与える動詞
`ad` `ar` 角度 （名詞）	複素数の角度 *degree* による表示 複素数の角度 *radian* による表示 `   1ad60 ; *. 1ad60` `+------------+--------+` `|0.5j0.866025|1 1.0472|` `+------------+--------+` `   1ar1;*. 1ar1` `+-----------------+---+` `|0.540302j0.841471|1 1|` `+-----------------+---+` $60^o = \frac{\pi}{60} = 1.0472$ ラジアン `   1p1%3` `1.0472` $\frac{180}{\pi} = 1$ ラジアン $\simeq 57.3$ 度 1*ar*1 $\Longleftrightarrow$ 1*ad*57.2958 `   180%1p1` `57.2958`  	*b* *n* 進数 *(*名詞*)*	左に基底を右にその基底の進数での値を置いた構成に対する *10* 進数の値 `  16b23   NB.16` **進数表示** `35  (「(2*16)+3」の値)` `   16b1j` `35   (「(1*16)+19」の値` `   2b111  NB.2` **進数表示の** `10` **進数での値** `7` `   #: 7` *nbm* は *m* がアルファベット *(a-z)* のときは *n* に関係なく *10-35* を表す `   1bj ; 2bj ; 3bx` `+--+--+--+` `|19|19|33|` `+--+--+--+`

e 浮動小数点 （名詞）	**１０進数の科学表記による表示** `1.2e14` `1.2e14` `1 x: 1.2e14` `120000000000000` `2.3e_2` `0.023` `0j16 ": 2.3e_10` `0.0000000002300000`	j 複素数 （名詞）	**複素数の実部、虚部による表示。** `3j4` **は実部** `3` **虚部** `4` **の複素数**
p π 円周率 （名詞）	*mpn* は $m\pi^n$ `1p1` `3.14159` `0j20 ": 1p1` `3.14159265358979310000` `2p1` `6.28319` `1p2` `9.8696` $2p1 \leftrightarrow 2\pi$ $1p2 \leftrightarrow \pi^2$	*r* *rational* 有理数 （名詞）	有理数の分数入力と表示。 `2r3` `2r3` $\frac{2}{3}$ `_1 x: 2r3` `0.666667` `0j24 ": 2r3` `0.666666666666666666666667` 倍精度を超える精度が用いられている
x *exponents* 指数表示 （名詞）	`m x n` **は** `(m e)` **の** `n` **乗** `1x1` `2.71828 NB. Euler` **の定数** `e` `0j20 ": 1x1` `2.718281828459045100000` *1x1* は *e* を表す `2x3` `40.1711` *2x3* は $2e^3$ を表す	x*(末尾)* *Extended-Preci-sion* 拡張精度 （名詞）	長い整数を指定する。 `]a=.123456789x` `123456789x` `*: a` `15241578750190521` `3 % a` `1r41152263`

x: *Extended precision* 拡張精度の表示 （動詞）	**拡張表現** 1 x: 1.2 NB. **少数の分数表示** 6r5 _1 x: 1r3 NB. **分数の小数表示** 0.333333 2 x: 1r3 NB. **有理数を分子分母で表示** 1 3 1 x: o.1 NB. **パイの分数表現** 1285290289249r409120605684		

定数 **(Constants)** 数の表記の実行順の規則

記号を用いた数の表記の優先順位（上から順に）同じ順序のものは同時には使えない。

.	*decimal points*	小数点
_	*negative sign*	マイナス
e	*Exponential*	指数
ad ar j	*Complex*(*in degree or angle*)	複素数
p x	*π Euler′s number*	円周率 オイラー数の表記
b	*BaseValue*	*n* 進数表記

```
   2.3e2 2.3e_2 2j3
230 0.023 2j3
   2p1 1p_1
6.28319 0.31831
   1x2 2x1 1x_1 NB. %1x1
7.38906 5.43656 0.367879
  2e2j_2e2 2e2j2p1 2ad45
200j_200 628.319j6.28319 1.41421j1.41421
 16b1f 10b23 _10b23 1e2b23 2b111.111
31 23 _17 203 7.875
```

数のいろいろな形とその演算結果の形

	B	*I*	*X*	*R*	*D*	*Z*
B	*B*	*I*	*X*	*R*	*D*	*Z*
I	*I*	*I*	*X*	*R*	*D*	*Z*
X	*X*	*X*	*X*	*R*	*D*	*Z*
R	*R*	*R*	*R*	*R*	*D*	*Z*
D	*D*	*D*	*D*	*D*	*D*	*Z*
Z	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*

B	*Boolean*	2 進数
I	*Integer*	整数
X	*eXtended Integer*	拡張整数
R	*Rational*	分数
D	*Floating point*	浮動小数
Z	*Complex*	複素数

1.1.3　無限大、超準数

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
_ *(under bar) Infinity* 無限大		単独では無限大を表わす名詞 `   2%0` `_`		
_. *Indeter-minate* 不定		無限大の演算で生じる不定形を表わす名詞 `   _. -: _.` `_.` `   (3+_ ),3 + _.` `_ _.`		
_: *Infinity* 無限大		無限大を生じさせる動詞 `   _: 1 2 3` `_`	*Infinity* 無限大 `X _: Y`	右引数の指定したランクに対応した形の無限大を派生 `   _:("0) i. 2 3` `_ _ _` `_ _ _` `   (_:"1)i.2 3` `_ _` `   3.14(_:"1)i.2 3` `_ _` **左引数には関係ない**
__ *Negative-Infinity*		マイナス無限大 (−∞) `   __*3` `__`		

1.1.4 円関数 · 三角関数

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
`o.` *Pi-times* 円周倍率 `o.Y`	∘	π と Y との積 $Y\pi$ `   (o.1),o.2` `3.14159 6.28319` `   1p1 2p1` `3.14159 6.28319` $\pi, 2\pi$	*Circular* *Function* 円関数 `X o.Y`	左引数で円関数の型を指定 `  rfd=:(%&180)@o.` `  rfd 90 180` `1.5708 3.14159` **（角度をラジアンに変換）** `      1 o. 1.5708` `1` **（SIN 90°の値）** `   2 o. 1.5708` `_3.67321e_6` `   2 o. rfd 90` `6.12303e_17` **（COS 90°の値）**

.

1 *o. y*	*sin y*	_1 *o. y*	*arcsin y*
2 *o. y*	*cos y*	_2 *o. y*	*arccos y*
3 *o. y*	*tan y*	_3 *o. y*	*arctan y*
5 *o. y*	*sinh y*	_5 *o. y*	*arcsinh y*
6 *o. y*	*cosh y*	_6 *o. y*	*arccosh y*
7 *o. y*	*tanh y*	_7 *o. y*	*arctanh y*

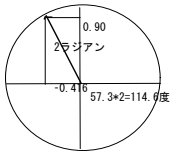
0 *o. y*	$\sqrt{1-y^2}$		_4 *o. y*	$\sqrt{y^2-1}$	
4 *o. y*	$\sqrt{y^2+1}$		_8 *o. y*	$-\sqrt{-1+y^2}$	
8 *o. y*	$\sqrt{-1+y^2}$		_9 *o. y*	y	
9 *o. y*	*RealPart*(y)	y の実数部	_10 *o. y*	*Conj*(y)	共役
10 *o. y*	*Magnitude*(y)	y の絶対値（ノルム）	_11 *o. y*	*j.y*	複素平面上で 90^o回転
11 *o. y*	*ImaginaryPart*(y)	y の虚数部	_12 *o. y*	$\wedge$*j.y*	$x+iy \to e^{-y+ix}$
12 *o. y*	*Angle*(y)	y の偏角（ラジアン）			

`require 'trig'` を用いれば *sin*, *cos* などの数学の記法が利用できる

1.1.5　複素数の演算

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
+ *Conjugate* 共役	+	**共役複素数を与える** `   + 3j4` `3j_4`	*Plus* 加算 *X + Y*	**複素数どうしの足し算** `    3i4 + 3j_4` `6`
- *Negate* 逆符号	-	**実部と虚部の符号を逆にする** `    - 3j4` `_3j_4`	*Minus* 減算 *X - Y*	**複素数どうしの引算** `    3.5j0.5 - 1j1` `2.5j_0.5`
* *Signum* 符号 * *Y*	×	**大きさが 1 で、同じ偏角の複素数を与える** `    * 3j4` `0.6j0.8`	*Times* 乗算 *X * Y*	**複素数どうしの掛け算** `    3j4 * 3j_4` `25`
% *Reciprocal* 逆数 *% Y*	÷	`% y` **は** `1 % y` と同じ。 複素数の逆数は大きさが逆数で、偏角が反対符号になる `    % 3j4` `0.12j_0.16` `    3j4 * % 3j4` `1` `    *. 3j4` `5 0.927295` `    *. % 3j4` `0.2 _0.927295`	*Devide by* 除算 *X % Y*	**複素数どうしの割算** `    3j4 % 1j1` `3.5j0.5` `    1j1 * 3j4 % 1j1` `3j4`

Magnitude 絶対値 *|Y*	|	**符号をとって絶対値にする** `   |6 _5` `6 5` `   | 3j4` `5` **複素数に対してはノルム** *xjy* のとき $\sqrt{x^2+y^2}$	*Residue* 剰余 *X |Y*	**右引数を左引数で割ったときの余り** `   3 | i.7` `0 1 2 0 1 2 0` `   3  | -i.7` `0 2 1 0 2 1 0` `   _3 |  i.7` `0 _2 _1 0 _2 _1 0` `   _3 |  -i.7` `0 _1 _2 0 _1 _2 0`
+. *Real/ Imagi-nary* 実部／虚部 `+. Y`		複素数の実部と虚部のリストを生成 `  +. 1j2, 2j3, 3j4` `1 2` `2 3` `3 4`	*GCD* 最大公約数 *Or* 論理和 *X +. Y*	左右の引数が共に *0* のときに *0*、そうでなければ *1* `   24 +. 60` `12` `   0 0 1 1 +. 0 1 0 1` `0 1 1 1`
*. *Length /angle* 長　さ/偏角 **. Y*		長さと偏角を与える (r,θ) $y = re^{i\theta}$ `  *. 3` `3 0` `   *. 3j4` `5 0.927295` `5 *1x1^ 0j0.927295` `3j4`	*LCM* 最小公倍数 *And* 論理積 *X *. Y*	左右の引数が共に *1* のときに *1*、そうでなければ *0* `   24 *. 60` `120` `   0 0 1 1 *. 0 1 0 1` `0 0 0 1`

j. *Imaginary* 虚数生成 j. Y		*j.y* は *0j1*y*]b=.j. a=.1j1 _1j1 (b-a),(b=.*.b),:a=.*.a 0 1.5708 1.41421 2.35619 1.41421 0.785398 rfd 90 1.57 j.y **は** y を複素平面上で 90^o 回転する	*Complex* 複素数生成 *X j. Y*	実数から複素数を生成する *x j. y* は *x* + *j. y* 3 j. 4 3j4 3 + j. 4 3j4
r. *Angle* 単位複素数の角度 r. Y		単位複素数の角度 r. y **は** ^ 0j * y $e^{i\theta}$ r. 0 1 1 0.540302j0.841471 ^0j1*0 1 1 0.540302j0.841471 *. r. i.3 1 0 1 1 1 2 **実数は単位円周上に移される** *. r. 1j1 1j0 1j_1 0.367879 1 1 1 2.71828 1 虚数部に反比例して絶対値が大きくなる。	*Polar* 極座標表示 X r. Y	「x r. y」**は**「x*^0j1*y」 r. 2 _0.416147j0.909297 2 r. 2 _0.832294j1.81859 *r. 2* $\Longleftrightarrow$ *1 r. 2* 2 ラジアン = 57.3 × 2 = 114.6^o  'marker'plot 0 ,(r.2),2 r.2

1.1.6 配列の演算

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
u .v *Deter-minant* 一般行列式	+ .	-/ . *は正則行列の行列式 A=: 1 6 6, 4 1 0,: 6 6 8 1 6 6 4 1 0 6 6 8 (-/ . *;+/ . *) A +---+---+ |_76|380| +---+---+ det ; per （**動詞の並列算**） *Permanent(+/ . *)* は *Determinant(-/ . *)* のマイナス方向も全てプラスする − − − | + | + + 1 6 6 | 1 6 4 1 0 | 4 1 6 6 8 | 6 6 サラスの方法 */ 6 1 6=36 */ 1 0 6=0 */ 6 4 8=192 ----------- */ 1 1 8=8 */ 6 0 6=0 */ 6 4 6=144 ----------- -228+152=-76 Determinant 228+152=380 Permanent	*Dot-product* 一般内積	X +/ . * Y は内積を表す (i.5)+/ . * i.5 30 (A=.>:i.3 4);(B=.>:i.4 2) ;A +/ . * B +----------+---+-------+ |1 2 3 4|1 2| 50 60| |5 6 7 8|3 4|114 140| |9 10 11 12|5 6|178 220| | |7 8| | +----------+---+-------+ A B A+/ .* B 50= +/ 1 2 3 4 * 1 3 5 7 140= +/ 5 6 7 8 * 2 4 6 8 u . v としていろいろな動詞を適用することも可能（一般内積） 1 2 3 +/ . = 1 2 5 2 NB. **マッチした数** 1 2 3 %: . + 1 2 5 1.41421 2 2.82843 NB. **対応する要素同士を加算して平方根** %: 2 4 8 1.41421 2 2.82843 4 6 4 3>./ . <./5 7 4 9 6 NB. MiniMax 4 6 4 3<./ . >./5 7 4 9 4 NB. MaxMini

%. *Matrix Inverse* 一般逆行列 %. Y	⍤	*(*行数 ≥ 列数*)* の行列の一般逆行列を与える。]a=. 1 0,: 0 2 1 0 0 2 %. a 1 0 0 0.5 M=.1 x: 3%~1 0,0 1,:2 1 1r3 0 0 1r3 2r3 1r3 %. M 1 _1 1 _1 5r2 1r2 **縦長の** Matrix **の逆行列**	*Matrix Divide* 行列の割算 X %. Y	*X* がベクトルで *Y* が *X* の次数に等しい正則行列ならば連立方程式の解を与える。 また *Y* の行数が列数より大きいときは、最小 2 乗解。 1 4 %. 1 3,: 1 4 _8 3 $\begin{cases} x_1 + 3x_2 = 1 \\ x_1 + 4x_2 = 4 \end{cases}$ cr=: %.}:"1 NB. Cramer **法** cr 1 3 1,: 1 4 4 1 0 _8 0 1 3 左に単位行列が出来れば良い 1 4 2 %. M _1 10 $y = -x_1 + 10x_2$ *NB.* 原点を通る 1 4 2 %. 1,. M 2 _3 6 $y = 2 - 3x_1 + 6x_2$ *1* を付加して定数項も求める。 reg=: %. 1&,.@] NB.Least square Regression 1 4 2 reg M 2 _3 6

/ *Insert* （副詞）		+/ 0 1 2 3 **は** (0+1+2+3) (+/ A); A=.i.3 4 +-----------+---------+ |12 15 18 21|0 1 2 3| | |4 5 6 7| | |8 9 10 11| +-----------+---------+ **スカラに作用しているので縦に合計** (+/ "1 A); A=.i.3 4 +-------+---------+ |6 22 38|0 1 2 3| | |4 5 6 7| | |8 9 10 11| +-------+---------+ **ランク 1 でベクトルに作用させ横に合計** (*/ A); A=.i.3 4 +------------+---------+ |0 45 120 231|0 1 2 3| | |4 5 6 7| | |8 9 10 11| +------------+---------+ **スカラに作用し縦に掛算**	*Table* 一般外積 X u/ Y	X */ Y は *X* と *Y* の外積 *u* としては他のいろいろな動詞を適用することも可能 1 2 3 */ 4 5 6 7 4 5 6 7 8 10 12 14 12 15 18 21 1 2 3 +/ table 4 5 6 7 +--+--------+ |+/|4 5 6 7| +--+--------+ |1 |5 6 7 8| |2 |6 7 8 9| |3 |7 8 9 10| +--+--------+ (>:i.9)*/ table >: i.9 **(かけ算の九九)** *table* は *J* の見出し用の英語のコマンド

次の *2* 項は外部接続詞に定義されており暫定サポート。
ADDON の *LAPACK* に精密な関数が多く含まれており、*J* から用いることができる（第 *4* 章参照）

128!:0 *QR* 分解		128!:0 が *QR* 分解を行う `] A=: 0 1 1 , 1 0 1,:1 1 0` `0 1 1` `1 0 1` `1 1 0` `6j3 ":(L:0)  128!:0 A` `+-----------------+-----------------+` `| 0.000 0.816 0.577| 1.414 0.707 0.707|` `| 0.707_0.408 0.577| 0.000 1.225 0.408|` `| 0.707 0.408_0.577| 0.000 0.000 1.155|` `+-----------------+-----------------+` **左はグラム・シュミットの直交化** ◇ 対称行列の固有値を *QR* 法で求める `rheval NB. R.Hui` `+/ .*&>/@|.@(128!:0)` `rheval ^:100 A NB.repeat 100` `2  0  0` `0 _1  0` `0  0 _1` 固有値は *2,_1*（重根）	
128!:1 上三角行列の逆行列		上三角行列の逆行列を計算 `128!:1 >{:128!:0 a=. 3 3 $ 20 ?. 20` `0.0618984 _0.0911557 _0.0220058` `0  0.141617  _0.104445` `0  0  0.0718834` `%.>{:128!:0]a` `0.0618984 _0.0911557 _0.0220058` `0  0.141617  _0.104445` `0  0  0.0718834`	

1.1.7 乱数と素数

用法	APL	結果と例（単項）	用法	結果と例（両項）
? ?. Roll Deal 乱数 ? Y ?. Y	?	?y は 0 から y – 1 までの整数乱数を生成する。 ?. はシードを固定した乱数（同じ乱数の発生が可能） ? 10 10 10 10 5 6 0 3 ? 3 # 0 0.0466292 0.330109 0.01346 0 1 間の一様乱数を 3 個生成 9!:42,9!:43 で Mersenne Twister 以外のシードの乱数も利用できる。	Deal 非重複乱数 X ? Y	6 ? 6 5 1 2 4 3 0 6 ? 6 1 5 0 4 3 2 ランダム配置とも言う ?. はシード固定の非重複乱数 6 ?. 6 5 1 2 4 3 0 6 ?. 6 5 1 2 4 3 0
p: Primes 素数 p: Y		右引数で指定した順位の素数を与える p: i.10 2 3 5 7 11 13 17 19 23 29 p:^:(_1) 10 4 右引数より小さい素数の数	X p: Y	左引数（8 個）で指定する素数の種々の要素 ({@>_4 _1 0 1 2 3 4 5) p:(L:0) 20 +--+-+-+-+---+-----+--+-+ |19|8|1|0|2 5|2 2 5|23|8| | | | | |2 1| | | | +--+-+-+-+---+-----+--+-+ _4 _1 0 1 2 3(q:) 4 5 _4, 4 20 の前後の素数 0 常に 1 1 素数が 1 のみの時 1 他は 0 2 2 2 5 の個数を表示 3 q :（素因数分解） 5 オイラー数 （20 以下の素数の数） _1 20 以下の素数の数
q: Prime-factor 素因数分解 q: Y		右引数の整数の素因数分解 q: 20 2 2 5 q: 700 2 2 5 5 7	Prime Exponents 素数の位置 X q: Y	素因数分解した素数の位置と個数を与える (p: i.9),: 9 q: 700 2 3 5 7 11 13 17 19 23 2 0 2 1 0 0 0 0 0

1.1.8　幾つかの解析関数

副詞や接続詞は名詞や動詞を引数とする。明示（*Explicit*）型では名詞と動詞を区別する場合の引用は次による。ここでも同じ記法を用いる。

名詞　*m*　左引数
名詞　*n*　右引数
動詞　*u*　左引数
動詞　*v*　右引数

d.	微分	接続詞
D.	微分	接続詞
D :	平均変化率	接続詞
H.	超幾何級数	接続詞
T.	テーラー級数	接続詞
t.	テーラー級数	副詞
t :	重み付きテーラー級数	副詞
p.	多項式	動詞
p..	多項式の微積分	動詞

副詞の機能は動詞のパワーアップである。副詞は左にしか動詞を引数として取らない。

接続詞は動詞と名詞、動詞と動詞を繋ぎながら一仕事する為に副詞とは区別して *J* に組み込まれた。接続詞は左右に動詞も名詞も引用する。

ここでは接続から生成される動詞が殆ど単項動詞であって両項動詞は別の動詞になるのではなく、単項動詞のオプション機能を持つものである。ここでは単項、両項とは区別しないで説明する。

用法	結果と例	参照
`d.` *Derivative* 微分 *(接続詞)* *u d.n*	*u d. n* は関数 *u* の *n* 階の数値微分を行う。ランクは *0* に固定されている `   *: d.1 NB. 1 階微分` `+:        NB. 2x` `   *: d.2 NB. 2 階微分` `2"0       NB. 2` `    *: d.1 x=.1 2 3` ` 2 4 6` `     (cube=.^&3) y=. 2 3 4` `8 27 64` `   cube d.(1) 2 3 4` `12 27 48` `   cube d.(2) 2 3 4` `12 18 24`	$x^2 \to 2x$ $1^2, 2^2, 3^2 \to 2\times 1, 2\times 2, 2\times 3$ $fx = x^3$ $f'x = 3x^2$ $2^3, 3^3, 4^3 \to 3\times 2^2, 3\times 3^2, 3\times 4^2$ $f''x = 6x$ $3\times 2^2, 3\times 3^2, 3\times 4^2 \to 6\times 2, 6\times 3, 6\times 4$

D.
Derivative
微分
*(*接続詞*)*
u D.n

「*u* D.*n*」も関数 *u* の *n* 階の数値微分を行う。
ランクは *0* に固定されてない

```
    (cube=.^&3"0)y=.2 3 4
 8 27 64
    cube D.1 y
 12 27 48
    cube D.2 y(2 階微分)
 12 18 24

   ^&3("0) D.(1) 2 3 4
12 27 48
   ^&3 D.(1) 2 3 4
12  0  0
 0 27  0
 0  0 48


   */\ 4 3 2
4 12 24
```

*/\はベクトルに働くと

$$x_0$$
$$x_0x_1$$
$$x_0x_1x_2$$

という多変数関数になる。

```
    */\ D.(1) 4 3 2
1 3  6
0 4  8
0 0 12
Gradient
```

K.E.Iverson(Calculus 参照*)*

$6\times2, 6\times3, 6\times4$

	x_0	x_0x_1	$x_0x_1x_2$
$\frac{\partial}{\partial x_0}$	1	x_1	x_1x_2
$\frac{\partial}{\partial x_1}$	0	x_0	x_0x_2
$\frac{\partial}{\partial x_2}$	0	0	x_0x_1

D: Secant Slope 平均変化率 *(接続詞)* `u D:n`	Δx を指定する *(例えば Δ x=1 0.1 0.01)* Δx が小さければ一階微分に近づく。 `   cube=.^&3"0` `   1 cube D:1 y=.2 3 4` `19 37 61` `1 0.1 0.01 cube D:1/2 3 4` `     19      37      61` `  12.61   27.91   49.21` `12.0601 27.0901 48.1201`	$\Delta x = 1\ 0.1\ 0.01$ $\frac{3^3 - 2^3}{1} = 19$ $\frac{2.1^3 - 2^3}{0.1} = 12.61$ $\frac{2.01^3 - 2^3}{0.01} = 12.0601$
`d._1` 積分（逆関数で定義） `u d._1 n`	微分の逆関数で積分を定義 `   cube=:^&3` `   cube d.(_1) 2 3 4` `4 20.25 64`	$\frac{2^4}{4}, \frac{3^4}{4}, \frac{4^4}{4}$ シンプソン積分は *system /packages/math/integrat.ijs* にある。

T. Taylor appro ximation (接続詞) `u T.n`	`u T.n` は関数 *u* の *n* 項までのテイラー展開で近似した値 `   ^T.5 i.3 NB. 5 項近似` `1 2.70833 7` `   ^T.30 i.3` `1 2.71828 7.38906  NB.30 項近似` `   ^ i.3` `1 2.71828 7.38906  NB.Exact e^n` e^0, e^1, e^2	
t. Taylor coeffi cient (副詞) u t. Y	`u t.` y は関数 *u* のテイラー展開の *y* での係数 `   ^ t. 1 2 3` `1 0.5 0.166667` $\frac{1}{1!}, \frac{1}{2!}, \frac{1}{3!}$ `   sin=.1&o.[cos=.2&o.` `   sin t. i.5` `0 1 0 _0.166667 0` `   cos t. i.5` `1 0 _0.5 0 0.0416667`	`x u t.` y は *x* ^ *y* と `u t.` y との積
t: Weighted Taylor coeffi cient (副詞) u t: Y	階乗により重みづけられたテイラー展開。「*u t: Y*」は `(!Y)* t. Y` と同じ $(-1)^k \frac{x^{2k+1}}{(2k+1)!}$ `1 x:(sin t. i.6),:sin t: i.6` `0 1 0 _1r6 0 1r120` `0 1 0   _1 0     1` `------------------` `0 1 2    3 4    5` `1 x:  %! 1 3 5 7` `1 1r6 1r120 1r5040` **テーラー展開の各項の符号ををを`_1 0 1` で示す.** `   (!i.5)*sin t. i.5` `0 1 0 _1 0` `   cos t: i.5` `1 0 _1 0 1`	$sinx = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \cdots$ $x - \frac{x^3}{3!} + \frac{x^5}{5!} \to 1 - \frac{1}{6} + \frac{1}{120}$ x^k を除いた部分

H. Hyper-geomet-ric 超幾何級数	非復元抽出の場合でほぼ *2* 項分布 ${}_rC_k p^k q^{r-k}$ に等しい。 $H_{n,m.r}(k) = \frac{{}_mC_k \times_{n-m} C_{r-k}}{{}_nC_r}$ ◊ *K.E.Iverson* が *D.Knuth* 達 *(GPK)* の *Concrete Mathematics Comparison* を著したときに組み込まれた。 *n=10000* 個のロットで不具合率 *0.03(m=300). r=10* 個取り出し *k=1* 個エラーの確率 `((1!300)*9!9700)%10!10000` `0.228249` ◊ 超幾何級数を用いて積分の部分がが簡略に定義でき、誤差関数 *(erf),* ガンマ関数などの特殊関数が作成できる。 ◊ *E.Show* 達が *H.* を用いた *erf* 関数から種々の分布関数を作成したものが *addon/stat/normal/distribs/normal.ijs* 等に入っている。 *erf(*誤差関数）を求める $erf(x) = \frac{2}{\sqrt{\pi}} \int^x e^{-t^2} dt$ 正規分布関数 $(\Phi) = \frac{1}{2}\left(1 + erf\left(\frac{x}{\sqrt{2}}\right)\right)$ `erf=:(1 H. 1.5)@*: * 2p_0.5&* % ^@:*:` `  NB. error function` `n01cdf=:-:@:>:@:erf@:((%:0.5)&*)` `NB. CDF of normal 0,1` ◊ ここの *H.* は *erf* を作成するための *J* の内部関数を公開したとも思われるが、この *H.* は合流型超幾何級数と呼ばれるものである。 *m H. n y* ↔ $F(m; n, y)$ $e^z, sinz, cosz, log(1+z)$ や正弦積分、余弦積分、指数積分など多くの関数が超幾何級数を用いて定義できる。（超幾何関数）

p. *Root* 多項式の根 *p. Y*		多項式の解 `   p. 2 3 1` `+-+-----+` `|1|_2 _1|` `+-+-----+` $fx = 2 + 3x + x^2$ 解は*_2,_1* 最初のボックスは収束までの反復状況を示す	*Polynomial* 多項式 *X* p. *Y*	左引数で与えた係数の多項式の右引数での値を与える `  x=.1 5 0 4 [y=._1 0 2` `    x p. y` `_8 1 43` `  1 5 0 4&p. _1 0 2` `_8 1 43` $fx = 1+5x+4x^3$ の *x=_1,0,2* の値
p.. *polynomial deriva-tive* *p.. Y*		多項式の微分 `  p.. 1 3 3 1` `3 6 3` $f = 1 + 3x + 3x^2 + x^3$ $f' = 3 + 6x + 3x^2$	*polynomial Integral* *m p.. Y*	左引数は付加する積分定数。多項式の積分の係数を示す *((x^k)* を除いた部分*)* `  3  p.. 1 3 3 1` `3 1 1.5 1 0.25` $3 + x + \frac{3x^2}{2} + \frac{3x^3}{3} + \frac{x^4}{4}$

Table of Derivative

Name			*Function*	*Derivative*
Constant			*a*”0	0”0
Identity			]	1”0
Constant Times	定数倍	$(kf)' = kf'$	*a*”0∗]	*a*”0
Sum	和の微分	$(f+g)'(x) = f'(x) + g'(x)$	$f+g$	$(f\ d.1) + (g\ d.1)$
Difference	差の微分		$f-g$	$(f\ d.1) - (g\ d.1)$
Product	積の微分	$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$	$f*g$	`(f*(g d.1)+((f d.1)*g)`
Quotient	商の微分	$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$	`f%g`	`(f%g)*((f d.1)%g)-((g d.1)%g)`
Composition	合成関数の微分	$(f(g(x)))' = f'(g(x))g'(x)$	`f@g`	`(f d.1)@*(g d.1)`
Inverse	逆関数の微分	$\frac{d}{dx}f^{-1}(y) = \frac{1}{\frac{d}{dx}f(x)} \to \frac{dx}{dy} = \frac{1}{\frac{dy}{dx}}$	*f INV*	`%@(f d.1@(f INV))`
Reciprocal	逆数の微分	$\left(\frac{1}{g}\right)'(x) = -\frac{g'(x)}{(g(x)^2}$	`%@f`	`-@(f d.1 % (f * f))`
Power			`^&n`	`n&p. * ^&(n-1)`
Polynomial	多項式の微分	$(c_n x^n + c_{n-1}x^{n-1} + \cdots + c_1 x + c_0)' = nc_n x^{n-1} + (n-1)c_{n-1}x^{n-2} + \cdots + 2c_2 x^2 + c_1$	`c&p.`	`(deco c)&p.`

```
deco=:}.@[ * i.@#)
INV=:^:_1
```

(Issue:) K.E.Iverson Calculus(2-K)

1.1.9　順列 · 組み合わせ · 群

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
! *Factorial* 階乗 `! Y`		！ *y* は 1, 2.....,*y* の積 *(*階乗*)* を与える。`*/@(>:@i.)"0` としても同じである `   ! 3 4 5` `6 24 120` **「`!@<: y`」はガンマ関数** `   !@<: 5` `24` 整数値以外でも算出できる。 Γ(4.5) `   !@<: 4.5` `11.6317` `   3.5*2.5*1.5*0.5*%:1p1` `11.6317` $3.5 \times 2.5 \times 1.5 \times 0.5 \times \sqrt{\pi}$	*Out- Of* ２項係数 `X ! Y`	*x*!*y* は *y* 個の中から *x* 個取り出す組合せの総数 $\binom{n}{k} = {}_nC_k = \frac{n!}{(n-k)!k!}$ `   2 ! 8` `28` ${}_8C_2 = \frac{8!}{6!2!}$ `   A=: i.5` `   |: A !/A` `1 0 0 0 0` `1 1 0 0 0` `1 2 1 0 0` `1 3 3 1 0` `1 4 6 4 1` *Pascal* の三角形

A. *Anagram Index* 辞書式順序のインデクス *A. Y*		数列や文字列の辞書式順序（順列）のインデックスを与える。 `   ]Y=.(i.6)A.0 1 2` `0 1 2` `0 2 1` `1 0 2` `1 2 0` `2 0 1` `2 1 0` `   A. Y` `0 1 2 3 4 5`	*Anagram* 辞書式順序 `X A.Y`	*X* で与えたインデックスに対応する辞書式順序（順列）を与える `   (i.6)A.'abcd'` `abcd` `abdc` `acbd` `acdb` `adbc` `adcb NB.6` **通りの組合せ** `   2!4 NB. 4C2` `6` `   2 sample_C 4` `0 1` `0 2` `0 3` `1 2` `1 3` `2 3 NB.6` **通りの取出し** `plist=: i.@! A. i.` `sample_C=: 4 : 0` `~./:"1~(i.x){"1 plist y` `)`
C. *Cycle Direct* 巡回置換ベクトル `C. Y`		巡回置換ベクトルと直接置換ベクトルとを相互に切り替える `   P=. 2 4 0 1 3` **直接置換ベクトル** `  ] C0=. C. P` `+---+-----+` `|2 0|4 3 1|` `+---+-----+` `   C1=. 1|. L:0 C0` `   C. C0` `2 4 0 1 3` `   C. C1` `2 4 0 1 3`	*Permute* 置換 `X C.Y`	*X* で与えた置換ベクトルに従って、*Y* を置換る。 ◇ `   P C. 'ABCDE'` `CEABD` `   P { 'ABCDE'` `CEABD` **とする事に等しい** ◇ `   C0 C. 'ABCDE'` `CEABD` **巡回置換ベクトルによる巡回置換** `   C1 C. 'ABCDE'` `CEABD` **ボックス内（巡回置換ベクトル内）で回転したものを** **使っても同じ巡回置換になる**

1.2　論理演算とブール数

1.2.1　論理演算と 0,1 の指標作成

用法	APL	結果と例（単項）	用法	結果と例（両項）
* *Signum* 符号 * *Y*	×	実数に対しては、正には *1*、*0* には *0*、負には *_1* を付与。 `    * _5 0 4` `_1 0 1` `    * 3j4` `0.6j0.8` **複素数には大きさと偏角を与える** `   |3j4` `5` `   *. 3j4 NB.Lerngth/Angle` `5 0.927295` `  *. * 3j4` `1 0.927295` **大きさが 1 で、同じ偏角を与える**	*Times* 乗算 *X * Y* *(1.1.1)*	
~. *(副詞)* *Nub* 重複排除 (~ .)		*Y* の重複要素を排除し出現順に並べる `   ~. 1 1 3 2 4 2 3` `1 3 2 4` `   ~. 'japan'` `japn`	無し	
~: *Nubsieve* 重複指示 ~: Y (~:)		重複している合は重複の最初の箇所を *1* で示す。(*2* 度目は *0*) 重複していない場合も *1* であり、オリジナルか重複かを表示 `    ~:1 1 2 2 3 3` `1 0 1 0 1 0` `   ~: 1 1 3 2 4 2 3` `1 0 1 1 1 0 0` `   ~: 'japan'` `1 1 1 0 1` **カット (<;.n) の指標としてグループ分けに有用**	*Not-equal* 不等 X ~:Y	X と Y が等しければ *0*、そうでなければ *1* `  20 30 40 ~: 40 30 20` `1 0 1` `  'japan' ~: 'jappn'` `0 0 0 1 0` **文字（要素）ベース** ~:と漢字 `~: s:/' 山田';' 田中';' 山本';' 田中';' 山田'` `1 1 1 0 0`

i. *(Integers)* *(1.3.2)*	⍳	(i. >./)9 8 7 6 7 9 0 **最大値の最初の位置（単項のフック）**	*Index of First* インデクス（前方） X i.Y	左引数の要素の右引数における最初の位置を返す。要素がない場合は最大値を返す 7 8 9 7 i. 8 7 6 7 9 1 0 4 0 2 'abcd' i. 'a bird' 0 4 1 4 4 3 a b i r d
i: *(Steps)* *(1.3.2)*		(i:>./) 9 8 7 6 7 9 5 **最大値の最後の位置（単項のフック）**	*Index of Last* インデクス（後方） X i:Y	左引数の要素の右引数における最後の位置を返す。要素がない場合は最少値を返す 7 8 9 7 i: 8 7 6 7 9 1 3 4 3 2 'japan' i. 'japan' 0 1 2 1 4 a **は** 1 **番目** 'japan' i: 'japan' 0 3 2 3 4 a **の後ろの位置** 3
I. *Indices* 指標 I. Y		**0 1 の指標の場合は 1 の箇所の位置を示す** A=: 0 0 1 0 1 0 1 1 I. A 2 4 6 7 A,: i.8 0 0 1 0 1 0 1 1 0 1 2 3 4 5 6 7 **指標が 0 1 でない場合は、個数（0 は 0 個）を重複した出現順の数で表示する** I. 0 3 2 1 0 2 1 3 1 1 1 2 2 3 5 5 6 7 7 7 3 - - 2 - 1 2 - 1 3 - -	*Interval-Index* X I.Y	左引数で示した昇順または降順に右引数を配置する順を示す 2 5 6 I. 10 3 5 7 8 9 3 1 1 3 3 3 8([:I. =)Y=.10 3 5 7 8 9 4 8 **の位置** 8(<:/)Y=. 10 3 5 7 8 9 1 0 0 0 1 1 8([:I.<:/)Y=.10 3 5 7 8 9 0 4 5 8 **と等しいかより大きい数値の位置**

= *Self-Classify* 自己分類 = *Y*	=	重複を分類して判別し、*1* で重複、*0* で非重複を表示する `   = 'aabbccaadd'` `1 1 0 0 0 0 1 1 0 0 NB.a` `0 0 1 1 0 0 0 0 0 0 NB.b` `0 0 0 0 1 1 0 0 0 0 NB.c` `0 0 0 0 0 0 0 0 1 1 NB.d` `a a b b c c a a d d` `    a=.3 3$'abcdefghi'` `  (=a); a` `+-----+---+` `|1 0 0|abc|` `|0 1 0|def|` `|0 0 1|ghi|` `+-----+---+` **リストは要素ベース、マトリクスは行ベースで照合** `    = i.3` `1 0 0` `0 1 0` `0 0 1` 単位行列を生成 *(*`=@i. y`*)*	*Equal* 等しい *X = Y*	XとYが等しければ *1*、そうでなければ *0* `20 30 40 = 40 30 20` `0 1 0` `   (i.3 3 )=i.3 3` `1 1 1` `1 1 1` `1 1 1` **マトリクスもスカラ(要素)ベースで照合** `a=: 3 3 $ 'abcdefghi'` `a=a` `1 1 1` `1 1 1` `1 1 1` **文字（要素）ベース**
E. 無し			*Memder-of Interval* パターンの所属 *X E.Y*	右引数の中に左引数 *(*パターン*)* の先頭の位置に *1* を返す。そうでなければ *0* を返す。 `   2 3 E. 1 2 3 5 2 3 5` `          x x   x x` `0 1 0 0 1 0 0` `patarn 2 3` **の始まる位置** `   2 3 4 E. 1 2 3 5 2 3 5` `0 0 0 0 0 0 0` `patarn 2 3 4` **はない** `'co' E. 'cocoa'` `1 0 1 0 0` ワードベースの比較であり、左引数がキーワード

e. *Raze (in)* 部分所属 *e. Y*	∈	Yの各要素を照合してブール数で表示する。 `(e.i.5);  e. 'japan'` `+---------+---------+` `|1 0 0 0 0|1 0 0 0 0|j` `|0 1 0 0 0|0 1 0 1 0|a` `|0 0 1 0 0|0 0 1 0 0|p` `|0 0 0 1 0|0 1 0 1 0|a` `|0 0 0 0 1|0 0 0 0 1|n` `+---------+---------+` `0 1 2 3 4  j a p a n` `  = 'japan'` `1 0 0 0 0|j` `0 1 0 1 0|a` `0 0 1 0 0|p` `0 0 0 0 1|n` `j a p a n`	*Member(in)* 要素の所属 *X e. Y*	左引数の要素が右引数の中に含まれていれば *1*、そうでなければ *0* を返す `8 7 6 7 9 e. 7 8 9 7` `1 1 0 1 1` `'japan korea' e. 'japan'` `1 1 1 1 1 0 0 0 0 0 1` `j a p a n _ k o r e a` **文字ベース（要素）で照合**
`-:` *(Halve)* *(1.1.1)*			`-:` *Match* 一致 `X -: Y`	左引数と右引数とがマッチしていれば *1*、そうでなければ *0* (要素ではなく全体を照合) `  1 2 3 -: 1 2 3` `1` `y;(y=y);y -: y=. i.3 3` `+-----+-----+-+` `|0 1 2|1 1 1|1|` `|3 4 5|1 1 1| |` `|6 7 8|1 1 1| |` `+-----+-----+-+` `  y  y=y  y-:y` `  'japanese'-: 'japan'` `0` `  'japanese' e. 'japan'` `1 1 1 1 1 0 0 0` **'石岡駅 着' -: '石岡駅 発'** `0` NB. **スペースの全半角も識別** **文字列も文全体での一致不一致**

< *(Box)* *(1.4)* `< Y`	<	`</ 2 3 4` `0 NB. 2<(3<4)` `</. 2 3 4` `+-+-+-+` `|2|3|4|` `+-+-+-+` `<\ 2 3 4` `+-+---+-----+` `|2|2 3|2 3 4|` `+-+---+-----+` `<\. 2 3 4` `+-----+---+-+` `|2 3 4|3 4|4|` `+-----+---+-+`	*Less-than* *X <Y*	左引数が右引数より小さけ *1*、そうでなければ *0* `20 30 40 < 40 30 20` `1 0 0`
> *(Open)* `> Y` *(1.4)*	>	`>/ 2 3 4` `1 NB. 2>(3>4)` `>\ 2 3 4` `2 0 0` `2 3 0` `2 3 4`	*Larger-than* 大きい `X > Y`	左引数が右引数より大きければ *1*、そうでなければ *0* `20 30 40 > 40 30 20` `0 0 1`
`<:` *Decrement* *(1.1.1)*	≤	*Decrement* (`<:`) `<: Y` $\Longleftrightarrow Y-1$ *Not(-.)* `-. Y` $\Longleftrightarrow 1-Y$	`<:` *Lesser or equal* 小さいか等しい `X <: Y`	左引数が右引数より小さいか等しければ *1*、そうでなければ *0* `20 30 40 <: 40 30 20` `1 1 0`
Increment `>:` *(1.1.1)*	≤	*Increment* (`>:`) `>: Y` $\Longleftrightarrow Y+1$	`>:` *Larger or equal* 大きいか等しい `X >: Y`	左引数が右引数より大きいか等しければ *1*、そうでなければ *0* `20 30 40 >: 40 30 20` `0 1 1`

1.2.2　ブール代数

X,Y 共に論理数で両項のとき

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
*. *Length* */Angle* *(1.1.5)*	∧		論理積 *And* 最小公倍数 *LCM* *(1.1.1)* *X* *. *Y*	左右の引数が共に *1* のときに *1*、そうでなければ *0* `0 0 1 1 *. 0 1 0 1` `0 0 0 1` ＸとＹの最小公倍数 `24 *. 60` `120` ブール代数で文字列は演算できない
+. *Real* */Imagin* *ary* *(1.1.5)*	∨		*Or* 論理和 *GCD* 最大公約数 *X* +. *Y* *(1.1.1)*	左右の引数が共に *0* のときに *0*、そうでなければ *1* `0 0 1 1 +. 0 1 0 1` `0 1 1 1` ＸとＹの最大公約数 `24 +. 60` `12`
*: *Squre* *(1.1.1)*	*		*Not-And* 否定論理積 X*:Y	左右の引数が共に *1* のときに *0*、そうでなければ *1* `0 0 1 1 *: 0 1 0 1` `1 1 1 0`

+: *Double* *(1.1.1)*			*Not-Or* 否定論理和 X+:Y	左右の引数が共に *0* のときに *1*、そうでなければ *0* 0 0 1 1 +: 0 1 0 1 1 0 0 0
=			= *XNOR*	左右の引数が同じときに *1*、異なければ *0* *XNOR(eXclusive NOR) (X,Y* が共に論理数のとき） 0 0 1 1 = 0 1 0 1 1 0 0 1 'JAPLA'='JPALA' 1 0 0 1 1
-. *Not* 否　定/補数 -.Y	~	右引数が *0* なら *1*、*1* なら *0* -. 1 0 0 1 **論理数でなければ補数** -. 0.6 0.7 0.55 1.2 0.4 0.3 0.45 _0.2 *(-*はマイナスキー*)*	*Less* X -.Y	左の引数から右引数の要素を除いたものを出力する (i.5) -. 1 3 0 2 4 'JAPLA' -.'JP' ALA (Remove Items) **マトリクスでは列の一致が必要** (i. 2 5) -. i.5 5 6 7 8 9 (i.2 5)-.0 1 3 4 5 0 1 2 3 4 5 6 7 8 9

1.2.3　基底と 2 進、n 進の計算

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
#. *Base-2* 2 進数 *#. Y*	⊥	2 進数で与えた右引数を *10* 進数に変換する `#. 1 1 1 1` `15` `2 #. 1 1 1 1` `15`	*Base* *10* 進化 *X #. Y*	左引数を底（*10* 進）とする加重合計値 `10 #. 1 2 3` `123` `(2;4;8;16) #./(L:0) 4 5 6` `+--+--+---+----+` `|32|90|302|1110|`**10 進の値** `+--+--+---+----+` `2 4 8 16` **進数** $(4 \times 2^2) + (5 \times 2) + 6 = 32$ $(4 \times 8^2) + (5 \times 8) + 6 = 302$ `60 #. 1 30 20` `5420` **（1 時間 30 分 20 秒の秒数）**
#: *Antibase* *2* *10* 進数の 2 進化 *#: Y*	⊤	*10* 進数で与えた右引数を 2 進数に変換する `#: i.8` `0 0 0 ····· 0` `0 0 1 ····· 1` `0 1 0 ····· 2` `0 1 1 ····· 3` `1 0 0 ····· 4` `1 0 1 ····· 5` `1 1 0 ····· 6` `1 1 1 ····· 7`	*Antibase* *n* 進化 *X #: Y*	*10* 進数で与えた右引数を左引数の底による進数で表示 `4 4 #: i.6` `0 0...0` `0 1...1` `0 2...2` `0 3...3` `1 0...4` `1 1...5` **（4 進数表示）** `10 10 10 #: 658` `6 5 8` **文字化しなくとも分解できる（2 桁目を照合する時などに便利）** `24 60 60 #: 5420` `1 30 20` **（秒の時間/分/秒へ変換）**

用法	APL	結果と例（単項）	用法	結果と例（両項）
b. (副詞) *Basic Characteristic (1.1.2)* *u b.n*		◊ *u b.0* はランクを表示する `   * b.0` `0 0 0` ◊ `u b._1` は逆関数を表示する `   *: b._1` `%:` ◊ *u b.1* は *J* のマクロ定義の内側を表示する（全てではない） `   ^ b.1 NB. e` `$&1@(}.@$)`	*Boolean* ブール数 ビットワイズ演算 *X b. Y*	*2* 進演算用に *19* 種類のプリミティブが別途構成されている。引数は次のように割り付けられており、(プリミティブの代わりに) 引数と *b.* で *bitwise* 演算ができる *a* 0 — 15 *truth table* *b* _16 – _1 *integer* *c* 16 – 34 *bitwise* *a* *b* *c* 1 _15 17 *. *AND* 6 _10 22 ~: *XOR* 7 _9 23 +. *OR* 8 _8 24 +: *NOR* 9 _7 25 = *XNOR* 14 _2 30 *: *NAND* 32 *rotate* 33 *shift* 34 *signed shift* `   1 0 1 0 (1 b.) 1 1 0 0` `1 0 0 0 NB. AND` `   1 0 1 0 (7 b.) 1 1 0 0` `1 1 1 0 NB. OR` `   123 (17 b.) 456` `72  NB. AND` `   (10#2)&#: 123 456` `0 0 0 1 1 1 1 0 1 1NB.123` `0 1 1 1 0 0 1 0 0 0NB.456` `x x` `#.0 0 0 1 0 0 1 0 0 0` `72` *bitwise* 演算を使用する場合は *Vocaburaly* の *Boolean* 参照

1.3 配列の形、変形、連結

1.3.1 ランク

J ではランクという概念が採用された。*APL* の軸とは微妙に差異がある。

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
" *Rank* ランク u"n Y		動詞 u を作用させるとき、ランクとしてセルのナンバーを指定する。] A=. i.2 3 0 1 2 3 4 5 ◊ 省略した場合は最大ランク。 +/ A 3 5 7 ◊ 列方向 +/ "2 A . 3 5 7 ◊ 行方向（リストに作用） +/ "1 A 3 12 ◊ ランク *0* は何もしない +/ "0 A 0 1 2 3 4 5 デフォルトではスカラ同士の方向（縦）を、*"1* ではベクトル方向（横）に作用する。	*Rank* ランク *X u"n Y*	X=.7 8 9 Y=.i.2 3 ◊ *X* は *Y* のリストに作用し、*X* が拡張される X ,"1 Y 7 8 9 0 1 2 7 8 9 3 4 5 ◊ *X* は *Y* のテーブルに作用。*X,Y* と同じである X ,"2 Y 7 8 9 0 1 2 3 4 5 ◊ *X* の要素（アトム）同士が連結 X ,"0 X 7 7 8 8 9 9

◊ テーブルの場合

```
 ]  B=. i.2 3 4
 0  1  2  3
 4  5  6  7
 8  9 10 11

12 13 14 15
16 17 18 19
20 21 22 23
```

◊ 行方向の和

```
   +/ "1 B
 6 22 38
54 70 86
```

◊ 列方向の和

```
   +/"2 B
12 15 18 21
48 51 54 57
```

◊ 串指し算

```
   +/"3 B
12 14 16 18
20 22 24 26
28 30 32 34
```

ランクを指定しない場合の最大ランク (`"3`) **は**
串刺し算

1.3.2 配列の形と変形、連結

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
`i.` *Integers* 整数生成 `i.Y`	*ι*	*0* から *Y-1* までの整数列を生成する（*0* オリジン） `i.4` `0 1 2 3` `i. _4` `3 2 1 0` **負の整数なら降順の整数列** `i.2 3` `0 1 2` `3 4 5` **テーブルの形でも生成できる**	*Index- of* 指標 `X i.Y`	右引数が、左引数の中で最初に表われる位置のインデクスを返す。 `7 8 9 7 i. 8 7 9 7 9 10` `1 0 2 0 2 4` `7 8 9 7 i. 8 7 9 6 9` `1 0 2 4 2` **左引数にない場合には`#X` を返す。** `i.~'JAPLA'` `0 1 2 3 1`
`i:` *Steps* ステップ `i: Y`		**マイナス側も含めた 1 刻みの整数列を生成** `i:4` `_4 _3 _2 _1 0 1 2 3 4` `i:_4` `4 3 2 1 0 _1 _2 _3 _4` `i:10 j. 5` `_10 _6 _2 2 6 10` *−y ↔ y* 間を *j.* で指定する数で刻む `(20%5->4)`	*Index- of* *Last* 最終指標	右引数が、左引数の中で最後に表われる位置のインデクスを返す。 `7 8 9 7 i: 8 7 9 7 9 10` `1 3 2 3 2 4` **左引数にない場合には`#X` を返す。** `i:~'JAPLA'` `0 4 2 3 4`

$ *Shape-of* 形状 $ Y	*ρ*	**Y の各ランクの形状（要素数）を返す** `   ]S=. i.3 4` `0 1  2  3` `4 5  6  7` `8 9 10 11` `   $ S` `3 4  NB. 3 行 4 列` `    'JAPLA',:'APL'` `JAPLA` `APL` `  $  'JAPLA',:'APL'` `2 5`	*shape* 変形 *X $ Y*	**X で指定する形に Y のアイテムを変形する。** `    3 4 $ i.12 NB. 3 行 4 列` `0 1  2  3` `4 5  6  7` `8 9 10 11` `   2 2 $  i.3 4 NB. 2 枚 2 列` `0 1  2  3` `4 5  6  7` `8 9 10 11` `0 1  2  3` `  3 3 $ 'abcdeghijklm'` `abc` `deg` `hij`
# *Tally* アイテム数 # Y		アイテム（引数より *1* つランクの低いセル）の数 `   # i.3 4` `3` `   # 4` `1` **アトムのアイテムはそれ自身** `mean=. +/%#  NB. mean` $\frac{\sum x}{n}$	*Copy* 複写 *X # Y*	右引数を左引数で指定した数だけコピーする。 `   5 # 1` `1 1 1 1 1` `   1 2 3 # 4 5 6` `4 5 5 6 6 6` `   10?.10` `6 9 1 4 0 2 3 8 7 5` `   (2&< # ]) 10?.10` `6 9 4 3 8 7 5` **10 より大きいもののみコピー** (u #]) y で動詞 *u* の指示により *copy* する

			(#) （圧縮）	0 0 1 0 1 # 1 2 3 4 5 3 5 **指標** 1 **の箇所のみに圧縮する** E=.=/〜i.3(**単位行列**) ,E #"1 A=.i.3 3 0 4 8 1 0 2 0 3 # 'JAPLA' JPPAAA 2j1 0 2j2 0 3 # 'JAPLA' JJ PP AAA 2j1 0 2j2 0 3 # !.'*''JAPLA' JJ*PP**AAA mjn # y **は** n **個のスペースを挿入。** mjn # !.'t' y **は** n **個の** t **を挿入** 行列の圧縮 *(A* の対角要素*) APL* の圧縮 *(/)* に対応。
#^:_1	\		*Expand* *expand* X#^:_1Y	*Copy* の逆関数。*X* の指標により *(1* に該当する個所を*)* 拡張する 1 0 1 1 0 1 1#^:_1(>:i.5) 1 0 2 3 0 4 5 **英語コマンドの** expand **もある。** 1 0 1 1 0 1 1expand(>:i.5) 1 0 2 3 0 4 5 1 0 1 1 0 1 1expand'JAPLA' J AP LA 1 0 1 1 0 1 1#^:_1 (!.100)]i.5 NB.100 **を挿入** 0 100 1 2 100 3 4 1 0 1 1 0 1 1 # ^:_1 !.'*' 'JAPLA' J*AP*LA NB. ***を挿入**

,
Ravel
リスト化
,Y

右引数をリストに変形する。

```
   ]S=.i.2 3
0 1 2
3 4 5
   , S
0 1 2 3 4 5
    ($ 3);$ ,3
++-+
||1|
++-+
```

アトムもリストに変形される

Append
連結
X , Y

右引数を左引数の最大ランクの方向に連結する。*(*アイテムを連結）左引数がリストなら横に、テーブルなら下に連結する

```
   1 2 3,4 5 6
1 2 3 4 5 6
  'japan','korea'
japankorea
 (i.2 3);(i. 2 3),6 7 8
+-----+-----+
|0 1 2|0 1 2|
|3 4 5|3 4 5|
|     |6 7 8|
+-----+-----+

  1 2 3, (4 5,: 6 7)
1 2 3
4 5 0
6 7 0
 a=.'a b',:'c d'
 b=.'A B C',: 'D E F'
 a,b
a b
c d
A B C
D E F
```

要素数が不足なら数値なら 0、文字列ならスペースが付加される。

,.
Ravel Items
テーブル化
`,. Y`
カンマとドット

見かけはリスト（ベクトル）のようだが
3 行 1 列のテーブル（行列）である

```
    ,. i.3
0
1
2
  ,.i.2 2 3
0 1 2 3  4  5
6 7 8 9 10 11
   $ ,.  i.2 2 2 3
2 12
```

あくまでテーブル化を行う

```
  ,./  ,: i.2 3
0 1 2
3 4 5
   $ ,./  ,: i.2 3
2 3
```

BOX を開いたときに *$ Y → 1 2 3* のような変則配列が現れることがあるが，`,./Y` で最大ランクを減次できる

Stitch
縦連結
X ,. Y

x,.y は「*x*」と「*,.y*」とを「,」で連結したのと同じ *(side by side)*

```
1 3 5 ,. i.3
1 0
3 1
5 2

5 ,. i.3 2
5 0 1
5 2 3
5 4 5
```

,:
Itemize
アイテム化
,: Y

ランクを 1 つ増やした高次のアレイを作る。

```
   ]S=.,: i.2 3
 0 1 2
 3 4 5
```

見かけはテーブルのようだが

```
   $ S
 1 2 3  NB.1 が入っていてレポート
 ($ ,: 3) ;  $ 3
+-++
|1||
+-++
```

Laminate
層連結
X ,: Y

セルを積重ねて、1 つ高いランクになるように上層、下層に連結する

```
   (i.6),: 3 4 5 6 7 8
0 1 2 3 4 5
3 4 5 6 7 8
  'korea',: 'japan'
korea
japan
 ]S=.(i.3),:i.2 3
0 1 2
0 0 0 NB. add 0

0 1 2
3 4 5
 $ S
2 2 3
   1 2 3,: 4 5 6 7
1 2 3 0
4 5 6 7
   'a b',: 'A B C D'
a b
A B C D
```

要素数が不足なら数値なら 0、文字列ならスペースが付加される。

|. *Reverse* 逆順 |.*Y*

Φ

アイテム（引数より１つランクの低いセル）を逆順に並べ替える

```
    |. 0 1 2 3 4
4 3 2 1 0
    |. 'abcdefg'
 gfedcba
```

各要素を逆順にする。

```
    |. 3 3$'abcdefghi'
 ghi
 def
 abc
     |. i.2 2 2
 4 5
 6 7

 0 1
 2 3
```

アイテム（ここではテーブル）の順序を反対にする

Rotate (Shift) 回転 *X* |. *Y*

右引数のアイテムをＸ個だけ回転する。

```
   1 |. 0 1 2 3 4
1 2 3 4 0
    _1  |. 0 1 2 3 4
 4 0 1 2 3
```

（Ｘが負なら後ろから回転）

```
    2  |.  0 1 2 3 4
 2 3 4 0 1
```

（1 |. を２度適用と同じ）

```
(i.3 4);0 _1 _2|."(0 1)i.3 4
+---------+---------+
|0 1  2  3| 0  1 2 3|0
|4 5  6  7| 7  4 5 6|_1
|8 9 10 11|10 11 8 9|_2
+---------+---------+
   i.3 4     twist
```

ランク"0 1 を用いたシフト。X の指定に従って捻れる。

|: *Transpose* 転置 |: *Y*	ϕ	**右引数のセルを転置させる。** (i.3 4);|: i.3 4 +---------+------+ |0 1 2 3|0 4 8| |4 5 6 7|1 5 9| |8 9 10 11|2 6 10| | |3 7 11| +---------+------+ (|:) (i.2 2 2);|: i. 2 2 2 +---+---+ |0 1|0 4| |2 3|2 6| | | | |4 5|1 5| |6 7|3 7| +---+---+ (|:) $ |: i.3 4 5 5 4 3 **3面4行5列のアレイは|:により5面4行3列のアレイになる。**	*Trans pose* 2項転置 *X*|: *Y*	**X で指定した軸が最も若い軸（0軸）となるよう転置。** 1 |: i.2 3 3 4 5 0 1 2 **X で指定した軸が 0 軸になるように変形されていて、** **（0 1）-> (1 0) のように軸が変換される。** <"2 i.2 3 4 +---------+-----------+ |0 1 2 3|12 13 14 15| |4 5 6 7|16 17 18 19| |8 9 10 11|20 21 22 23| +---------+-----------+ $ 0 |: i.2 3 4 3 4 2 (0 1 2) -> (1 2 0) **のように軸が変換される。** $ 0 1 |: i.2 3 4 4 2 3 (0 1 2) -> (2 0 1) (<0 1) |:i.4 4 0 5 10 15 diag=: (<0 1)&|: **対角要素が取り出される。** (<0 1 2) |: i.2 3 4 0 17 **(0 0 0) と (1 1 1) の要素が取り出される。**
~ *Reflexive* 両側化 副詞 *u* ~ *Y*		二項動詞に ~ を付加すると、*Y* を左右の引数として両側演算を行なう。u~ y **は** y u y と同じ /:~ 1 3 0 5 2 4 0 1 2 3 4 5 NB.up-sort +~3 NB. 3 + 3 6	*Passive* 交換 副詞 *X u* ~ *Y*	*u* ~ は左右の引数を反対（交換）にして演算を行なう。 1 2 3 -~ 4 5 6 3 3 3 4 5 6-1 2 3 3 3 3

`$.` *Sparce-Array* 粗行列 `$. Y`		大型粗行列 *(Sparce Array)* の *0* 以外の数のみをアドレスと *0* 以外の数で表示する。この粗行列の形のままで演算できる。 `Y=.(100+i.4)2 7 8 13}16$0` `Y=. 4 4 $ Y` `0 0 100 0` `0 0 0 101` `102 0 0 0` `0 103 0 0` `$. Y NB. Sparce` `0 2 ┆ 100` `1 3 ┆ 101` `2 0 ┆ 102` `3 1 ┆ 103` `($. Y) -: Y NB.` **一致** `1` `3* $. Y NB. 3` **倍** `0 2 ┆ 300` `1 3 ┆ 303` `2 0 ┆ 306` `3 1 ┆ 309` `$.^:_1(3* $.Y)NB.` **行列形に戻す** `0 0 300 0` `0 0 0 303` `306 0 0 0` `0 309 0 0` 文字列とボックスはサポート外。使えるプリミティブにも一部制限がある。	*Sparse* `n $. Y`	粗行列の要素の表示などを左引数により８種の方法で行う `2 $. Y` `0 1` **軸の表示（0 1 の 2 軸）** `4 $. $. Y` `0 2` `1 3` `2 0` `3 1` **アドレスを表示** `5 $. $. Y` `100 101 102 103` **値を表示** `7 $. $. Y` `4` **元の行列の行** `8 $. $. Y` `0 2 | 100` `1 3 | 101` `2 0 | 102` `3 1 | 103` **粗行列の表示** *(0,1,3* 複雑 *6* 欠番*)*

1.3.3　取り、落し

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
{ *Catalogue* カタログ { Y		要素の組み合わせ（カタログ）を作る { i. 3 3 +-----+-----+-----+ |0 1 2|3 4 5|6 7 8| +-----+-----+-----+ {1 2 3;4 5 6 +---+---+---+ |1 4|1 5|1 6| +---+---+---+ |2 4|2 5|2 6| +---+---+---+ |3 4|3 5|3 6| +---+---+---+ {'japan';'korea' +--+--+--+--+--+ |jk|jo|jr|je|ja| +--+--+--+--+--+ |ak|ao|ar|ae|aa| +--+--+--+--+--+ |pk|po|pr|pe|pa| +--+--+--+--+--+ |ak|ao|ar|ae|aa| +--+--+--+--+--+ |nk|no|nr|ne|na| +--+--+--+--+--+ {@> i.2 3 +-+-+-+ |0|1|2| +-+-+-+ |3|4|5| +-+-+-+ <"0 i.2 3 **でも同じ結果**	{ *From* 選択 X{Y	左引数で指定したアイテムを取り出す。 (i. 3 4); 1 2 { i. 3 4 +---------+---------+ |0 1 2 3|4 5 6 7| |4 5 6 7|8 9 10 11| |8 9 10 11| | +---------+---------+ 0{i.3 4 0 1 2 3 **0 行の取り出し** (i.3 4); 1 2 { "1 i.3 4 +---------+----+ |0 1 2 3|1 2| |4 5 6 7|5 6| |8 9 10 11|9 10| +---------+----+ **1，2 列の取り出し** (i.3 4); (<1 2;0 2){ i.3 4 +---------+----+ |0 1 2 3|4 6| |4 5 6 7|8 10| |8 9 10 11| | +---------+----+ **ブロックで取り出すときは 2 重ボックスのアドレスを左引数で指定する。** <1 2;0 2 +---------+ |+---+---+| ||1 2|0 2|| |+---+---+| +---------+ *1,2* 行の *0,2* 列のセルを指定

{. *Head* 先頭取り {. Y		A=: 0 1 2 3 4 {. A 0 **右引数の先頭アイテム（要素）を取り出す。** {. B=.i.3 4 0 1 2 3 **先頭アイテム（行）の取り出し** {. C=.i.2 3 4 0 1 2 3 4 5 6 7 8 9 10 11 **先頭のアイテム（テーブル）の取り出し。**	*Take* 取り X {.Y	*X* が正整数のときは、*Y* の先頭から *X* 個のアイテムをまた負なら末尾から |*X*| 個のアイテムを取り出す。 A=: 0 1 2 3 4 2 {. A 0 1 _3 {. A 2 3 4 (i.3 4); 2 3{. i.3 4 +---------+-----+ |0 1 2 3|0 1 2| |4 5 6 7|4 5 6| |8 9 10 11| | +---------+-----+ **2 行 3 列のアイテムの取出し** 6 {. A 0 1 2 3 4 0 $ 6 {. 'JAPLA' JAPLA $ 6 {. 'JAPLA' 6 **X が要素数を超えると数値は 0 文字列はブランクが付加される**
Tail 末尾取り {: *Y*		**右引数の最終アイテム（要素）を取る** {: 0 1 2 3 4 4 {: i.3 4 8 9 10 11 **最終アイテム（行）を取る**	（無し）	

}. *Behead* 先頭落し }. *Y*		A=:0 1 2 3 4 }. A 1 2 3 4 **先頭アイテム（要素）を落す。** }. B=.i.3 4 4 5 6 7 8 9 10 11 **先頭のアイテム（行）を落す。** }. C=.i.2 3 4 12 13 14 15 16 17 18 19 20 21 22 23 **先頭のアイテム（テーブル）を落す。**	*Drop* 落し *X* }. *Y*	*X* が正整数のときは、*Y* の先頭から *X* 個のアイテムをまた負なら末尾から |*X*| 個のアイテムを落す。 2 }. 1 2 3 4 8 9 10 11 _2 }. B 0 1 2 3 1 2 }. i.3 4 6 7 10 11 **先頭から 1 行 2 列を落す。** ind=.-.(i.4)e. 0 2 0 1 0 1 NB. 0 2 **列を落す** (i.3 4); ind #"1 i. 3 4 +---------+----+ |0 1 2 3|1 3| |4 5 6 7|5 7| |8 9 10 11|9 11| +---------+----+ **指定した行や列の落しは落す指標（0 1）を作成し** copy(#) **で行う**
}: *Curtail* 末尾落し }: *Y*		**右引数の最終アイテム（要素）を落す** }: 0 1 2 3 4 0 1 2 3 }: i.3 4 0 1 2 3 4 5 6 7 **最終アイテム（行）を落す**	（無し）	

1.3.4 修正（ アメンド）

J では、修正（アメンド）を行っても *Y* 自体は変更されない。*Y* を書換えるには次のように=: =. を用いて顕に定義する必要がある。

```
Y=.1 0 1 0 1 } Y
```

APL の場合には、← で内容が直ちに修正される。

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
} *Item-Amend* アイテム修正 m } Y		*m* で指定した *Y* の各アイテムを取り出して表示する。 2 1 0 2 }i. 3 4 8 5 2 11 i.3 4 0 1 2 3 NB. 0 4 5 6 7 NB. 1 8 9 10 11 NB. 2 ---------- 2 1 0 2 NB. **各 m は行を指定** 0 1 2 3 **列の順に取り出す行を指定** Y=.'japan','korea',:'china' japan korea china 1 0 1 2 0 } Y karnn (i.# Y) 番目を順に行を指定して取り出す	*Amend* 修正 X m }Y	*Y* のアイテム（要素）を *X* に変更して表示する。*m* は変更するアドレス ◊ リストを変更 '*' 1 3 5 7}'abcdefghij' a*c*e*g*ij 'BD' 1 3 } 'abcd' aBcD ◊ 配列のポイントを変更 m=.0 1;1 2;2 3 NB.index +---+---+---+ |0 1|1 2|2 3| +---+---+---+ (120 130 140)m}i.3 4 +---------+-------------+ |0 1 2 3|0 120 2 3| |4 5 6 7|4 5 130 7| |8 9 10 11|8 9 10 140| +---------+-------------+ i.3 4 amend

◇ 配列の行や列を変更

```
(100+i.2 4)(<<0 2)} A
100 101 102 103
  4   5   6   7
104 105 106 107
```

0,2 行を修正

```
(100+i.3 2)(<<0 2)}"1 A
100 1 101  3 NB.rank("1)
102 5 103  7
104 9 105 11
```

0,2 列を修正

◇ *2* 重ボックスで配列のブロックを変更

```
   (<0 1;1 2)
+---------+
|+---+---+|
||0 1|1 2||
|+---+---+|
+---------+

  120 130(<0 1;1 2)}i.3 4
0 120 130  3
4 120 130  7
8   9  10 11
```

0,1 行の 1,2 列の個所を修正（2 重ボックスのアドレス）

1.3.5 並べ替え (ソート）ー指標と操作

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
/: *Grade - up* 昇順 /: Y	↑	**Y を昇順に並べ替えるインデクスを返す。** /:Y=. 23 11 13 31 12 1 4 2 0 3 **Y を取り出す順を示す。最小値はインデクス** 1 **に、最大値はインデクス** 3 **にある。** 1 4 2 0 3{Y 11 12 13 23 31 /:˜ Y NB. **ソートを実行(Y/:Y)** 11 12 13 23 31 Y2=. 3 4 $ 12?.60 6 47 13 46 34 44 8 37 5 59 56 32 /: Y2 2 0 1 NB. **列を取り出す順** /:˜ Y2 5 59 56 32 6 47 13 46 34 44 8 37 テーブルのときは *0* 行の昇順の指標。 /: 'ASSOCIATION' 0 6 4 5 8 10 3 9 1 2 7 /:˜'ASSOCIATION' AACIINOOSST **文字の場合はアルファベット順。**	*Sort* 昇順ソート X /: Y	*X* を *Y* で与えた指標に従って（昇順に）並べ替える。 74 33 66 /: 2 1 0 66 33 74 2 1 0 { 74 33 66 66 33 74 Y=.23 11 13 31 12 Y /: Y 11 12 13 23 31 /:˜ Y 11 12 13 23 31 (/:Y){ Y 11 12 13 23 31 **いずれも昇順に並べ替える** 'JAPAN APL ASSOCIATION' /:˜'JAPAN APL ASSOCIATION' AAAAACIIJLNNOOPPSST A=.12 3 5,0 5 6,: 12 4 7 A;(/: A);/:˜ A +------+-----+------+ |12 3 5|1 0 2| 0 5 6| | 0 5 6| |12 3 5| |12 4 7| |12 4 7| +------+-----+------+ A index sort **同じ数があれば次列（以降）の当該箇所を昇順に並べる**

\: *Grade down* 降順 \: *Y*	↓	**Y を降順に並べ替えるインデクス を返す。** ＼: 23 11 13 31 12 3 0 2 4 1 **最大値はインデクス３に、最小値はインデクス** 1 **にある。** A=.s: **’山川谷川川田太田 相生’** /:˜ A **‘太田‘山川‘川田‘相生‘谷川** \:˜ A **‘谷川‘相生‘川田‘山川‘太田** **（漢字は複雑）**	*Sort* 降順ソート X ＼: Y	*X* を *Y* で与えた指標に従って *(*降順に）並べ替える。 74 33 66 \: 2 1 0 74 33 66 \:〜Y=.23 11 13 31 12 31 23 13 12 11 \:˜’JAPAN APL ASSOCIATION’ TSSPPOONNLJIICAAAAA Y 74 1 26 23 33 87 34 10 66 50 20 45 \: Y 0 2 1\: Y 74 1 26 23 66 50 20 45 33 87 34 10

1.4　ボックスと分割（パーティション）

J ではボックス（*BOX*）は数値、文字列と並ぶ第三の型として導入された。ボックスを用いて次のような操作ができる。ボックスを分割（パーティション）するいろいろなプリミティブがある。

- *From,Amend* の指標
- 数値と文字の混合配列（構成 · 表示）
- 数値の並列演算
- *nested array(*配列の入れ子）
- 関数の並列演算
- *plot* のデータ

Ｊでは、元来ボックスのままでは演算しないので、ボックスを開いて演算する必要がある。ボックスの状態で演算させるにはレベル `(L:0)` 等を使用する

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
< *Box* `< Y`		**右引数をボックスで囲む。** `     < i.3 3` `+-----+` `|0 1 2|` `|3 4 5|` `|6 7 8|` `+-----+`	*Less than* 小さい *X < Y* *(1.2.1)*	
> *Open* `> Y`		**ボックスを開く。** `  >  < i.3 3` `0 1 2` `3 4 5` `6 7 8` `  >:&.>  <i.3 4` `+----------+` `|1  2  3  4|` `|5  6  7  8|` `|9 10 11 12|` `+----------+` 一度開いて *1* を加え再度閉じている	*Larger than* 大きい *X > Y* *(1.2.1)*	

;
Raze
ほぐし
; Y

主軸に沿って右引数をほぐす

```
    ; i.2 3
 0 1 2 3 4 5
    ;/ i.2 3
+-----+-----+
|0 1 2|3 4 5|
+-----+-----+
 ] A=. <"0 i.3 4
+-+-+--+--+
|0|1|2 |3 |
+-+-+--+--+
|4|5|6 |7 |
+-+-+--+--+
|8|9|10|11|
+-+-+--+--+
   ;("1) A
0 1  2  3
4 5  6  7
8 9 10 11
```

配列の形を保ったほぐしには ;"1 **や** ;"2 が有効

Link
結合
X;Y

左右の引数をボックスの形で結合する。

```
   0 2 4 6;1 3 5 7
+-------+-------+
|0 2 4 6|1 3 5 7|
+-------+-------+
```

(<X),(<Y) と同じ

◇ 数値と文字列の結合 *(*混合配列*)*
APL ではシームレスになっている。

```
     0 2 4 6;'abcdef'
+-------+------+
|0 2 4 6|abcdef|
+-------+------+
 ; (0 2 4 6); 'abcdef'
 domain error
```

数と文字の混合はほぐせず

◇ 入れ子の配列（ネスティッドアレー）を作成する

```
   (<0 2,: 4 6);1 3 5 7
+-----+-------+
|+---+|1 3 5 7|
||0 2||       |
||4 6||       |
|+---+|       |
+-----+-------+
```

L. *Level* レベル 動詞 L.Y		右引数のボックスのレベルのの深さを与える。*Y* が空又はオープン形のときは0である `    L. i.5` `0` `    L. 1 2;3 4` `1` `  (<0 2,: 4 6);1 3 5 7` `+-----+-------+` `|+---+|1 3 5 7|` `||0 2||       |` `||4 6||       |` `|+---+|       |` `+-----+-------+` `  L. (<0 2,: 4 6);1 3 5 7` `2`	無し	
L:			*Level-At* レベル 接続詞 u L:n Y	動詞 *u* を *L:n* というレベル *(深度)* のボックス内で演算する `   +/(L:0)1 2 3;3 4 5` `+-+--+` `|6|12|` `+-+--+` `  Y=.(<0 2,:4 6);1 3 5 7` `+-----+-------+` `|+---+|1 3 5 7|` `||0 2||       |` `||4 6||       |` `|+---+|       |` `+-----+-------+` `  ((+/ L:0);(+/ L:1))Y` `+----------+-----+--+` `|+-----+--+|+---+|16|` `||+---+|16|||0 2||  |` `|||4 8||  |||4 6||  |` `||+---+|  ||+---+|  |` `|+-----+--+|     |  |` `+----------+-----+--+` `  L:0         L:1`

{:: *Map* マップ 動詞 {:: Y		**ボックスへのパスを表示** Y=. 2 3;(2;3);i.2 3 +---+-----+-----+ |2 3|+-+-+|0 1 2| | ||2|3||3 4 5| | |+-+-+| | +---+-----+-----+ {:: Y +---+-------------+---+ |+-+|+-----+-----+|+-+| ||0|||+-+-+|+-+-+|||2|| |+-+|||1|0|||1|1|||+-+| | ||+-+-+|+-+-+|| | | |+-----+-----+| | +---+-------------+---+	*Fetch* フェッチ X{:: Y	左引数 *(X)* で与えたインデクスに従い右引数 *(Y)* のサブアレイの要素を取り出す。 (2; 1 2){::Y 5 (1;1 2){:: 5;i.2 3 4 +-+-----------+ |5| 0 1 2 3| | | 4 5 6 7| | | 8 9 10 11| | | | | |12 13 14 15| | |16 17 18 19| | |20 21 22 23| +-+-----------+ (1;1 2){:: 5;i.2 3 4 20 21 22 23 **サブアレイ（1）のテーブル 1 の第 2 行を取り出す** 0{:: 5;i.2 3 4 5 **サブアレイ 0 の取り出し**
/ *Insert* 挿入 *u*/*Y* *m*/*Y* （副詞）		動詞 *(u)* の場合は、アイテム間に *u* を挿入した演算と同等 ジェランド (u‘v) の場合は、*u,v* の各要素を右引数の各アイテムに指定順に挿入する。 +/ i.5 10 NB. (0+1+2+3+4) */ >:i.5 120 NB. **パイ関数** 1*2*3*4*5 Y;(+/ Y);+/ "1 Y=.i.2 3 +-----+-----+----+ |0 1 2|3 5 7|3 12| |3 4 5| | | +-----+-----+----+ **テーブル等の一般アレイに作用させるときは** **ランクで方向を指定する**	*Table* 一般外積 X u Y/ *-1.5*	(>:i.5) */ table >:i.5 +--+-----------+ |*/|1 2 3 4 5| +--+-----------+ |1 |1 2 3 4 5| |2 |2 4 6 8 10| |3 |3 6 9 12 15| |4 |4 8 12 16 20| |5 |5 10 15 20 25| +--+-----------+ *table* は見出しを付ける *J* の組込関数

\ *Prefix* 前から *v* \ *Y* （副詞）		動詞 *u* を 1 からアイテム数（*#Y*）まで逐次各アイテムに作用させる `   <\1 2 3` `+-+---+-----+` `|1|1 2|1 2 3| RunningSum` `+-+---+-----+` `  <\i. 3 3` `+-----+-----+-----+` `|0 1 2|0 1 2|0 1 2|` `|     |3 4 5|3 4 5|` `|     |     |6 7 8|` `+-----+-----+-----+` ◊ *BOX* の代わりに動詞を置くことができる。この場合一々開いたり *L:0* は不要 `   +/\ 1 2 3 4 5 NB.sum scan` `1 3 6 10 15` `  */\ 1 2 3 4 5NB.times scan` `1 2 6 24 120` `   */ 1 2 3 4 5` `120` `   */\ 5#2` `2 4 8 16 32` `2^>:i.5` `2 4 8 16 32`	\ *Infix* 中から副詞 *u m*\ *Y*	左引数を区切りとして前から逐次取り出し作用させる *X* が正ならオーバーラップするが、負のときは重複しない。 `   3<\ 1 2 3 4 5` `+-----+-----+-----+` `|1 2 3|2 3 4|3 4 5|` `+-----+-----+-----+` `   _3<\ i.9` `+-----+-----+-----+` `|0 1 2|3 4 5|6 7 8|` `+-----+-----+-----+` ◊ *BOX* の代わりに動詞を置くことができる。例は移動平均 `   3+/\ 1 2 3 4 5` `6 9 12` `  3 (mav=:+/\%[) i.10` `1 2 3 4 5 6 7 8`
\. *Surffix* 後から *u*\. *Y* （副詞）		*1* からアイテム数 *(#Y)* まで後ろを取って（前から逐次減少させてた）組み合わせを作る `   <\. 1 2 3` `+-----+---+-+` `|1 2 3|2 3|3|` `+-----+---+-+` `Reverse Running Sum` `   +/\. 1 2 3` `6 5 3`	*Outfix* 外から *X u*\.*Y*	**|X|で指定した数を外側から逐次取った組み合わせに u を作用させる** `   3<\. 1 2 3 4 5 6` `+-----+-----+-----+-----+` `|4 5 6|1 5 6|1 2 6|1 2 3|` `+-----+-----+-----+-----+` `   3+/\. 1 2 3 4 5 6` `15 12 9 6`

/. Oblique 斜め *u /. Y m /. Y (*接続詞*)*		動詞 *(u)* の場合は、テーブルの形の右引数に対して右上から左下に斜めに作用させる。 ジェランド *(m)* の場合は、*m* の各要素を右引数の各アイテムに逐次作用させる。 `   i.3 4` `0 1  2  3` `4 5  6  7` `8 9 10 11` `   +//. i.3 4` `0 5 15 18 17 11` 以下の斜めの和 ◇ ↙ ↙ ↙ ↙ ↙ ↙ ↙ ↙ ↙ ↙ ◇	*Key* キー *X u/.Y X m/.Y*	動詞 *(u)* の場合は、*X* をキーとして右引数の要素をグループ別に一縛りにし配分する。 ジェランド *(m)* の場合は、*m* の各要素を逐次適用する *X=.1 2 3 1 3 2 1* `    X=.1 2 3 1 3 2 1` `   X,:i.7` `1 2 3 1 3 2 1 NB. X` `0 1 2 3 4 5 6 NB. i.7` `   X </.i.7` `+-----+---+---+` `|0 3 6|1 5|2 4|` `+-----+---+---+` **X(キー）の同じものがまとめられて BOX で区分される** `  X +//. i.7` ` 9 6 6` **X をキーとして BOX 毎に合計** `    X (<@:+:)/.i.7` `+------+----+---+` `|0 6 12|2 10|4 8|` `+------+----+---+` **X をキーとして BOX 毎に 2 倍**

;. *Cut* 区切り *(*接 続詞*)* u ;.nY *(n* はカットの型）

右引数 *Y* を（先頭または末尾のアイテムを区切り文字として）フレットで区切る。*n* はカットの型

```
   <;.3 i. 4
+-------+-----+---+-+
|0 1 2 3|1 2 3|2 3|3|
+-------+-----+---+-+
   <;.3 D=.i. 2 3
+---+---+-+
|0 1|1 2|2|
|3 4|4 5|5|
+---+---+-+
|3 4|4 5|5|
+---+---+-+
   <;._3 i. 2 3
+---+---+
|0 1|1 2|
|3 4|4 5|
+---+---+
   <;.3 'JAPLA'
+-----+----+---+--+-+
|JAPLA|APLA|PLA|LA|A|
+-----+----+---+--+-+

    ];.3 i.3
 0 1 2
 1 2 0
 2 0 0
```

モザイク模様

<;.3 y*(*リスト*)* は (#y)<;.3 y と同じ演算。
<;.3 D(2 2 **のテーブル**) は 2 2 <;.3 D と同じ演算
;._3 D についても同様

Cut 区切り *(*接 続詞*)* *X* *u*;.*n* *Y*

左引数 *(X)* と型指定 *(n)* により *Y* を区切る

◇ *n=1,2* のときはブール数の 1 の個所でカットする

```
 ind=.1 0 0 1;1 0 0 1 0
 ind <;.1 i. 4 5
+--------+-----+
| 0  1  2| 3  4|
| 5  6  7| 8  9|
|10 11 12|13 14|
+--------+-----+
|15 16 17|18 19|
+--------+-----+
```

~:で 1 0 の指標作成

```
  ind=. ~: a=: 3 3 4 4 4 5 5
1 0 1 0 0 1 0
 ind <;.1 a=.3 3 4 4 4 5 5
+---+-----+---+
|3 3|4 4 4|5 5|
+---+-----+---+
```

◇ *_1* は<;.1 の先頭を削除する

```
(~:a)<;._1 a=.3 3 4 4 4 5 5
+-+---+-+
|3|4 4|5|
+-+---+-+
```

◇<;.2 は末尾から 1 が現れるごとに区切って出力。

```
(|.~:a)<;.2 a=.3 3 4 4 4 5 5
+---+-----+---+
|3 3|4 4 4|5 5|
+---+-----+---+
```

◇*n=3* は左引数の形でカット

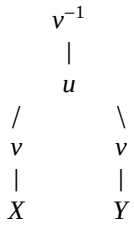
```
   1 2<;.3 i.3 2
+---+-+
|0 1|1|
+---+-+
|2 3|3|
+---+-+
|4 5|5|
+---+-+
```

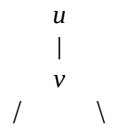
1.5 動詞（関数）の合成

動詞（関数）を合成するため接続詞や副詞またフォークやフックの機能がある。

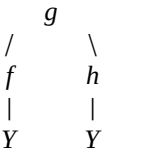
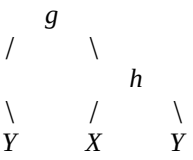
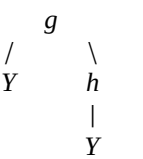
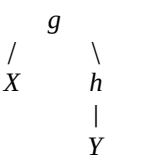
1.5.1 接続詞 ボンドとアットプ

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
& Bond ボンド （接続詞） u&n Y n&v Y *Compose* u&v Y		動詞と数 *(*名詞*)*、又は 数と動詞を接続して新たな動詞を作る。 さらに *2* つの動詞を連結し動詞を作る。 *2* つの動詞が単項ときは u@v と同じ働きをする。 *u* | *v* | *Y* `   ^&3 Y=.>:i.5` `1 8 27 64 125` **NB. Y の 3 乗** `   3&^ Y=.>:i.5` `3 9 27 81 243` **3 の Y 乗** `    %&2 Y=.1 2 3 4` ` 0.5 1 1.5 2` **NB. Y を 2 で割る** `    2&% Y` ` 2 1 0.6666667 0.5` **(2 を Y で割っている)** `  +:&>: 5` **NB. 2*(1+5)** `12` ◊ 単項の@は同じ働き） `  +:@>: 5` `12`	*Compose* コンポーズ X u&v Y	二項の動詞 *u* と単項の動詞 *v* を &連結した場合は、その働きに注意が必要。 つまり x u&v y は (vx)u(vy) のように作用する。 *u* / \ *v* *v* | | *X* *Y* `  100 -&+: 40` `120` `  (+:100)-(+:40)` `120`

&. *Under* アンダー *(*接続詞*)* u&.v Y		u&. v は、u&v の後に *v* の逆演算を行なった結果を返す。  +:&.>: 5 11 <:+:&>: 5 11 (>:^:_1)&+:&>: 5 11 >:^:_1 <: **「>：」の逆演算は「<：」** <"0 i.5 +-+-+-+-+-+ |0|1|2|3|4| +-+-+-+-+-+ +:&.> <"0 i.5 NB. 2 **倍** +-+-+-+-+-+ |0|2|4|6|8| +-+-+-+-+-+ each =.&.> BOX **を開いて動詞** u **を適用して戻す**	*Under* アンダー X u&.v Y	&で結合させた演算の後で右動詞 *v* の逆演算を行なった結果を返す。  100 -&.+: 40 60 -:(+:100)-(+:40) 60 +:^:_1(+:100)-(+:40) 60 $\frac{(100\times 2)-(40\times 2)}{2}=60$ の計算
&: *Appose* アポーズ u&:v Y		ランクが無限であることを除き &と同じ働きをする。 @:参照 +:&:>: 5 12 +/&:>:i.5 15 +/&>:i.5 1 2 3 4 5	*Appose* アポーズ X u&:v Y	ランクが無限の場合で&とほぼ同じ働き 100 -&:+: 40 120 (+:100)-(+:40) 120

@ *Atop* アトップ (接続詞) u@v Y		*2* つの動詞を関数合成により連結して動詞を作る。*2* つの動詞が単項ならば u&v と同じ働きをする。 +:&>: 5 12 +:@>: 5 12 %&2 i.5 0 0.5 1 1.5 2 %@2 i.5 domain error **「@」は「&」と異なり、名詞と数値の連結が不可能である。**	*Atop* アトップ (接続詞) X u@v Y	単項の動詞 *u* と二項の動詞 *v* を @で連結した場合は、その働きに注意が必要。つまり X(u@v)Y は u(XvY) のように作用する。  3 |@- 7 4 |(3-7) 4
@: *At* アット (接続詞) u@:v Y		ランクが無限であることを除き「@」と同じ働きをする。 +:@:>: 5 12 「@」は接続される直前の右のランクを継承するが「@:」は継承しない。 ◇ +/ には +/@:を用いる。 +/@:>: i.5 15 +/@>: i.5 1 2 3 4 5 (@でうまく作動しない場合は@:に変えてみるとよい場合がある。)	*At* アット (接続詞) X u@:v Y	@とほぼ同じ働き 100 +:@:- 40 120 3 |@:- 7 4 3 |&:- 7 _1 (-3)|(-7) _1

1.5.2 フックとフォーク

用法	APL	結果と例（単項）	用法	結果と例（両項）
Fork フォーク *(*3 連動詞*)* *(f g h)Y*		Ｊで動詞が連結されたときは3つの動詞の結合 *(*フォーク*)* が最優先される。ただ *2* つだけの場合はフックになる。  `(+/%#)Y=.1 2 3 4 5` `3 NB. 平均` `mean=: +`	*Fork* フォーク *X(f g h)Y*	 `5 (+*-) 3` `16` `(5+3)*(5-3)` `16` `3 (+,-) 5 NB. plus minus` `8 _2` ４つ以上の連結の場合には `k f g h` `└──┘ Fork` `└────┘ Hook` 右から *3* 個ずつの組み合わせを行ってフォークを作る。最後が *2* 個になればフックになる
Hook フック *(2* 連動詞*)* *(gh)Y*		二項動詞 *g* と単項動詞 *h* では `(g h )Y` **は** `Y g h(Y)` のように演算する。  `(*-)3` `_9` `3 *(-3)` `_9` `-(+/%#)` `Y=. 1 2 3 4 5` `(- +/%#)Y` `_2 _1 0 1 2` $x - \bar{x}$ 偏差	*Hook* フック *X(gh)Y*	単項の場合と同様で `X(g h)Y` は `X gh(Y)` と演算する。  `5(*-)3` `_15` `5 *(-3)` `_15`

[: *Cap* キャップ （動詞） ([:gh)Y		*2* 連動詞の左端の単項動詞をフックではなく（右から順に）働かせたいときにその左端に付けて、「([: g h)Y」又は X([: g h)Y の形で演算させ g{h(Y)}, g(X h Y) と同じ結果を与える。 *Monad*/片側 | *Dy* *ad* 両側 *g* | *g* | | *h* *h* | / \ | *Y* | *X* *Y* ([:*-)3 **片側の場合** _1 *(-3) _1 5([:*-)3 **両側の場合** 1 *(5-3) 1	*Cap* キャップ X[:Y	単項又は二項関数の意図しない用法を排除する。 abs=.| : [: abs _4 0 5 4 0 5 3 | _4 0 5 4 0 5 3 abs _4 0 5 |domain error: abs | 3 abs _4 0 5

1.5.3 ジェランドとコントロール機能

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
~			*Evoke* 呼出し *(*接続詞*)* *'m' ~ Y*	ユーザー定義の動詞 *(*代動詞*)* を名前で呼び出してその動詞を実行させる。 `'m'~ y` **は** `m y` と同じ結果を与える。 `sum=.+/` `'sum'〜 i.5` `10`
‘ *Tie* *(*接続詞*)* *u‘v/* *u‘v/.* *u‘m@.* *n ‘v@.* *u‘v@.*		複数個の動詞を連結して動名詞 *(*ジェランド*)* を作る。 スラッシュ *(/)* やオブリーク *(/.)* アジェンダ *(@.)* と共に用いる `+‘*/ i.6` `29` `0+1*2+3*4+5` `29` `+‘%/ 3 1 4` `3.25` `3+1%4` `3.25` **演算子を交互に挿入する** `+:‘*/. _2 3` `_4` `1` `!‘*: /. 3 4 5` `6` `16` `120` **演算子を交互に作用させる。** `/.` **を用いて順に** `(!3),(*:4),(!5)` **を求めている**	‘: *Evoke-Gerund* `m‘:0` `m‘:3` `m‘:6`	次の *3* タイプがあり、右引数により指定する。 *m ‘:0* | *Append* *m ‘:3* | *Insert* *m ‘:6* | *Train* `(+:‘-:‘*:‘%:)‘:(0) i.5` `+: 0 2 4 6 8` `-: 0 0.5 1 1.5 2` `*: 0 1 4 9 16` `%: 0 1 1.41421 1.73205 2` `Append` **（** `0` **）では演算子を逐一作用させる** `+‘* ‘:(3) i.5` `14` `Insert` **は** `(3)` **では交互に演算する** `+‘*/ i.5` **と同じ** `+/‘%‘# ‘:(6) i.5` `2` `(+/%#)i.5 NB. mean` `2` `Train(6)` **はトレインとして実行される**

@. *Agenda* アジェンダ m@.v Y		関数型の条件文でジェランド *m* を実行させるときの条件を記述する a,: 2&| a=: 2 3 4 5 6 7 2 3 4 5 6 7 NB. a 0 1 0 1 0 1 NB. 2&| **条件文＝偶数奇数の判別** cond=. +:'-: @.(2&|) cond "0 a 4 1.5 8 2.5 12 3.5 **条件文により 2 倍（+:）と半分（-:）を選択して実行。** **条件文は () に入れる。ランク"0 も必要**	*Agenda* アジェンダ X m@.v Y	**片側形の場合と同様** 2 +'^ @. <3 8 2<**3 は真** (1) **なので、~ の方** (2^3) **を実行する** 3 +'^@.< 2 5 (**3** +2) **の方を実行する**
^: *Power* 反復 （接続詞） u^:**n** Y		動詞 *u* を *n* 回実行する。*n* は一般アレイでも構わない。タシット型のループ >: ^:(3) i.6 3 4 5 6 7 8 >: ^:(i.3) i.6 0 1 2 3 4 5 NB. 0 times 1 2 3 4 5 6 NB. 1 times 2 3 4 5 6 7 NB. 2 times **反復を** (i.n) **とすると経過を観察できる** %:^:_1 i.5 0 1 4 9 16 ◊ u ^:_1 は *u* の逆演算である。**「%:」は「*:」***(2* 乗）と同じである。 ◊ ^:_は収束するまで反復する *(*無限ループに注意） ◊ 関数 *f* 作成し**「f ^:(100)」**などと使用できる。	*Power* 反復 Xu^:**n** Y	**片側形と同じく反復演算** 3 *^:3 i.5 0 27 54 81 108 $(3\times3\times3)\times$ *0 1 2 3 4* 3 *^:(i.5) 3 3 9 27 81 243 $3\times3\times3\times3 \rightarrow 81$

Adverse （接続詞） `u::v`		`u::v` の結果はエラーが無ければ *u* を、エラーのときは停止せずに *v* を実行する。 *cf. try catch*		
M.			*M.* *Memo* 記憶	反復計算で関数と結果をメモリーに保存して効率を高める。（早くなる場合があるが効かない場合もある） *(*詳細は *2* 章）
`$:` *Self- Reference* 再帰		`$:`を用いた再帰 `1:‘(] * $:@<:)@.*5` `120` `*/ >:i.5` `120` 詳細は *2* 章		

1.6　定義と実行

1.6.1　定義と引数

同じ名前の名詞や動詞を後から定義したり、ロードした場合に関数名が重複（オーバーラップ）した際には、後の定義が局所定義の場合は影響を受けないが、大局定義の場合には以前の定義が警告なしに変更され、後の方が用いられるので注意を要する。。

大きなプロジクトの場合はロケールやクラスファイルを用いて定義を区分けして整理する。

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
=. =:	←		*Copula (is)* 連結詞 *(*名詞*) X =. Y* *X =: Y*	「=.」は局所 *(local)* 定義 「=:」は大局 *(global)* 定義 名詞には数値や文字等を挿入する。 動詞のときは処理を記述する `A=. 123` `'A B C'=: 123 256 768` **A は** 123 **B は** 256 **C は** 768 **と定義される** `plus=: +` `pm=: +,-`
x y *m n* *u v* 代用値		スクリプトの明示型で引数の代用とし用いる *x* 右引数 *y* 左引数 *m* 数値の左引数 *n* 数値の右引数 *u* 動詞の左引数 *v* 動詞の右引数 *u,v* は主として副詞に用いる。*m,n* は *u,v* と数値とを引用する接続詞に用いる事が多い。 *Ver6* で *x.y.m.n.u.v.* が廃止された。スクリプトの中で名詞の定義にに *x y m n u v*（小文字）を使ってあると衝突する 使用法は第 *6* 章参照		

NB. *Comment* コメント		**コメント** `  mean=.+/%# NB.mean` `  mean 1 2 3 4 5` `3` **NB. 以降 行のは演算に関係しない**		
[(]) *Same* 同等 （名詞）		**定義内容を実行表示させる。** `  ] n=.i.2 3` `0 1 2` `3 4 5`	*Left* */Right* 右引用 左引用	**「[」は左引数を取り出す** **「]」は右引数を取り出す** `   2 3 [ 4 5` `2 3` `   2 3 ] 4 5` `4 5`
: *Monad* */Dyad* 単項／二項 （接続詞）		「*(*単項*)*：*(*二項*)*」の形で定義し、左引数の有無によって単項又は二項として演算する `  log=.10&^. : ^.` `  log 10 100` `1 2` `   1x1 log 10 100` `2.30259 4.60517` `   ^. 10 100` `2.30259 4.60517`		

f. 無し			*Fix* 動詞の固定 （副詞）	動詞の定義を固定し、その後の変更の影響を排除する `   sum=. +/` `   mean1=. sum % #` `   mean=. mean1 f.` `   mean >:i.5` `3` `   sum=. -/` `   mean1 >:i.5` `0.6` `   mean >:i.5` `3` **後の** sum **の変更でも** mean **は不変**
!.			*Fit* *Customize* フィットカストマイズ （接続詞）	様々な処理に使い分けられる ◇ 許容誤差 `  (^!._1)~ i.5` `1 1 2 6 24` `  ([:*/]+_1:*i.)"0 i.5` `1 1 2 6 24` `  (^!.1)~ i.5` `1 1 6 60 840` `  ([:*/]+i.)"0 i.5` `1 1 6 60 840` ◇ *!.* で指定した数値や文字列を挿入する `1 0 1 1 0 1#^:_1(!.100)]i.4` `0 100 1 2 100 3` `3j1 0 2 0 3 # !.'*''JAPLA'` `JJJ*PPAAA` ◇ *stope(*海岸段丘*)* を作成し計算する `   5^!.0.1 ] 4 NB.4` **段** `702.78` `  */ 5.0 5.1 5.2 5.3` `702.78`

.. 無し			*Even* （接続詞） *u .. v*	２つの動詞を接続して偶数項だけを取り出す。 `    1 2 3 4 5&p. 2` `129` $f(x) = 1 + 2x + 3x^2 + 4x^3 + 5x^4$ の $f(2)$ の値 `    1 2 3 4 5&p. ..- 2` `93` `    1 0 3 0 5&p. 2` `93` 上の式の偶数個鵜だけの式 $g(x) = 1 + 3x^2 + 5x^4$ での $g(x) = 2$ の値
.: 無し			*Odd* （接続詞） `u .: v`	２つの動詞を接続して奇数項だけ取り出す `1 2 3 4 5 &p. .:  - 2` `36` `  0 2 0 4 0 &p. 2` `36` *Even* の式の奇数項だけの式 $h(x) = 2x + 4x^3$ での $h(2)$ の値

S:（大文字） *Spread* 展開 S: Y		タシット定義の内容を表示す `reg=: %. 1&,.@]` `5!:2 <'reg'` `+--+--------------+` `|%.|+--------+-+-+|` `|  ||+-+-+--+|@|]||` `|  |||1|&|,.|| | ||` `|  ||+-+-+--+| | ||` `|  |+--------+-+-+|` `+--+--------------+` `<S:(0) 5!:2 <'reg'` `+--+-+-+--+-+-+` `|%.|1|&|,.|@|]|` `+--+-+-+--+-+-+`		
:. 無し			*Obverse* オブバース *(裏)* 逆定義 （接続詞）	**逆定義が正しく定義されたかどうかを確かめる。** `f=.*: :. %:` `g=.*: :. +:` `f i.5` `0 1 4 9 16  NB. x^2` `g i.5` `0 1 4 9 16` **先の動詞だけが実行される** `f ^:_1 f i.5  NB. OK` `0 1 2 3 4` **逆関数が正しく定義されていれば右引数の値を返す。** `g ^:_1 g i.5` `0 2 8 18 32` **「`+: *: y`」の演算結果から `g` は正しく定義されていない。**

1.6.2 書式/数値と文字の変換

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
": *Default Format* デフォルト書式 ": Y		数値で与えた右引数を文字化する。 `  ]y=.":i.2 5` `0 1 2 3 4` `5 6 7 8 9` `  $ y` `2 9` **文字化されている！**	*Format* 書式 `m ":Y`	左引数で書式を整え文字化して表示整数部で全体の桁数、小数部で小数以下の桁数を示す *(Ver 6* 以降*)* 左引数 *(m)* は「*m.n]* から「*mjn*」に変更された（互換無し） 例えば *6.2* から *6j2* へ `   6j2 ": 1 2 3 %3` `0.33  0.67  1.00` `   3j2 ": 1 2 3 %3` `*********` **スペースが足りないときは表示しない**
". Do 実行 *". Y*		文字列で与えた右引数を実行し数値化する。 `  a=.'3+4+5'` `  ". a` `12` *CSV* ファイルで読み込んだデータは *BOX* に入った文字なので開いて数値化する。 `".@> DATA`	*Numbers X ". Y*	右引数 *Y* を数値化したときに欠けた部分を左引数で補う。 `    y=.'1 2 3','4 5',:''` `    10 ". y` `  1  2  3` `  4  5 10` `10 10 10`

b. *Basic Charac terristics (*動詞*)* u b .n		◊ u b. 0 は動詞 *u* のランクを表示する。 *: b.0 0 0 0 **(単項),(二項左),(二項右)** ◊ u b. _1 は動詞の逆関数を表示 *: b._1 %: **(逆関数の表示)** ◊ u b. 1 はプリミティブの定義を表示 * b. 1 1 $~ }.@$ **(プリミティブの原始定義)**	*Boolean (1.1.2)*	

1.7　文字列の演算

型　*J* は型定義がないので文字列も数値も名詞（変数）の定義で区別しない

unicode　*Ver.5* からユニコードがサポートされた。変数名には使用できないが、データやダイアログボックス、グリッドの見出し、コメントにユニコードで漢字表示が利用できる。

文字とワード、文　*J* のプリミティブには *1* 文字毎（スペースも一文字）を対象とするものと単語や文を扱うものがある。

文章の分解　単語、文の取り扱いには文章の分解が大切である。

文字の種類

Font　*Shift-JIS,EUC,Unicode* が入り混じっている。日本語は *Shift-JIS,U8* 両用が多い。*J* で推奨されている *ISIJ* は *Unicode* 対応で、漢字も一応表示できる。*(*香港在住の開発メンバーがいる）

J – data	*char*	1*byte*	0 – 255
	wchar	2*byte*	0 – 65535
Code	*ASCII*		0 – 127
	*U*8	*Unicode*	0 – 65535

用法	*APL*	結果と例（単項）	用法	結果と例（両項）
a. *Alphabet* アルファベット		アルファベットなどの *256* の文字を表示する。 `  ]s=.a.i.'aA'` `97 65` **a，Aの位置を検出している。** `  (s+/i.26){a.` `abcdefghijklmnopqrstuvwxyz` `ABCDEFGHIJKLMNOPQRSTUVWXYZ`		
a: *Ace* （名詞）		**空のボックスのリスト** `  a:` `++` `||` `++`		

u: *Unicode*		*J* の *Ver5* から *Unicode* が利用できる *(U8)* *(u:*を用いなくとも普通に *Unicode* は利用できる*)* `u: 'JAPLA'` `JAPLA` `3!:0 u: A` `131072` `131072` **が出れば** `nicode`	*u:*	パラメータは *1-8* まである。 *1,2/8bitt* の *char,wchar* の高位置を改変 *6,7,8/wchar ASCII U8* の変換 *(Font* による差異あり） `3&u: 'JAPLAjapla'` `74 65 80 76 65 106 97` `112 108 97` `a.` **での文字の位置を示す**
`;:` *Word* 単語 `;: Y`		単語を識別 `;: ' ABC xyz'` `+---+---+` `|ABC|xyz|` `+---+---+` `> ;: 'ABC xyz'` `ABC` `xyz` `|.&. s: ' ABC xyz'` `+---+---+` `|xyz|ABC|` `+---+---+` *a.* を併用して数字、*Space*、文字、*NB.* 等を識別できる *(*複雑*)Vocabraly* 参照	*sequential* - *Machine* `m;:Y`	*0* － *5* の引数と右に複雑な引数を付加した文字列処理。*Vocaburary* に *quote(')* による文の区切りや途中の抜きなどの事例あり（複雑）

s: （小文字） *symbol* シンボル s: Y		文字ではなく単語を区分し認識させる a=: ’ APL JAPLA JAPAN’ **最初に１スペース空ける** s: a ‘APL ‘JAPLA ‘JAPAN |. s: a ‘JAPAN ‘JAPLA ‘APL |. a NAPAJ ALPAJ LPA **文字列に多くの関数が適用できる** \:˜ s: a ‘JAPLA ‘JAPAN ‘APL]A=.s:’**大 久 保 中 野 高 円 寺 荻窪**’ **‘大久保 ‘中野 ‘高円寺 ‘荻窪** **先頭にスペースを。単語の区切りは半角で。** **単語の前に‘が入れば良い** ,.A **‘大久保** **‘中野** **‘高円寺** **‘荻窪** |. A **‘荻窪 ‘高円寺 ‘中野 ‘大久保**	s: *symbol*	**文字列の演算のため** **多くのパラメータが用意されている** 5 s: s: a +----+--+----+ |hand|in|hand| +----+--+----+ ;: a +----+--+----+ |hand|in|hand| +----+--+----+

第2章

明示的定義と制御構文

2.1 明示的定義

verb_dyad	4 : 0	二項動詞の定義 (「4」と「コロン (:)」の間にはスペースが必要)
verb_monad	3 : 0	単項動詞の定義
conjunction	2 : 0	接続詞の定義
adverv	1 : 0	副詞の定義
noun	0 : 0	名詞の定義

グローバル (*global*) 定義 =:
ローカル (*local*) 定義 =.

global 定義では、その関数定義の枠を超えて定義が働く。

関数名の定義 .

◊ 動詞（関数）定義を書くときは関数名に=:を用いなければならない。

◊ 定義名が衝突すると後からロードした方が者優先になるので注意が必要である。（無警告）

明示的定義の中での変数などの定義 .

◊ *global* 定義は関数の枠を超えるので定義名が衝突すると後者優先になる。（無警告）

◊ 動詞の中での変数のグローバルデバックに便利であるが、最後に *local* に戻しておくとよい

代用値 「*y*」は右引数用で左引数には「*x*」を使う。

終了 ）は定義の終了を示す。

```
    mean=: 3 : 0
    (+/y) % # y
)

   mean >:i.10
5.5
```

◊ 明示的に定義した動詞は次のようにしてタシット *(*関数型*)* へ変換できる。

```
   mean=: 0 : '(+/y)%#y'
   3 : mean
3 : '(+/y)%#y'
```

```
   13 : mean
+/ % #
```

他の品詞についても、それぞれに呼応して *10*〜*12* にコロンを付して変換できる。（ただし変換が可能な場合に限る。）

2.2 制御構文

Ｊはリリース *2* から制御構文を採用した。制御構文は、明示的定義で利用できる。

2.2.1 if

J の条件文では、条件テストの結果が ” 真 ” ならば値「*1*」が、” 偽 ” ならば値「*0*」が返される。

if.T do. A end.

A が “ 真 ” のときに *A* を実行する。

if.T do.A else. B end.

A が “ 真 ” のときに *A* を実行し、”偽 ” のときには *B* を実行する。

```
if.T1 do.A1
   elseif.T2 do.A2
      elseif.T3 do.A end.
```

- 条件 *T1* が”真”のとき *A1* を実行する。
- ” 偽 ” ならば
  - 条件 *T2* を実行し、” 真 ” ならば *A2* を実行する。
  - ” 偽 ” ならば
  - 条件 *T3* を実行し、” 真 ” ならば *A3* を実行する

```
    zero=: 3 : 0
if. 0=y do. 'zero'
    else. 'nonzero' end.
)
```

```
    zero 3
nonzero
    zero 0
zero
```

```
    pzm=: 3 : 0
if. y=0 do.":0
  elseif. y>0 do.":1
    elseif. y<0 do.":_1 end.
 )
```

```
   pzm (L:0) 3 ; 0 ; _3
+-+-+--+
|1|0|_1|
+-+-+--+
```

2.2.2 while

while. A do.B end.
A が”真”である限り *B* を実行する。
whilst. A do.B end.
まず *B* を実行し、*A* が”真”である限り *B* を実行する。

```
   fact=:3 : 0
f=.n=.1+y
while. n>1 do.
   f=.f*n=.n-1 end.
 )
```

```
   fact ("0) i.5
1 2 6 24 120
   !i.6
1 1 2 6 24 120
```

```
   box=:3 : 0
b=.'' [ n=.>:y
whilst. n>1 do.
  b=.b,<i.n=.<:n end.
|.b
 )
```

```
   box 4
+-+---+-----+-------+
|0|0 1|0 1 2|0 1 2 3|
+-+---+-----+-------+
```

2.2.3 for

try. B1 catch. B2 end.
B1 を実行し、エラーがあれば *B2* を実行する。
for. B1 do.B2 end.
B1 の回数だけ *B2* を実行する。

```
   sum=:3 : 0
try. +/1+i.0>.y
  catch.'miss!' end.
 )
```

```
   sum 10
55
   sum 'a'
miss!
```

```
    fact1=:3 : 0
f=.n=.(<:n)*1:>.|)y
  for.i.<:n do. f=.f*n=.<:n end.
 )
```

```
   fact1 ("0) i.6
1 1 2 6 24 120
```

```
    iota=:3 : 0
s=.0:`$@.(]=0:) y
  for_j. i.|y do.
    if.j=0 do. continue. end.
    s=.s,j
  end.
|.^:(_1=*y)s
)
```

```
   iota 5
0 1 2 3 4
   iota _5
4 3 2 1 0
 iota 0
```

2.2.4 for_xyz

カウンターやインクリメントが不要。

```
csum=: 3 : 0
 for_ctr. i. # y do.
1!:2&2 tmp=. ctr{y
 end.
)
```

for_xyz
xyz に任意の文字を入れてカウンターに用いる *(i,j)* は *J* では至る所に用いられているので注意

```
   csum i.10
0
1
2
.
.
9
```

2.2.5　select

```
select. Y
   case. 0 do.D0
   case. 1 do.D1
   case. 2 do.D2
end.
```

「*Y=0,1,2*」に呼応して「*D0,D1,D2*」を実行する。

◊ さらに、次のような制御命令も用意されている

`break.` while(whilst) **ループ構文から抜け出す。**

`continue.` while(whilst) **ループ構文を続ける。**

`goto_name.` **ラベル名** label_name **へジャンプする。**

`label_name.` goto **のジャンプ先**

`return.` **定義構文から抜け出す。**

```
   case=:3 : 0
select. y
   case. 0 do. i.1
   case. 1 do.i.2
   fcase.2 do.i.3
end.
)
```

```
   case (L:0) {@> >:i.3
+-+---+-----+
|0|0 1|0 1 2|
+-+---+-----+
```

L:0 **による並列演算**

2.3 強制終了

無限ループに入ったときなどに、*J* の *bin* に入っている *jbreak.bat* のアイコンを *WINDOWS* の *pulldown menu* などに *copy* しておき、*2*, *3* 回クリックすると強制終了できる。

2.4　再帰

再帰の *primitive*(`$:`) が復活した。その使用法と最近出来た ***M.(Memo)*** と合わせて効果を計測する。

$!3 \rightarrow 3 \times 2 \times 1 \rightarrow 6$

J の階乗の *primitive* は**「*!*」**であり、良く錬成されていて早い。

```
   ! i.10
1 1 2 6 24 120 720 5040 40320 362880
   ts '! i.10'
6.05313e_6 896
```

ここでは再帰の学習のため再帰を用いた *3* 様の階乗を用いる。*Script* は *Chris.Burk and Cliff Reiter* による*)*

1　　明示型で定義

```
fac1=: 3 : 0
if. y<:1 do. 1
else. y * fac1 y-1
end.
)
```

2　　関数型による再帰。

```
fac2=: 1: '(*fac2@<:)@.*
```

3　　再帰の *primitive*(`$:`) を用いた定義。

```
fac3=: 1:'(*$:@<:)@.*
```

◊ また実行速度と ***Memo M.*** の有効性も計測する動詞をそれぞれ定義

```
fac10=: fac1 M.
fac20=: fac2 M.
fac30=: fac3 M.
```

タイマー　外部接続詞を用いて実行速度と使用メモリスペースを計測する

```
ts=: (6!:2),7!:2
```

各定義による計算結果

```
fac1 "0 i.10
1 1 2 6 24 120 720 5040 40320 362880
   fac2 "0] i.10
1 1 2 6 24 120 720 5040 40320 362880
   fac3 "0] i.10
1 1 2 6 24 120 720 5040 40320 362880
```

計測結果

- *fac1*　*M.* の方が幾分早い

```
   ts 'fac1 "(0) i.100'
0.0394941 148288
```

```
   ts 'fac10 "(0) i.100'
0.0352531 130944
```

- *fac2* は *fac1* より高速

```
   ts 'fac2 "(0) i.100'
0.0139541 46464
   ts 'fac20 "(0) i.100'
0.0137531 48320
```

- *fac3* は *fac2* より早い。*M.* の効果はない。*Tuning* が進むと *M.* の効果は相殺されるようだ。

```
   ts 'fac3 "(0) i.100'
0.00592404 46272
   ts 'fac30 "(0) i.100'
0.00655782 48320
```

関数型の再帰スクリプトは難解であり、単独の科学技術計算やプログラムやロジックの学習に限定した方が無難である。

2.5 Mapped file

J には更に直接メモリ周りを扱う *Mapped file* の機能がある。詳細は *LAB* の *tutorial* 参照

.

第3章

外部接続詞

3.1 概要

外部接続詞「*!:*」とは所謂システム関数である。

0! : *n*	*Scripts*
1! : *n*	*Files*
2! : *n*	*Host*
3! : *n*	*Conversions*
4! : *n*	*Names*
5! : *n*	*Representation*
6! : *n*	*Time*
7! : *n*	*Space*
8! : *n*	*Format*
9! : *n*	*GlabalParameters*
11! : *n*	*WindowDriver*
13! : *n*	*Debug*
15! : 0	*DynamicLink*
18! : *n*	*Locales*
128! : *n*	*Miscellaneous*

外部接続詞はここ一番では有用であるが *load,require* など英語で別途定義されてものが増えてきており、普段使用する頻度はそれ程多くはない。

128!:n は暫定サポートの色合いが濃い

よく使われるものを *my_util.ijs* 等に登録しておいて、随時読み込んで使用すると便利である。[*1]

Help → Foreign Conjunction に詳細な解説が入っているので、詳細はそちらを参照のこと

[*1] 起動時に *profile.ijs* で読み込むこともできる。

使用する *EXAMPLE*　フィボナッチ数

filename fib_test.ijs

```
NB. fibonacci number
 fib=:(0 1,: 1 1) +/ . * ]
 NB. Usage: fib ^:(i.100) 1 1
```

```
    fib ^:(i.12) 1 1
 1   1
 1   2
 2   3
 3   5
 5   8
 8  13
13  21
```

3.2　外部接続詞の解説

3.2.1　0!:n /Script

load ,require があるので直接使うことは少ない。

key-in には必要となる

0! : *k*　*Y*

0! : 2　*Y*　　`in=:0!:2@<`

0! : 3　*Y*　非表示で読み込み

k(*bit*)	*a*	*b*	*c*
0	ファイル	*S* エラー停止	*Silent*
1	キー	継続	*Display*

◇　*k* = 111 の例

```
0!:111<'/temp/fib_test.ijs'
```

3.2.2　1!:n/Files

1!:0　ファイルの属性を表示

```
   1!:0<'/temp/fib_test.ijs'
+------------+-----------------+--+---+------+
|fib_test.ijs|2010 2 3 16 15 58|77|rw-|-----a|
+------------+-----------------+--+---+------+
```

1!:1(1)	キーボードから入力 `ask=: 3 : '1!:1(1)'` `a=: ask 'whats your point'` `eiko is 20` `a` `eiko is 20` `a=: ask 'whats your point'` `akiko is 30` `a` `akiko is 30`
1!:2	`1!:2&2` *loop* の結果を取り敢えず画面に出せば良いときなどに便利 `1!:2&2 fib ^:(i.10) 1 1` `1!:2&4 fib ^:(i.10) 1 1`
1!:4	ファイルサイズ `1!:4<'/temp/fib_test.ijs'` `77`

3.2.3 2!:n/HostCommands

2! : 55″ *J* を強制終了

3.2.4 3!:n/ Conversion

3! : 0 *y* 名詞の型の判別
3! : 1 *y* 内部表現の２進表示
3! : 2 *y* 「3! : 1*y*」の逆変換
3! : 3 *y* 内部表現の１６進数

3!:0 y のリターンコード

1	*Boolean*
2	*Literal*
4	*Integer*
8	*Floatingpoint*
16	*Complex*
32	*Boxed*
64	*ExtendedInteger*
128	*Rational*

```
3!:0 u: 'JAPLA'
```

```
131072  NB.131072 は unicode
  3!:0 'JAPLA'
2       NB. Literal
```

Integer Hex Float 等の *conversion* 機能あり

3.2.5　4!:n/Names

◇ 取り敢えずは *names ” を!*

```
   names''
Y    a     cube  fib
```

4! : 0　　*<′ y′*　定義内容の品詞
4! : 1　　*y*　　*namelist*
4! : 1(0)　*y*　　名詞を全て表示 *t*
4! : 1(3)　*y*　　動詞を全て表示 *t*
4! : 3　　*<′ y′*　*script*
4! : 4　　*<′ y′*　*script index*
4! : 55　　*<′ y′*　*erase*

◇「*4!:0 y*」のリターンコード

0　　*Noun*
1　　*Adverb*
2　*Con junction*
3　　*Verb*
6　　*locale*

3.2.6　5!:n/Representation

定義内容のいろいろな表示

*x*5! : 0′*y*′　　*De f ine*
5! : 1 *<′ y′*　*Atomic*
5! : 4 *<′ y′*　*Tree*
5! : 5 *<′ y′*　*Linear*
5! : 6 *<′ y′*　括弧表現

```
  mean=: +/%#
   5!:1<'mean'  NB. 詳しく
+-----------------+
|+-+-------------+|
||3|+-------+-+-+||
|| ||+-+---+|%|#|||
|| |||/|+-+|| | |||
|| ||| ||+|||  | |||
|| ||| |+-+||  | |||
|| ||+-+---+|  | |||
```

```
|| |+-------+-+-+||
|+-+-------------+|
+-----------------+
   5!:2<'mean' NB. box
+-----+-+-+
|+-+-+|%|#|
||+|/|| | |
|+-+-+| | |
+-----+-+-+
   5!:4<'mean' NB. tree
  +- / --- +
--+- %
  +- #

     5!:5<'mean' NB. linear
+/ % #
   5!:6<'mean' NB. 括弧付き
(+/) % #
```

(1/2) 5!:7<'foo' NB. explicit の時 (1/2) は単項・両項

◇「*5!:0 y*」のリターンコード

0	*Noun*
2	*Hook*
3	*Fork*
4	*BondedConjunction*

3.2.7 6!:n/Time

タイムスタンプ。実行速度計測に *6!:2* を使用

6! : 0′*y*′	現在時間 `6!:0 ''`	2010127144932.203
6! : 1′*y*′	セッション経過時間 `6!:1 ''`	16196.8
6! : 2′*y*′	実行時間 `6!:2 'fib ^:(i.1200) 1 1'`	0.00613346
6! : 8′*y*′	*clock* の精度	6! : 8″
6! : 9′*y*′	同じ	6! : 9″
6! : 10′*y*′		
6! : 11′*y*′		
6! : 12′*y*′		

3.2.8 7!:n/Space

使用メモリーの量

```
   7!:0'y'        現在使用量          7!:0 ''     1311232
   7!:1'y'     セッション使用量       7!:1 ''     9127168
   7!:2'y'     センテンス使用量     7!:2 'fib'      640
   7!:3'y'     Jメモリー使用量
   7!:5'y'  オブジェクトの使用量   7!:5 <'fib'      704
   7!:6'y'     ロケールの使用量
```

```
   7!:3 '' NB. free Space with J block size
  64 1537
 128  732
 256  325
 512  139
1024   27
```

3.2.9　8!:n/Format

```
''8!:0 i.3 3
+-+-+-+
|0|1|2|
+-+-+-+
|3|4|5|
+-+-+-+
|6|7|8|
+-+-+-+
```

```
''8!:1 i.3 3
+-+-+-+
|0|1|2|
|3|4|5|
|6|7|8|
+-+-+-+
```

```
'b<nil>d<n/a>c8.2' (8!:2)  1.23 123 0.123 , __ 0 _123
    1.23  123.00    0.12     n/a     nil -123.00
```

いろいろなフォーマットを指定したフォーマットに統一する。(8j2 **ではない**)

文字列

3.2.10　9!:n/Global Parameters

諸機能を大勢集めた

9! : 0 *'y'* *Random Seed/Roll*
9! : 1 *'y'* *Random Seed/Deal*
9! : 2 *'y'* *Default Displays*
9! : 3 *'y'*
9! : 6 *'y'* ボックス枠のキャラク
9! : 7 *'y'* ボックス枠の変更
9! : 8 *'y'* *ErrorMessage* をフランス語で
9! : 9 *'y'* *FrenchErrorMessag*
9! : 14 *''* J の *Version*
9! : 18 *''* *Boxdisplay*

- *Explicit* の *Tacit* への変換（*13*：*0*）を用いる

```
       9!:3(5)
reg=: 13 : 'x %. 1,. y'
   reg
[ %. 1 ,. ]
```

- *box* や *tree* で定義を確認する *(13 : 0)* を用いる。*9!:11* でも *tr ee* 構造が見られる

```
   9!:3(2) NB. Box
   reg
+-+--+--------+
|[|%.|+-+--+-+|
| |  ||1|,.|]||
| |  |+-+--+-+|
+-+--+--------+

   9!:3(4)   NB. Tree
   reg
  +- [
  +- %.
--+    +- 1
  +----+- ,.
       +- ]
```

3.2.11 11!:n/WindowDriver

11! : 0*'y'* *WindowDriver*

3.2.12　13!:0/Debug

13! : 0　*y*　*reset*
13! : 1　*y*　*display stack*
13! : 2　*y*　*stop query*
13! : 3　*y*　*stop set*
13! : 4　*y*　*run again*
13! : 5　*y*　*run next*
13! : 6　*y*　*exit return*
13! : 7　*y*　*continue*
13! :　9, 11, 12, 13, 14, 15, 17, 18 *y*

3.2.13　15!:n/Dynamic Link Libraly

15! : 0　*y*　*Call DLL function*
15! : 1　*y*　*Memory read*
15! : 2　*y*　*Memory write*
15! : 3　*y*　*Allocate memory*
15! : 4　*y*　*Release memory*

3.3 外部接続詞 128&n

外部接続詞 128!:n は暫定サポートが含まれている

J の固有値関数「*c.*」のサポートが遅れており、*LAACK* を利用できるようにした。

128! : 0	*y*	*QR* 分解	1.1.6 参照
128! : 1	*y*	上三角行列の逆行列	1.1.6 参照
128! : 2	*y*	文字化した動詞 *u* を *y* に適用	
128! : 3	*y*	*CRC polynomial*	
128! : 4	*y*	*RnG*/*RAW*	他の乱数の採用
128! : 5	*y*	*NaN*	*notanumber*

◇ 128!:0 の左は *Gram·Schmidt* の直交変換である

◇ *NaN* は http://www.jsoftware.com/jwiki.Essays の *NaN* を参照

3.4　英語で書かれた幾つかの関数（コマンド）

プリミティブは記号でという *APL_J* にも例外がある。

- 一つは制御構造を *APL* 流から普通の言語風に変えた。
- 次は特定のファイルをロードしなくても予め組み込まれている英語の関数（コマンド）がある。幾つかの例を挙げる。

  require　*require 'plot numeric trig' NB.* 一度ロードされてあれば再ロードしない

  load　*load 'plot numeric trig' NB.* 毎回ロードする

  table　見出しを付ける

  jpath　*J6.02* からレジストリを切断し、*J* は *USB* や *CDROM* からも起動できるようになった。代償として *J* の入っているディレクトリが多岐になった。　***jpath*** が認識されていれば、

  `require jpath '~user/classes/xxx.ijs'` などと *J* のディレクトリから記述できる。

```
   (>:i.9) */ table >:i.9
+--+-------------------------+
|*/|1  2  3  4  5  6  7  8  9|
+--+-------------------------+
|1 |1  2  3  4  5  6  7  8  9|
|2 |2  4  6  8 10 12 14 16 18|
|3 |3  6  9 12 15 18 21 24 27|
|4 |4  8 12 16 20 24 28 32 36|
|5 |5 10 15 20 25 30 35 40 45|
|6 |6 12 18 24 30 36 42 48 54|
|7 |7 14 21 28 35 42 49 56 63|
|8 |8 16 24 32 40 48 56 64 72|
|9 |9 18 27 36 45 54 63 72 81|
+--+-------------------------+
```

  names　ロードされている名詞や動詞などの一覧

```
   names '' NB. Example
char             char_ecev
char_evec0       char_frame
char_invmat      char_lf
```

  expand　*APL* にはあった *expand* に対応する *J* のプリミティブが当初はなかった。後に *copy*　「`#`」の逆関数`# ^:_1` として組み込まれ、英語のコマンド ***expand*** も別途用意された。

```
   1 0 1 1 0 1 0 1 expand 'JAPAN'
J AP A N
   1 0 1 1 0 1 0 1 expand 1 2 3 4 5
```

```
1 0 2 3 0 4 0 5
   1 0 1 1 0 1 0 1 #(^:_1) 1 2 3 4 5
1 0 2 3 0 4 0 5
  1 0 1 1 0 1 0 1 #(^:_1) !.(100) 1 2 3 4 5
1 100 2 3 100 4 100 5
```

erase　ロードされている名詞や関数の消去

```
   erase 'A0 C0'
1 1
```

- 関数定義の明示型も英語で型を定義することができる。区切りにアンダーバーは不要。最終的に *J* の内部で *4 : 0* や *3 : 0* 等に戻している。

  dyad define　*4 : 0*

  verb define　*3 : 0*

```
reg=: dyad define
x %. 1,. y
)

mean =: verb define
 (+/ y) %  # y
)
```

  ◊ *dyad* と *define* の間はスペース *(␣)* である

第4章

Install Manual

4.1 Jのインストール

Download	http://www.jsoftware.com からダウンロードできる *Platform* *Linux 32/64 bit* *WIN 32/64bit* *MAC PPC/INTEL* *WIN-CE* 何れも *Download* は自由で使用は無料。自由に複写できる。	*BSD* でも稼働
Install	Registry　*J* は *Registry* を切断しているので、*CDROM* や *USB* に入れて持ち運べるので、どこ、どれからも *J* を使用できる。 Install　取り敢えずはお任せ *Install*。*WIN* ならば *Documents and Settings* の *user-name* の下の *Folder* に *install* される。 ここに、*j*602　*j*602 – *user* の *2* の *folder* が作成される。 copy　好みの *folder* や *USB* に *copy* する。	
configure	使う *folder* 決まったら、*configure* を *Click* して、*font,color* などを調整する。 *Font* は *ISIJ* が見やすい	*EDIT* の一番下にある ここの結果は *j602-user/config* に出力される。

4.2 Package と Add-on

便利な *package* や *add-on* が提供されている。

Document は *lapack.ijt* など *J* の *tutorial form(xx.ijt)* で提供されるものは *xxx.ijt* を印刷するか、*J* の *Studio/Labs* から、又は直接 *xx.ijt* を読み込んで *Ctrl* + *J* で *page* 送りをしながら読む。*demo* になっている場合もある。

xx.ijt が入っていない場合は、*xx.ijs* を印刷して *comment* を解読する。*(*解読結果の *report* をお願いしたい。*)*

4.2.1 Packages

J の *system* の下に *Examples* と *packages* の *folder* があり、一通りの *package* や *tool* 類が入っている。代表的なものは次の通り。

ディレクトリ	*ijs*
Packages	*finance*
	math
	stats
Examples	*phrase*

4.2.2 addon

主に *supporter* を中心に多くの *ADD-ON* が作成されている。

Internet に繋がっていれば *RUN* の下の *Package Manager* に *check* を入れ、ここで好みのものを *check* すると、自動で *install* される。

Iniernet に繋がっていないときは

`http://www.jsoftrare.com` から *J* の *version* を合わせて、*Download* して、解凍する。*file* の場所は一番下にあるので分かりにくい。

`http://www.jsoftware.com/jal/j602/addons/`

繋がっている人から *copy* させて貰う方が早道である。

LAPACK

Add-on に *LAPACK* がある。*J* の固有値と固有ベクトルが満足できる精度が得られないのかサポートが随分遅れており、*LAPACK* が推奨されている。

LAPACK の使用法は *Studio* → *LAB* に入っている。（最近改訂されたようだ。）

次の *2* 個のファイルを *load* する。*Run File* が便利

lapack.ijs

*geev.ijs(*または *dgeev.ijs)*

`geev_jlapack_ data(matrix)` で中央の *Box* に固有値が、右には固有ベクトルがでる。*Locale* の書式で `jlapack_`が入る。

LAPACK には *SVD,QR,LU,Choleski* 分解なども入っている。

4.3 tutorial

tutorial は *Studio* → *LAB* に入っている。他にも.*ijt* 形式の *file* が入っていれば *tutorial* であり、*LAB* で扱うことができる。

4.4 Books Manual

Manual は *Help* から *Usr* で辿ることができる。

Help → *Vocabulary* の上の欄に一覧が出る。

多くの *online-book* が利用できる。

J Primer が最初の一冊。

K.E.Iverson の著書は

`http://www.jsoftware.com` に上がっている。

Calculus

Concrete math Comparison *D.Knuth et.al*

Math for Liemann

Exploring math

Arithmatic

Iverson の *Schumann series* の一部は *LAB* → *LiveText* に入っている。

J の *homepage* から *online-books* の一覧が得られる。

1. 入門, 初級

Henry Rich		*J Reference Card*
Cris Burk	*Clifford Reiter*	*J brief Reference*
Linda Alvord	*Norman Thomson*	*Easy J*
Roger Stock		*Learning J*
		J Phrases

2. 中、上級

Henly Rich	*J for C Programmers*

3. *J Phrase*

   言葉のフレーズで *J* 言語の語彙集。関数定義の宝庫

J 関係の図書で出版されたもの一覧と入手方法が紹介されている。

4.5 J602 での非互換の変更

APL では考えられない次のような**非互換の改変**が行われた。*J602* 以前で *x y m n u v* を小文字 *1* 文字で変数名に使っている *script* は大幅な書き換えが必要になる。*(*例えば *x0 y0* などに）

代用値	*x.*	*y.*	→	*x*	*y*	汎用
	m.	*n.*	→	*m*	*u*	名詞を引用
	u.	*v.*	→	*u*	*v*	動詞を引用
文字化	6.3	”:	→	6*j*3	”:	表示桁数指定

graphics では画面が左下 *(0,0)* 右上 *(1000,1000)* から左上 *(0,0)* 右下 *(200,200)* に変更。
graphics command も多少変更 *glshow* → *glpaint*

第5章

J の各種の機能

5.1 PLOT

- *J* の *plot* は *graphics* 専用 *tool* と遜色がない。
- 簡易と本格的な用法がある。
- 複素数の *plot* もできる。
- 簡易 *graphics tool* として用いることもできる。
- *3Dbar* はない

5.1.1 require

`require 'plot numeric trig'` *NB.* 最初に
trig は円関数の定義 *file*
numeric は *utility*

5.1.2 data の plot

- *data* を作る　*sin i:10*

```
   3 7 $  sin i:10
 0.544021 _0.412118 _0.989358 _0.656987 0.279415 0.958924  0.756802
 _0.14112 _0.909297 _0.841471         0 0.841471 0.909297   0.14112
_0.756802 _0.958924 _0.279415  0.656987 0.989358 0.412118 _0.544021
```

- *X;Y 2Dplot* の *data* の型。*X* は省略可能で省略すると自動で `i. # Y(0 1 2 3 ..)` をとる。
  `(i:10);sin i:10`

  `X;Y` が `PLOT` のデータの基本

```
   (i:2);sin i:3
+-----------+-------------------------------------------------------+
|_2 _1 0 1 2|_0.14112 _0.909297 _0.841471 0 0.841471 0.909297 0.14112|
+-----------+-------------------------------------------------------+
```

- 一寸実用向けに *steps* 関数を用いる。
  `'line,stick' plot sin steps _3 3 100`
  （*steps from _3 to 3 divide by 100*）
- 画像 *file*。*eps* は *postscript* 画像で *TeX* で良く用いる。*WIN* では *emf* が扱いやすい。

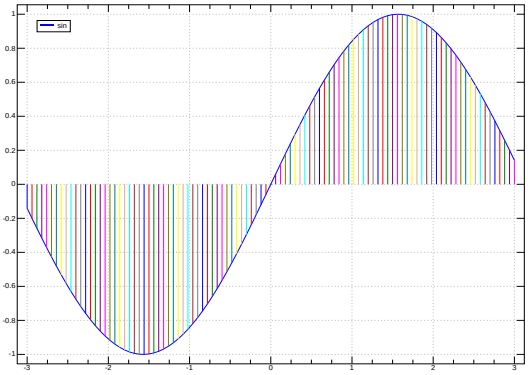


図 5.1　sin

`pd 'eps /temp/plot_01.eps'` *NB. eps file* を作成する

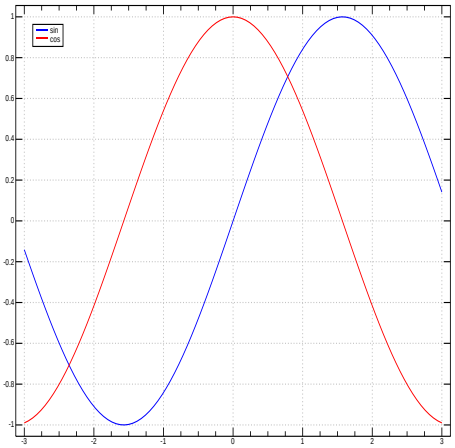
5.1.3　Science plot

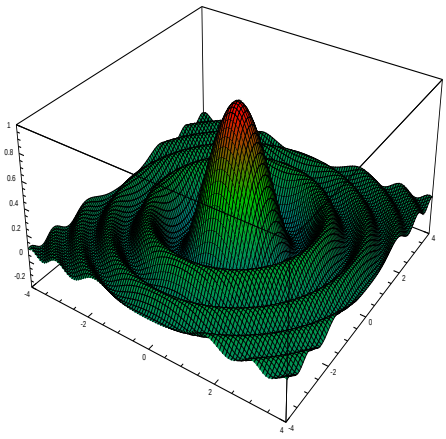
最近のバージョンでは *science plot* がサポートされている。

`'key sin cos' plot _3 3 ;'sin'cos'`

`_3 3 ; 'sin'` と区間と関数を指定するだけで足りる。*user* 定義関数も当然使える。

```
fnc=: 4 : '(cos r)% >: r=: x + &:*: y' NB. ソンブレロ
plot _4 4 100;_4 4 100;'fnc'  NB. 3D
```





5.1.4　Type

代表的なタイプ

bar　　*sbar*　　*fbar*

line

stick

surface　　*3D – only*

marker　　*point*　　*dot*

'bar,line','bar,stick','marker,line' 等と指定できる

5.1.5 画像の save

画像が出ている状態で

clip *pd 'clip' clipboad* に入る。
eps `pd 'eps /temp/plot_01.eps'` *NB. eps file* を作成する
ems `pd 'save emf /temp/plot_01.emf'`

画像は身近な*/temp* などに出した方が早道。
bitmap で *save* したいときは *C.Reiter* の *add-on* を組み込む。

5.1.6 複素数

複素数は一つの数で *X,Y* 軸の座標を持っている。*line* では *2* 個以上のデータが必要。
（実数も同時に描けるが *X* 軸の指定が出来ないので、値を *X* 軸上に描く。）

```
'marker' plot 3j6 2j3 _1j3 0j_3  0  1  2
```

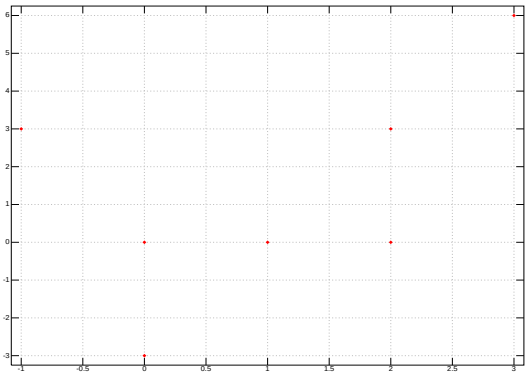


図 5.2 complex number

複素数を *Graphics* に積極的に活用したのが *turtle graphics* である。

5.1.7 Presentation 用の画像

明示型で *pd driver* を用いてプログラムする。
plot の *Manual* は
`http://www.jsoftware.com` の *WiKi* の *page* に移った。
Example を上げておく。*y2axis* を用いた本格的派。

```
plot_example=: 3 : 0
pd 'reset'      NB. or new
pd 'type line,stick'
pd 'keypos open bottom'
pd 'keystyle open horizontal'
pd 'key sin cos'
pd 'color blue'
pd sin steps _3 3 100
pd 'y2axis'
pd 'color green'
pd cos steps _3 3 100
pd 'show'
```

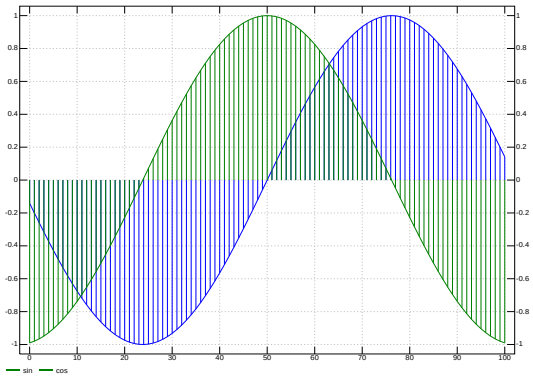


図 5.3　sin

さらに画面を分割したもの。

```
    plot_cascade2 =: 1 : 0
DAT=: 3 20$ 60?200
NB. ----------------------------
pd 'reset'
pd 'sub 0 0 1000 _40'
pd 'sub 3 1  '                    NB.   3 画面を定義
NB. ---------0 0-----------------
pd 'new'                         NB. 画面 0
pd 'keypos open bottom'
pd 'keystyle open horizontal'
pd 'key 0'
pd  0{DAT
NB. ----------0 1------------
pd 'new'                          NB. 画面 1
pd 'keypos open bottom'
pd 'keystyle open horizontal'
pd 'key 1'
pd  1{DAT
pd 'new'                        NB. 画面 2
pd 'keypos open bottom'
pd 'keystyle open horizontal'
pd 'key 2'
pd 2{DAT
pd 'show'                      NB. 表示
)
```

5.1.8 miscellaneous

どうしても *3D-bar* が使いたい場合は *Oleg Kobchenko* の *page* にキメラ風 *3D-bar* がある

5.2　Turtle Graphics

タートルグラフィックスと言えば *Logo* が有名であった。*Fraser Jackson* の貢献により、ほぼ同様に *J* でも使える様になった。

ガイダンスが *J* の *LAB* に入っている。[*1]

詳細な *LAB* なので、マニュアルの代用にもなる。*Lab* を参照しながら *turtle* を動かしてみよう。この *LAB* は印刷すれば *20* ページを越え、マニュアルとしても十分利用できる。

この章は *turtlegraphics* の入門と鑑賞を専らとする。

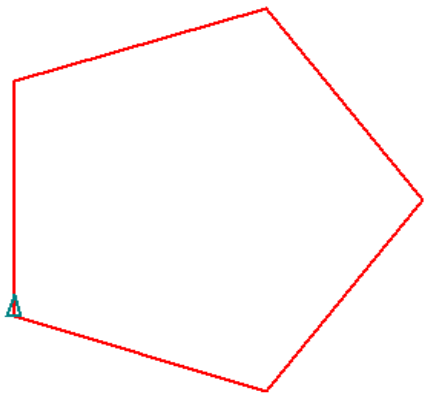
Logo のテキストが手元に有ればほとんどそのまま使えるようだ。

最初に *turtle* を読み込む。*LAB* から動かす場合は *Studio* → *Lab* でダイアログボックスの *Turtle Geometory,J Turtle User guide* を選ぶ。

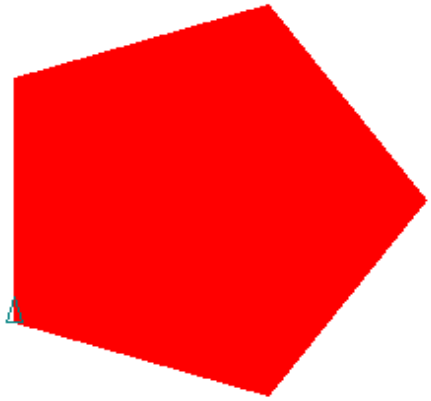
5.2.1　Turtle 事始め

最初に、

```
'require 'turtle'
```

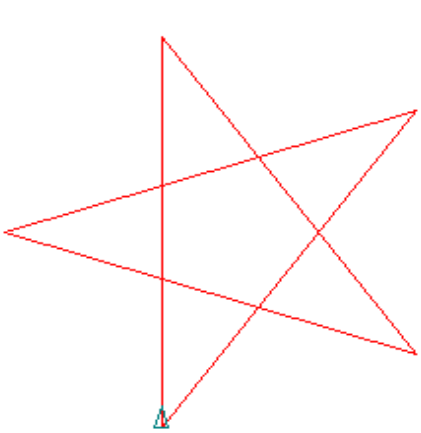


show repeat 5 fd 1 rt 360r5
NB. pentagon = 72^o
1 進み 72^o のターンを 5 回
繰り返し正 5 角形を描く
show repeat 6 fd 1 lt 360r6
NB.hexagon = 60^o

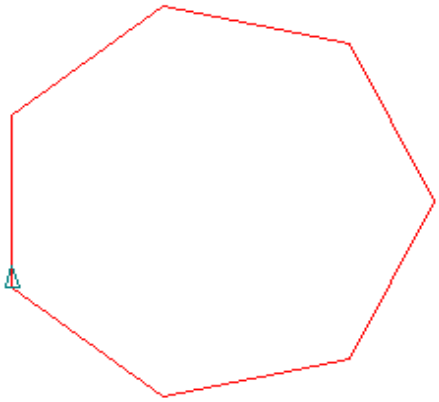


show *fill repeat 5 fd 1 rt 360r5*
fill 塗りつぶし
色は *J* の *COLOR16* の指定が利用できる。デフォルトは *RED*
show pencolor GREEN fd 2
pencolor は *pen*（線）の色
詳細には *rgb* で指定する。
show pencolor (123 34 12) fd 2

[*1] Graphics ではなく General Interests に入っている。



show poly 1 144



show poly 1 360r7

5.2.2 基本コマンド

move,turn

fd *forward*
bk *backwards*
rt *turns right*
lt *turns left*

show

3 種類ある

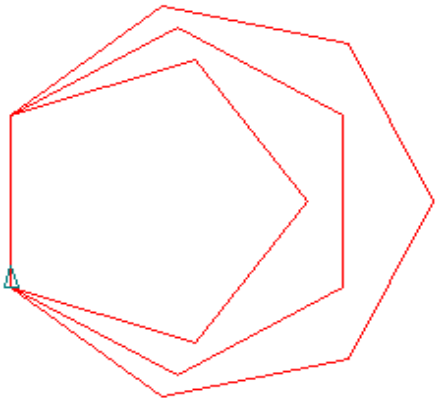
show		標準
showf	*fix*	中心から始め、上下左右各 *100* ステップの固定画面
showp	*plot*	*plot* 画面に描く

save/clip

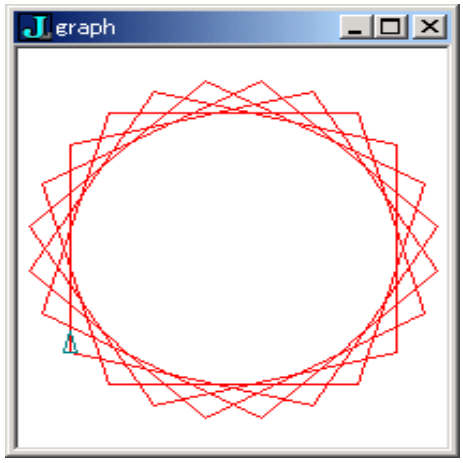
clip	*clip”* で、クリップボード経由でツールで *save* する。	*clip”* *NB. pd* は不要
save	*save ’temp/turtle_29.emf’*	*emf* は可能 *save eps* はなさそうだ。（*pd* は不要)

5.2.3　いろいろな形

polygon



show 1 360r5
前の 1 を 2 にすると縮小する



show poly 1 75
$75^o \times 4 = 300^o$, 余り 60^o
余りとの最大公約数まで回転する。75 を変化させるといろいろな patarn が現れる

spiral

　ループを簡単に入れ込み、その上に *repeat* で更に繰り返している。[は左側を用いる。*(*右] はエラーが出る*)*

show repeat 30 ;’fd (a =: a+1) rt 360r3’ [a =: 0
3 角形
show repeat 60 ;’fd (a =: a+1) rt 360r8’ [a =: 0
8 角形
show repeat 40 ;’fd (x=:x+1) rt 360r4’] x=.0
4 角形

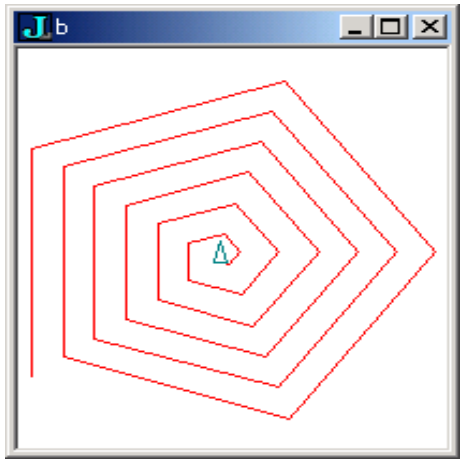


図 5.4　spiral

5.2.4 少し高度なテクニック

```
show ({1 2 3 4 5;1 2 3)
start (rt 18 poly 0.5 144)
```

poly 0.5 360r5 では正 5 角形

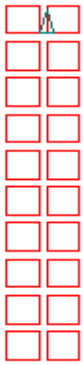


図 5.5 cars

```
show ({1 2 ;1 2 3 4 5 6 7 8 9 10)
```

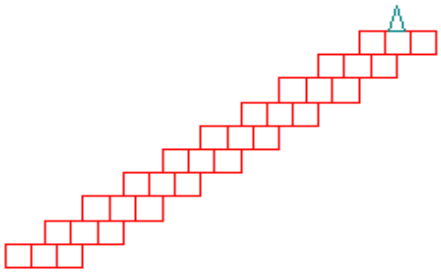
start sq 0.8

catalog ({) の組み方を 1 2 3 4 5 ; 1 2 3 などと変えてみる。

図 5.6 box

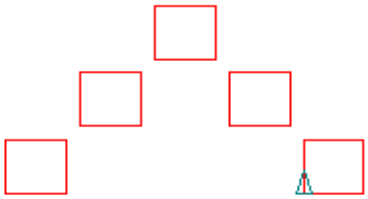
```
  show  (23 {. ,a j./a =. i.10) start sq 0.8

                j./~ (i.10)
0 0j1 0j2 0j3 0j4 0j5 0j6 0j7 0j8 0j9
1 1j1 1j2 1j3 1j4 1j5 1j6 1j7 1j8 1j9
2 2j1 2j2 2j3 2j4 2j5 2j6 2j7 2j8 2j9
3 3j1 3j2 3j3 3j4 3j5 3j6 3j7 3j8 3j9
4 4j1 4j2 4j3 4j4 4j5 4j6 4j7 4j8 4j9
5 5j1 5j2 5j3 5j4 5j5 5j6 5j7 5j8 5j9
6 6j1 6j2 6j3 6j4 6j5 6j6 6j7 6j8 6j9
7 7j1 7j2 7j3 7j4 7j5 7j6 7j7 7j8 7j9
8 8j1 8j2 8j3 8j4 8j5 8j6 8j7 8j8 8j9
9 9j1 9j2 9j3 9j4 9j5 9j6 9j7 9j8 9j9
```

ベクトル化して最初の 23 個取り出す。25 とすると右に 2 個積み上がる



show blocksquares 2 10 3
1 10 3 とすると垂直

図 5.7　block



show 1j3 2j4 3j5 4j4 5j3 start sq 0.8

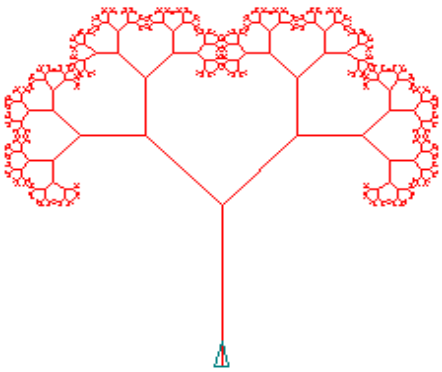
3*j*5
2*j*4　　4*j*4
1*j*3　　5*j*3

横方向は整数部で、縦方向は虚数部で表す。

図 5.8　position

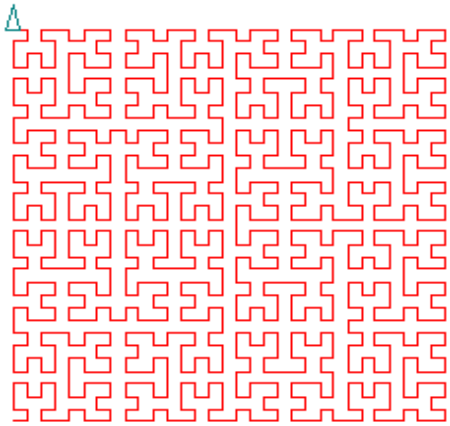
5.2.5　鑑賞

LAB を動かしながら鑑賞



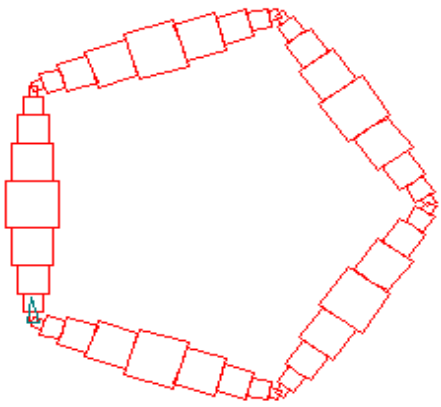
show branch 2 10

図 5.9 branch



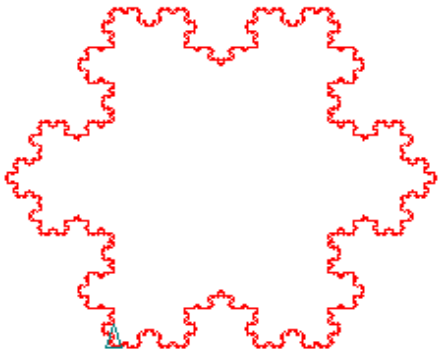
show rhilbert 5 5

図 5.10 hilbert



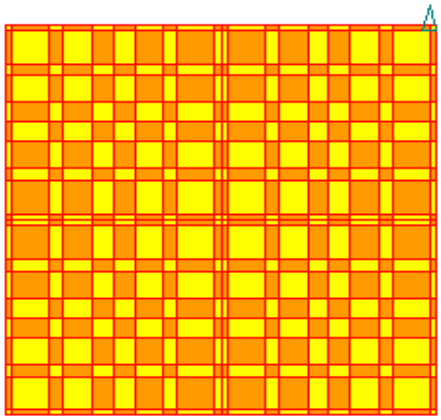
b =: recursivesq ”0] 1 2 3 4 5 4 3 2 1
show repeat 5 ,(,b) rt 72
72°の 5 角形で模様（*b*）を繰り返し描く

図 5.11 recursive



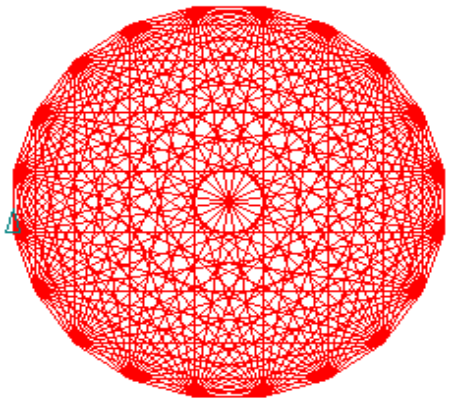
show snowflake 4 5

図 5.12　koch



show (js,(0j30+js),(30j0+js),30j30+js) start
<”1 jn0,jn1,jn1,jn0
詳細は LAB の最後を参照

図 5.13　aya



Fraser Jackson の傑作
show repeat 20 ;’ point (n=.n+1) fd 1 rt 18’ [
n=. _1
to ’goreturn a b’;’goto b goto a’
to ’goreturnall a’;’goreturn for a,.i.20’
show goreturnall for i.20

図 5.14　Fraser Jackson

5.3 J のファイルシステム

5.3.1 CSV ファイル

CSV ファイルとはカンマで区切った数値のブロックデータ。数字に見えるが数字の形をした文字で構成されている。数値化出来ない（本来の）文字とはボックスを用いないと混在できない

CSV Comma Separated Value

最初の約束 `require 'files csv'`

TestData **の作成** `DAT=. 20 5 $ ?. 100#100`

作成と書き込み *csv file* の作成と書き込み

```
   DAT  writecsv '/temp/testfile.csv'
509
```

csv file 書き込んだ *csv file* の中身を *editor* で見ると

```
"46","55","79","52","54"
"39","60","57","60","94"
"46","78","13","18","51"
"92","78","60","90","62"
"31","16","60","64","64"
"71","13","3","76","26"
```

読み込み *csvfile* を読み込む。*Box* に入った数字の形をした文字である。

```
   readcsv '/temp/testfile.csv'
+--+--+--+--+--+
|46|55|79|52|54|
+--+--+--+--+--+
|39|60|57|60|94|
+--+--+--+--+--+
|46|78|13|18|51|
+--+--+--+--+--+
```

数値化 数値化して *Box* を開く。

`(".@>)` で形が崩れるときは次数が多いときが多い。このような場合は

- 先ず *Box* のままで`". (L:0)` で数値化を行う
- 次に`;("1)` **や**`;("2)` で *Box* を開く。

```
".@>  readcsv '/temp/testfile.csv'
```

```
46 55 79 52 54
39 60 57 60 94
46 78 13 18 51
..............
37  8 37 42 97
35 66 35 24 39
```

5.3.2 J 固有のファイルシステム Jfiles

J の *filesystem* として堅牢な *Jfiles* がある。*binaly* などの *data* には別に *nfiles* がある。

- *CSV* ファイルのように *editor* で見ることは出来ない。
- *CSV* と異なり複数の *data* を扱うことが出来るので一つのセッションをカバーできる。
- *J* の数値 *matrix* は *3* 次でもそのまま扱える
- *box* *data* や文字列も扱える
- 全体を扱う作業向き。（*APL* の *workspace* ライク）*file* から *1* 件ずつデータを読み書きするには不向き

基本命令は次の *7* 個.*_1* などマイナスが出たら *error*

操作	*J* のコマンド	作用
create	*jcreate*	*jcreate filename*
copy	*jdup*	*jdup filename*
write	*jappend*	*data jappend filename*
replace	*jreplace*	*jreplace filename*
check	*jsize*	*jsize filename*
read	*jread*	*jread filename*; *n*
erase	*jerase*	*jerade filename* *file* 自身を完全消去（部分消去不可）

load 最初の約束

```
require 'jfiles'
```

create *temp* に *testfile.ijf* が作成される。*extention(.ijf)* は一切不要

```
jcreate '/temp/testfile'
```

write 書き込みは *append* で。

```
   DAT1=. 2 3 5 $ 30 ? 30
  DAT1 jappend '/temp/testfile'
0
```

check 書き込みの *check*

```
  jsize '/temp/testfile'
0 1 2048 0
```

append 追加の書き込み *append DAT2*

```
]   DAT2=. 5 4 $ ?. 20#20
 6 15 19 12
14 19  0 17
```

```
 0 14  6 18
13 18 11 12
18  0 10  2
```

```
   DAT2 jappend '/temp/testfile'
1
```

check *2 の file がある*

```
    jsize '/temp/testfile'
0 2 2176 0
```

read *DAT1 を読み込む。**(;0)** で指定する*

```
   jread '/temp/testfile';0
+--------------+
|10 28 12 26  4|
| 2  9  8 23 16|
| 1 19  5 18 11|
|              |
| 6 20 22  0 14|
|29 25 15  3 13|
|21 27  7 17 24|
+--------------+
```

read *DAT2 を読み込む。(;1)*

```
   jread '/temp/testfile';1
+-----------+
| 6 15 19 12|
|14 19  0 17|
| 0 14  6 18|
|13 18 11 12|
|18  0 10  2|
+-----------+
```

read once 一度に中身を見る

```
jread '/temp/testfile';0 1 2 3
```

replace 次のように置き換える。

```
   DAT2 jreplace '/temp/testfile';1
1
```

3 **次** matrix *2 枚の table も分割しないで扱える*

```
  ] DAT3=: 2 4 3 $ ?.  ?. 30#30
   DAT3 jappend '/temp/testfile'
```

```
2
   jsize '/temp/testfile'
0 3 2688 0
   jread '/temp/testfile';2
+--------------------+
| 0         0       11|
| 3         2        0|
| 4         1        1|
| 5         5        4|
|                     |
|11         1        0|
| 0         0 0.250049|
| 9         5        1|
| 1 0.0824937       13|
+--------------------+
```

box data　*boxdata* も扱える。<が必要。

```
 (<DAT4)  jreplace '/temp/testfile';0
0
 jread '/temp/testfile';0
+-------------+
|+--+--+--+--+|
||6 |3 |19|15||
|+--+--+--+--+|
||10|14|0 |7 ||
|+--+--+--+--+|
||12|17|16|4 ||
|+--+--+--+--+|
||13|2 |1 |9 ||
|+--+--+--+--+|
||18|5 |11|8 ||
|+--+--+--+--+|
+-------------+
```

文字列　文字列も *box* を介して扱える

```
NAM=:'大久保';'中野';'高円寺';'阿佐ヶ谷';'荻窪'
  s: NAM
‘大久保 ‘中野 ‘高円寺 ‘阿佐ヶ谷 ‘荻窪

DAT5=. MNAM,.{@> DAT2
 (< DAT5)  jappend '/temp/testfile'
 a=.  jread '/temp/testfile'; 4
+--------------------------+
|+------------+--+--+--+--+|
||大久保     |6 |3 |19|15||
|+------------+--+--+--+--+|
||中野       |10|14|0 |7 ||
|+------------+--+--+--+--+|
||高円寺     |12|17|16|4 ||
|+------------+--+--+--+--+|
||阿佐ヶ谷|13|2 |1 |9 ||
|+------------+--+--+--+--+|
||荻窪       |18|5 |11|8 ||
|+------------+--+--+--+--+|
+--------------------------+
  }."1 >a
+--+--+--+--+
|6 |3 |19|15|
+--+--+--+--+
|10|14|0 |7 |
+--+--+--+--+
|12|17|16|4 |
+--+--+--+--+
|13|2 |1 |9 |
+--+--+--+--+
|18|5 |11|8 |
+--+--+--+--+
```

5.3.3 EXCEL ファイルの整形

EXCEL から *CSV* *file* を作成する場合の注意事項。
CSV に落とすときは数値のみにし、カンマを取り除き、空欄は適当な数字で埋めておく。

Excel **の見出し取る** 数値のみにする。

comma **を取る** *EXCEL* 側で *comma* を抜く。*comma* があると数字が *2* ー *3* 個に分離する。
EXCEL の書式 → セル → 表示形式 → 数値 の桁区切りのテェックボックスを空欄にする

空欄対策 空欄は *J* でオープンしたときに左詰めされてしまい、右に桁数調整の *0* が入る。従って事前に *EXCEL* 段階で空欄を *0* や *99999* で埋めておく。
(このとき *00,01,09* などを用いると数値化できないことがある*)*

save *csv* 形式を指定して *save* する。

5.3.4 EXCEL ファイルの読み書き *Tara*

特色
- *Tara* は *Bill Ram* に依って開発された
- *EXCEL* や *OpenOffice* の *bif*8 形式の *file* を扱える。*(EXCEL 2003*)
- *EXCEL* や *OpenOffice* の *system* 本体は用いず *file* のみを用いる。

tara **の入手** *J* の *package manager* で *addon* から *table*/*tara* を選んでインストールする。
(インストール参照)

tara.ijs *addon* の *tara.ijs* を先に読み込む
file → *Open* → *addons* → *tables* → *tara* → *tara.ijs*

EXCEL file **の読み込み**
- *tara* での *file* の読み込み
```
DAT=.readexcel '/temp/xxx.xls'
```
- *sheet* を指定するときは *sheet* 名を明示する。*(sheet* 名を英文に *rename* しておく)
```
DAT2=.'sheet1' readexcelstring'/temp/xxx.xls'
```

書き込み
- 最初の定義 *object* の作成
```
 bi=. ''conew 'biffbook'
```
- 書き込み *data*
```
  ] a=. 5 5 $ 25?25
15  4 23  1  6
17 20 18 10  7
 9 21 11 22  0
12  2  5 13 19
24 14 16  3  8
```
- 書き込みの登録 __と *object* を用いる。*adress* は ***0.0*** から
```
writenumber__bi 0 0 ;a    NB. 0 行 0 列から
```

- *file* に書き込む

  ```
  save__bi '/temp/test0.xls'
  ```

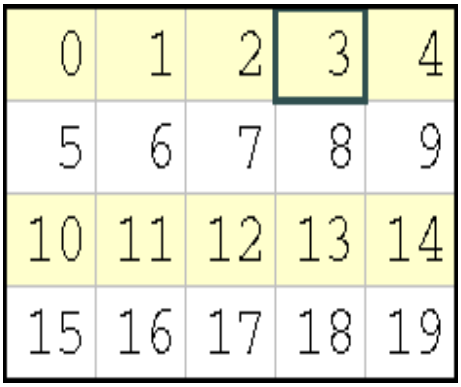
- *append*

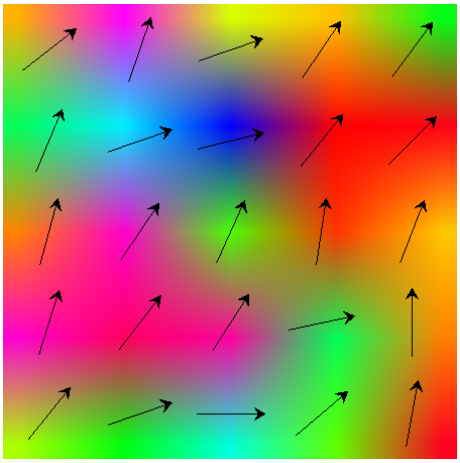
  ```
  writenumber__bi 10 0 ;a  NB. 10 行 0 列から
  save__bi '/temp/test0.xls'
  ```

5.4 Grid

J の *Grid* をマトリクスの簡易ビューアーとして利用する。

```
require 'grid'
 grid i. 4 5
```





J の *demo* に見出しを付ける方法が載っている。（少し複雑）更にはマウスを駆使して簡易 *EXCEL* として用いることもできる

5.5 Viewmat

各種のグラフィックス用マトリクスの表示に用いる
require 'viewmat' NB. 最初の約束
詳細は *LAB* の *viewmat* を参照

- *0 1* のマトリクス　白黒の格子点で表示
- 各ドットを *RGB1600* 万色の色要素で作成したマトリクス
- 複素数のマトリクス。　方向ベクトルで表示
- *bmp* テーブルー *bmp* ファイルの表示

```
a=.  j. 17%~  5 5  $ 25 ?. 100
b=. 13%~ 5 5 $ 25 ?. 50
viewmat  a+b
```

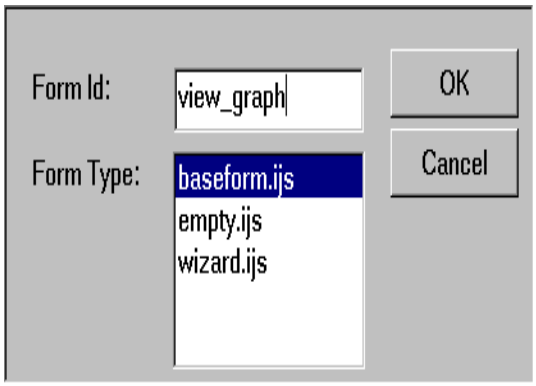
5.6　FORM の作成

5.6.1　EDITOR の使用法

file new ijs で *ijs* の新たな画面を作成する

Form Editor　*ijs* の画面で

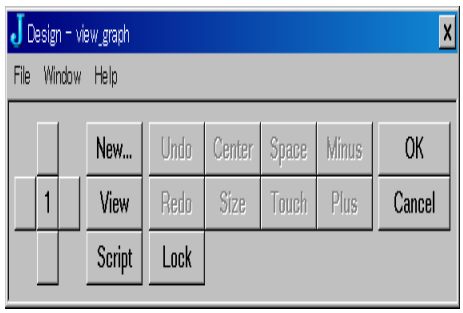
Edit →　*Form_Editor* に *check* する。



form ID　*form ID* を *view_graph* にした。次の画面が現れる

ここで *Forn ID* を *view_graph* としたら起動しなかった。*Script* を手で全て *viewgraph* に直した。*viewgraph* と入れて欲しい

Form Type は通常は *baseform.ijs* を選ぶ。



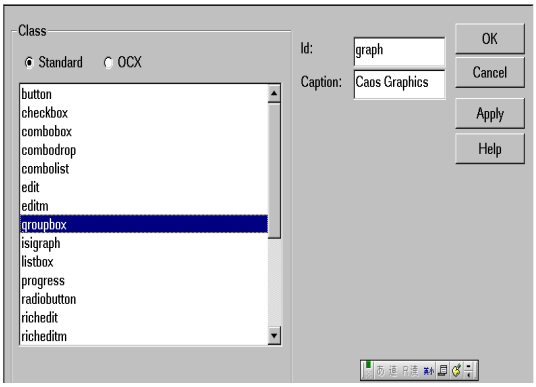
New　*New* で次の画面が現れる。

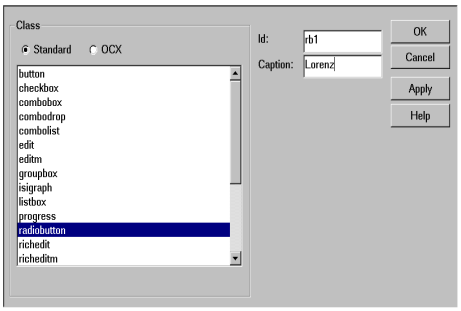
groupbox　*groupbox* を選択し、*ID* を *graph,Caption* を *Caos Graphics* とする。

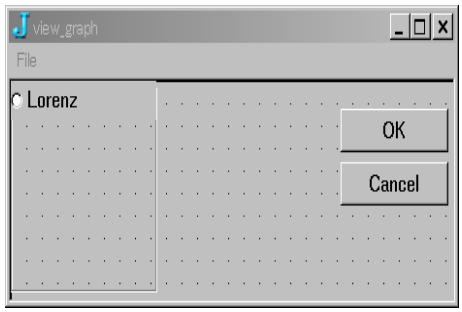
- *groupbox* を *mouse-drag* で少し下げる。*(*次々と左上に出てくるので重なりを避けるため。最後に戻す）
- *groupbox* の大きさはマウスで自由に変更できる
- *groupbox* の中に入れた *radiobutton* は自動で択一選択となる

radiobutton　*New* で *radiobutton* を選択し *ID* を *rb0*　*Caption* を *Lorenz* とする。

- *mouse-drag* で *groupbox* の中に入れる
- *Caption* の文字に着いている箱は *mouse-drag* で文字一杯に縮めた方が邪魔になら



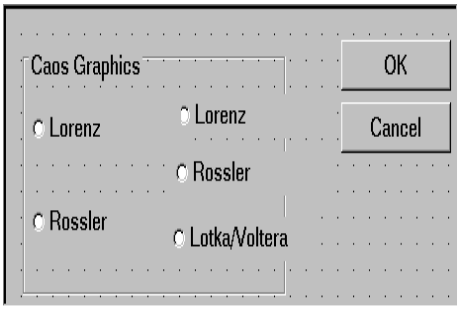


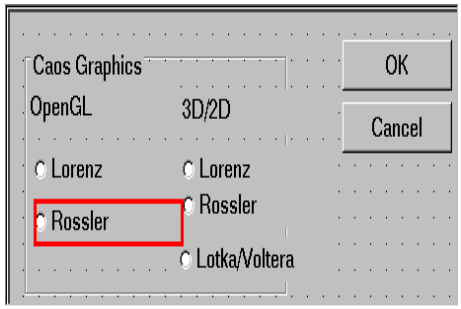


ない

- 後で *Script* からでも画面を修正できるので大ざっぱに並べておく

radiobutton　　次々と *radiobutton* を計 *5* 個作成する。





- *mouse-drag* で *groupbox* の中に配置する
- *id* は　*rb1,rb2,rb3,rb4* とする
- *Caption* は　*Rossler,Lorenz,Rossler,Lotka/Volterra* とする

整形　少し形を整える

static　　見出しの文字を作成する。

- *New* で *static* を選択し *ID* を *d3*、*Caption* を *3D/2D* とする。*mouse-drag* で配置する
- もう一個 *static* を作成する。*ID* は *ogl*,*Caption* は *OpenGl* とする
- *mouse* で整形する

Script　*Script* ボタンを押すと *Script* が見られる。

5.6.2 Script の編集

run *run* は長いので最初に使いやすく編集する。*ijs* 画面で *Ctrl+W* で画面が出る
`run=: viewgraph_run ''` を加えると自動起動する。

DRAW *ok button* を *DRAW* に変更する

```
xywh 137 9 44 12;cc ok button;cn "OK";

xywh 137 9 44 12;cc draw button;cn "DRAW";
```

rb2 *raddiobutton* の *default* の位置を決める

```
viewgraph_run=: 3 : 0
wd VIEWGRAPH
NB. initialize form here
wd 'set rb2 1' NB. set radiobutton
wd 'pshow;'
)
```

DRAW *DRAWbutton* を選択した場合の各描画動作を割り付ける。

```
RDX=:I. (+/ e. rb0,rb1,rb2,rb3,rb4) e. 1 NB. position of radiobutton
```

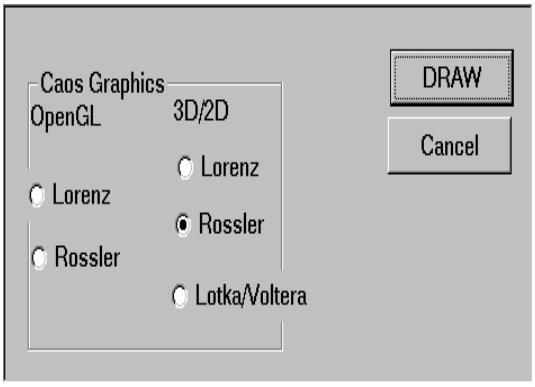
次は添付 *Script* の起動 *parameter*

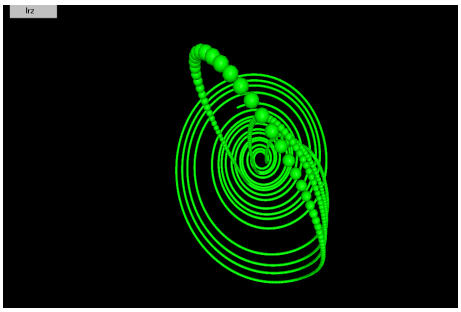
```
viewgraph_draw_button=: 3 : 0
RDX=:I. (+/ e. rb0,rb1,rb2,rb3,rb4) e. 1 NB. position of radiobutton
NB. select. statement is denaild ???
 if. 0 = RDX  do. opengl_run 0 end.
 if. 1 = RDX  do. opengl_run 1 end.
 if. 2 = RDX  do.  10 50 8r3&calc_lorenz plot_lorenz 1 1 1 end.
 if. 3 = RDX  do.  0.2 0.2 14&calc_rossler plot_lorenz 1 1 1 end.
 if. 4 = RDX  do. plot {|:  PAR&vtr ^:(i.100000) FST  end.
)
```

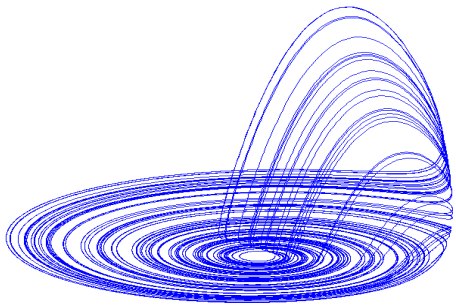
では やってみよう。*Rossler* の *OpenGl* と *3D* である。

5.6.3 Script

```
VIEWGRAPH=: 0 : 0
pc viewgraph;
menupop "File";
menu new "&New" "" "" "";
menu open "&Open" "" "" "";
menusep;
```







```
menu exit "&Exit" "" "" "";
menupopz;
xywh 137 9 44 12;cc draw button;cn "DRAW";
xywh 136 23 44 12;cc cancel button;cn "Cancel";
xywh 8 12 91 60;cc graph groupbox;cn "Caos Graphics";
xywh 8 34 35 11;cc rd0 radiobutton;cn "Lorenz";
xywh 9 47 34 11;cc rb1 radiobutton group;cn "Rossler";
xywh 61 28 31 11;cc rb2 radiobutton group;cn "Lorenz";
xywh 60 40 32 11;cc rb3 radiobutton group;cn "Rossler";
xywh 59 55 50 11;cc rb4 radiobutton group;cn "Lotka/Voltera";
xywh 60 18 27 11;cc d3 static;cn "3D/2D";
xywh 9 19 32 11;cc ogl static;cn "OpenGL";
pas 6 6;pcenter;
rem form end;
)

viewgraph_run=: 3 : 0
wd VIEWGRAPH
NB. initialize form here
wd 'set rb2 1' NB. set radiobutton
wd 'pshow;'
)

viewgraph_close=: 3 : 0
```

```
wd'pclose'
)

viewgraph_cancel_button=: 3 : 0
viewgraph_close''
)



viewgraph_draw_button=: 3 : 0
RDX=:I. (+/ e. rb0,rb1,rb2,rb3,rb4) e. 1 NB. position of radiobutton
NB. select. statement is denaild ???
 if. 0 = RDX  do. opengl_run 0 end.
 if. 1 = RDX  do. opengl_run 1 end.
 if. 2 = RDX  do.  10 50 8r3&calc_lorenz plot_lorenz 1 1 1 end.
 if. 3 = RDX  do.  0.2 0.2 14&calc_rossler plot_lorenz 1 1 1 end.
 if. 4 = RDX  do. plot {|:  PAR&vtr ^:(i.100000) FST  end.
)

run=: viewgraph_run ''
```

第6章

若葉のページ

6.1 K.E.Iverson

コンピューター言語は天才達の頭脳に宿った人類への贈り物である。*FORTRAN* の *John Backus(1924-2007 Philadelphia), COBOL* の *Grace Hopper(1906-1992 New Jersey)*。

Bachus は *IBM* で科学計算のプログラムを手短に書くために *FORTRAN* を作成した。まだ紙テープの時代であり、*IBM* が音頭を取って作成したものでもない。

G.Hopper はカレッジで数学を教えていたが第 *2* 次大戦に海軍に志願し、軍事訓練の後コンピューター部門に配属された。女性初の海軍少将で、その名を冠げた駆逐艦 *Hopper* は *COBOL50* 周年が開催された *San Diego* に回航し祝砲を奏でた。

K.E.Iverson(1920-2004) は *Canada* の *Alberta* 州（ロッキー山脈の北。カルガリーが有名）にノルウエイ移民の子として生まれ、農業に従事していたが第 *2* 次大戦にカナダ空軍のフライトエンジニアとして従軍した。その後、オンタリオ湖畔のクイーンズ大学で数学と物理学を学び、ハーバードで *Haward Aiken* と *Wassily Leontief* の下で研究を続け、准教授であった。*Aiken* は *IBM* の真空管コンピューター *MARKI* の開発者。*Leontief* はロシア出身の経済学者で産業連関表で後にノーベル賞を受賞した。この時期に *Iverson* は真空管の大型機で産業連関表の計算を担当した。

「*500* セクタを *50* セクタに圧縮してレオンテェフ逆行列は *48* 時間うなりを上げて計算出来た。」「当時のハーバードの会計はプログラムに支出を認めなかったので、マトリクス何キロを何ドルで購入した。」

この経験は *K.E.Iverson* が数学とコンピューターを深く考える嚆矢となり、*1960* 年に *IBM* ワトソン研究センターに移って厳密で簡潔な関数記述法を織り込んだ関数型言語 *APL(A Programming Language)* を紙と鉛筆で完成させ、機能が整ってきた *IBM* の大型機に搭載された。

晩年トロントに帰った *Iverson* は、新しいアイデアを *IBM* のスコットランド研究所の *E.Mcdanel* を訪ねて錬成し、*R.Hui* の協力の下に、*APL* をキーボード記号に移し、関数型定義や幾つかの解析関数を盛り込んだ *J* 言語を作成した。

3 人はコンピューター言語の源流にいた。*Backus* は手早く開発した *FORTRAN* が好きになれず、*IBM* で言語記法や関数型言語の研究を続け、*C* の祖先アルゴルにも深くかかわっている。

[*1] *G.Hopper* は初期に *Aiken* の指導を受けている。*Backus* と *Iverson* は *ACM* の *Turing* 賞を受賞し、*Backus* と *Hopper* は数学史の本にも載っている。

[*1] FORTRAN は 90,95 で配列計算、マルチ CPU に対応し、スーパーコンピューターの標準言語である

6.2　実行の優先順位

K.E.Iverson はチューリング賞受賞講演「思考の道具としての表記法 *(1979)*」で次のように述べている。

> カジョリは「数学的表記法の歴史第 *2* 巻（*1929*）」の中で実行の順序についての規則に言及すらしていないが、良く知られている優先順位 $(\hat{}, \times\div, +-)$ の動機は括弧無しで多項式が表されるようにしたと仮定したことは無理がないと思われる。

K.E.Iverson は数学の実行順位ではなく全ての演算記号は平等で実行順位を「原則右から左」に統一した。*Iverson* の永年の表記法の研究の結果、*2* 項関数を含む関数型記述ではでは優先順位を定めない方が極めて簡潔に表記できると考えたからである。明示型では右から左の原則が貫かれている。

J で新たに採用した関数型記述では引数を一々記述しないで、関数間の演算を行うという発想で構成されており、演算順位は関数の組み方[*2]に依存する。

6.3　明示型と関数型

Tacit（関数型）は *K.E.Iverson* が *J* で導入した簡易で *elegant* な関数記述法。

6.3.1　bond atop

.

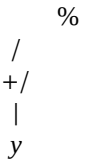
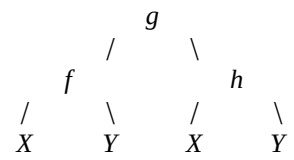
bond &	数字と *primitive* を接続するのは& *(primitive* と　*primitive* の接続もできる） e^1, e^2, e^3	``` ^&1 2 3] 1x1 2.71828 7.38906 20.0855 1x1 ^ 1 2 3 2.71828 7.38906 20.0855 1x1 1x2 1x3 2.71828 7.38906 20.0855```
atop @	*primitive* と *primitive* を接続 $\sum 2^2 + 3^2 + 4^2$ $\sqrt{\sum 2^2 + 3^2 + 4^2}$	``` (+/@:*:) 2 3 4 29 %:@(+/)@:*: 2 3 4 5.38516``` +/の（右との）接続にはランクを外した@:を用いる。

[*2] compose=作詞 · 作曲

明示型	明示型は引数を明確に記述する 実行は右から左	`calc0=: 3 : ' %: +/ *: y'` `   calc0 2 3 4` `5.38516`

6.3.2 fork hook

.

fork *3* の動詞	$\frac{\sum x}{n}$ *fork* の単項を用いる 	`mean=: +/ % #` `NB. 明示型` `meane=:3 : '(+/y)% # y'` `   mean >:i.10` `5.5` `   meane >:i.10` `5.5`
	移動平均 *3* 項の例 *Infix* 「\」の機能を用いて一つずつずらした組み合わせを作成する *box* 毎の合計を左引数で割る 	`  3 <\i.6` `+-----+-----+-----+-----+` `|0 1 2|1 2 3|2 3 4|3 4 5|` `+-----+-----+-----+-----+` `   3 +/\ i.6` `3 6 9 12` `   mav=: +/\ % [` `   3 mav i.6` `1 2 3 4`

	偶数の場合は図に表せないので移動平均した値を再度 *2* 項ずつ移動平均するか左引数に *1* を加える	`13 mav i.20` `6 7 8 9 10 11 12 13` `2&mav 12&mav i.20` `6 7 8 9 10 11 12 13`
Hook *2* の動詞	*Fork* 程頻繁には使用しないが便利な機能 $x - \bar{x}$ `  _` `/ \` `y +/%#` `  |` `  y`	`mean >:i.10` `5.5` `dev=: - mean` `dev >:i.10` `_4.5 _3.5 _2.5 _1.5 _0.5` `0.5 1.5 2.5 3.5 4.5` 明示型では `deve0=: 3 : 'y - (+/%#) y'` `deve1=: 3 : 'y - (+/ y) % # y'` `deve2=: 3 : 'y - mean y'`
(Hook) 黄金比 *Golden ratio*	黄金比（連分数による） `    1` `1 + ----------------` `          1` `  1 + ------------` `            1` `    1 + ---------` `              1` `       1 + -----` `                ...`	`4 4 $(+%)/\ 16#1 NB. 1 1..1 1` `1 2 1.5 1.66667` `1.6 1.625 1.61538 1.61905` `1.61765 1.61818 1.61798 1.61806` `1.61803 1.61804 1.61803 1.61803`

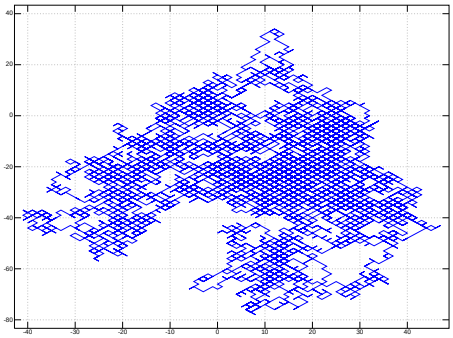
ランダムウオーク

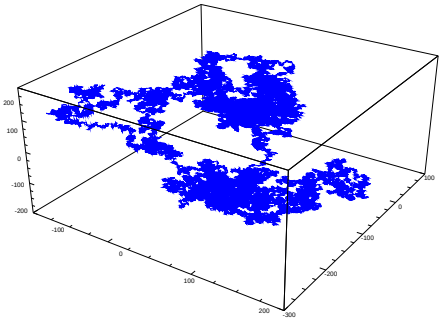
2D ランダムウオークと *3D* ランダムウオーク

```
require 'plot'
rw=: 3 : ' +/\((-: y)> ?~ y) { 1 _1'
rw2=:  rw ; rw
rw3=: rw;rw;rw
```

```
plot  rw3 100000
pd 'eps /temp/rand3.eps'
```

```
plot rw2 10000
```
NB. いろいろに変化する。





6.3.3 Explicit の Tacit への変換

Explicit を *Tacit* に変換する。

まずは *Explicit* をしっかりと作成しよう。

```
  rw=: 3 : ' +/\((-: y)> ?~ y) { 1 _1'
  13 : ' +/\((-: y)> ?~ y) { 1 _1' NB. 13 : 0 で変換
[: +/\ 1 _1 {~ -: > ?~                NB. out of Tacit

rwt=:[: +/\ 1 _1 {~ -: > ?~ NB. define yourself in Tacit
  plot (rwt;rwt) 10000      NB. fine!
```

6.3.4 power 接続詞

.

fibonacci 数を求める。

1. *repeat* は関数型の`^:(n)` を用いる。
2. 内積 `+/ . *`

```
fib=: (0 1,:1 1)&( +/ . *)
```

fibonacci 数を求める。

$$F_n = \begin{cases} 0 & if\ n = 0 \\ 1 & if\ n = 1 \\ F_{n-1} + F_{n-2} & if\ n > 1 \end{cases}$$

Matrix form:

$$\begin{bmatrix} F_{k+1} \\ F_{k+2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} F_k \\ F_{k+1} \end{bmatrix}$$

```
   fib ^:(i.5) 1 1
1 1
1 2
2 3
3 5
5 8
```

6.4　ニュートン法

ニュートン法は、

$$x - \frac{f(x)}{f'(x)}$$

の反復計算を行うものである。*J* は微分演算子 *D.* があるので *APL* より容易にスクリプトが作れる。次のスクリプトは *J* の定番である。

ニュートン法は微分演算子を用いてシンプルに定義できる。動詞を左パラメーターに取るので副詞 *(1 : 0)* で定義しなければならない。ランクはベクトルを引数に取ることができるように **("0)** とする。**(^:_)** は収束まで計算するが失敗すると無限ループに陥る。最初は **(^:100)** 程度で適当に打ち切る方が無難。

$y = -5 - 2x + x^3$ をニュートン法で解いてみよう。

多項式は *p.* を用いて定義する。

```
 2 5 $  _5 _2 0 1&p. new_1 i:5
2.09455 2.09455 2.09455 2.09455 2.09455
2.09455 2.09455 2.09455 2.09455 2.09455

NB. Newton method
 new_1=: 1 : ' ] - x % x D.1' (^:_)("0)
```

関数の定義部分のみに明示型を用いても良い。（明示型の内部に反復部分を記述しない）

6.5　レオンチェフ逆行列

K.E.Iverson の下にレオンチェフから産業連関表の計算が持ち込まれた。

産業連関表の肝要はレオンチェフ逆行列である。正則行列なので計算は現代の *PC* で 1000×1000 でも可能と実際にデモをした時に、橋梁の専門家が「その規模の逆行列の計算は私の学生時代は *T* 大まで九州からパンチカードを送って計算して貰った。間違えると大変だった」と感激していた。

Purchased from	*Manufac turing*	*Agriculture*	*Service*
Manufact.	0.5	0.4	0.2
Agriculture	0.2	0.3	0.1
Service	0.1	0.1	0.3

Example　最後の行はラミネート「,:」で連結する

```
C0=: 0.5 0.4 0.2,0.2 0.3 0.1,:0.1 0.1 0.3
```

投入産出行列と単位行列 `=/~i. # C0`	$C0$ ‖ I 0.5 0.4 0.2 ‖ 1 0 0 0.2 0.3 0.1 ‖ 0 1 0 0.1 0.1 0.3 ‖ 0 0 1
$(I-A)$	`(=/~i. 3)-C0` `0.5 _0.4 _0.2` `_0.2 0.7 _0.1` `_0.1 _0.1 0.7`
$(I-A)^{-1}$	`%. (=/~i. 3)-C0` `2.96296 1.85185 1.11111` `0.925926 2.03704 0.555556` `0.555556 0.555556 1.66667`

```
leon_inv=:3 : '%.(=/~ i. # y)-y'
leon_invt=: [: %. =/~@i.@# - ]
cr=: %.}."1  NB. cramer method
```

レオンチェフ逆行列は投入産出行列がもたらす波及効果を計測する。レオンチェフ逆行列はマルコフ行列であり、マルコフ行列の反復収束と同じ結果となる

	demand	*Inputs*
Finaldemand	d	Cd
(*Intermediatedemand*)		
1*stround*	Cd	$C(Cd)=C^2d$
2*ndround*	C^2d	$C(C^2d)=C^3d$
3*rdround*	C^3d	$C(C^3d)=C^4d$
	⋮	⋮

$$x = d + Cd + C^2d + C^3d + \cdots = (I + C + C^2 + C^3 + \cdots)d$$
$$(I-C)(I + C + C^2 + C^3 + \cdots + C^m) = I - C^{m+1}$$
$$(I-C)^{-1} \simeq I + C + C^2 + C^3 + \cdots + C^m$$

$I + C + C^2 + C^3 + \cdots + C^m$

```
   (=/~i.3)+  +/>  C0&(+/ . *)^:(i.24) C0
 2.95994  1.84882  1.10916
0.924412  2.03552 0.554578
0.554578 0.554578  1.66604
```

最終需要が示されると各部門の算出額が求められる

$x = (I - C)^{-1}d$

final demand(m. a. s.)
= 50 30 20

```
(%. (=/~i.3)-C0) +/ . * 50 30 20
225.926 118.519 77.7778

   50 30 20 %. (=/~i.3)- C0
225.926 118.519 77.7778
```

```
((=/~i.3)- C0),. 50 30 20
 0.5 _0.4 _0.2 50
_0.2  0.7 _0.1 30
_0.1 _0.1  0.7 20
```

```
clean cr((=/~i.3)- C0),.50 30 20
1 0 0 225.926
0 1 0 118.519
0 0 1 77.7778
```

clean は numeric.ijs にある微少誤差を取る関数
require 'numeric'

David C. Lay [Linear Algebra and its Applications] 2nd.ed. Addison-Weslay 2000

6.6　逆行列と最少自乗法

その後、*K.I.Iverson* は *IBM* に移って配列計算言語 *APL* を開発した。*APL* や *J* では縦長の行列の逆行列も求められる。

```
reg_t=: %. 1&,.@]          NB. Tacit
reg=: 4 : 'x %.1,.y'    NB. Explicit
```

独立変数を縦長のマトリクスで与えればこの *1* 行で多変数（重相関）回帰も可能である。

縦長の行列の逆行列は今では他の多くの言語も数値演算ライブラリで採用している。

横長の行列の逆行列はムーア・ペンロース逆行列になり、解は一位には定まらない。（線形計画ではスラッグ変数を入れる）

◇ 連立方程式を**クラメール法**で解く。左に単位行列が出たときは良く解けている（数値計算であるので微細誤差が残ることもある。ここに幾らかの数値が残る場合は行列が相当歪んでいる）

$$\begin{cases} x & +2y & -3z & = 15 \\ x & +y & +z & = 12 \\ 2x & -y & -z & = 0 \end{cases}$$

```
cr=: %.}:"1   NB. Cramer Method
```

```
1 2 _3 15,1 1 1 12 ,: 2 _1 _1 0
```

```
1  2 _3 15
1  1  1 12
2 _1 _1  0

 cr  1 2 _3 15,1 1 1 12 ,: 2 _1 _1 0
1 0 0 4
0 1 0 7
0 0 1 1
```

6.7　非対称行列の固有値計算　ルベリエ・ファディーエフ法

対称行列では固有値は異なる実数であり、固有ベクトルは直交することが保証されている。しかし、非対称行列では固有値に重根や複素数が現れることが珍しくなくい。

2007 年夏に固有値問題のメーリングリストに参加したときに、非対称行列の美しい解法としてルベリエ・ファデェーエヴァ法のスクリプトを *John Randell*（*NJ* 州 *Rutgers Univ.* の数学　情報科学の准教授）が紹介してくれた。特性方程式 *(characteristic equation)* を用いる方式である。

$AX = \lambda X$

λ : マトリクス　A の固有値
X : 固有値　λ　に対応する固有ベクトル

$(A - \lambda I_n)X = 0$

$$f(\lambda) = |A - \lambda I_n| = 0$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

$$c|A - \lambda I_n| = \begin{bmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = 0$$

更に展開すれば λ の n 次の代数方程式となる。

$$f(\lambda) = (-1)^n(\lambda^n + c_1\lambda^{n-1} + \cdots + c_{n-1}\lambda + c_n) = 0$$

6.7.1　LF 法のアルゴリズム

$det(A - \lambda I_n) = 0$

λ は多項式　$p(\lambda) = det(A - \lambda I_n)$ の根である。

Leverrir-Faddeev 法は、多項式　$p(\lambda)$ の係数 c_k を求める優れた方法である。

$$p(\lambda) = \lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + \cdots + c_{n-t}\lambda^2 + c_{n-1}\lambda + c_n$$

マトリクスのトレース（対角行列を足した合計）を*Tr[A]*とする

$$Tr[A] = a_{1,1} + a_{1,t} + \cdots + a_{n,n}$$

補助マトリクス $(Bk)_{k=1}^{n}$ を作成する。

$$\begin{aligned}
B_1 &= A & \rightarrow \quad p_1 &= Tr[B_1] \\
B_2 &= A(B_1 - p_1 I) & \rightarrow \quad p_2 &= \frac{1}{2}Tr[B_1] \\
&\cdots & &\cdots \\
B_k &= A(B_{k-1} - p_{k-1} I) & \rightarrow \quad p_k &= \frac{1}{k}Tr[B_k] \\
B_n &= A(B_{n-1} - p_{n-1} I) & \rightarrow \quad p_n &= \frac{1}{n}Tr[B_n]
\end{aligned}$$

次の多項式を得る。これをニュートン法などで解けば固有値がめられる。

$$p(\lambda) = \lambda^n - p1\lambda^{n-1} - p_2\lambda^{n-2} - \cdots - p_{n-t}\lambda^2 - p_{n-1}\lambda - p_n$$

次の様に A の逆行列も求められる。

$$A^{-1} = \frac{1}{p_n}(B_{n-1} - p_{n-1} I)$$

数値例

$$A = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{pmatrix}$$

■特性多項式　特性多項式から, 固有値を求める

$$A = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{pmatrix}$$

$$B_1 = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{pmatrix} \qquad NB.A1 \text{ と同一}$$

$p_1 = Tr[B_1] = 6$　　*NB. B1* の対角行列の合計

$$B_2 = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{pmatrix} \times \left[\begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{pmatrix} - 6 \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right] = \begin{pmatrix} -6 & 1 & 3 \\ 3 & -8 & -3 \\ -1 & 1 & -8 \end{pmatrix}$$

NB. $B_2 = A \times (B_1 - 6I)$

$p_2 = \frac{1}{2} Tr[B_2] = \frac{1}{2} \times -22 = -11$　　*NB.* $\frac{1}{2} \times B_2$ の対角行列の合計

$$B_3 = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{pmatrix} \times \left[\begin{pmatrix} -6 & 1 & 3 \\ 3 & -8 & -3 \\ -1 & 1 & -8 \end{pmatrix} - (-11) \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right] = \begin{pmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{pmatrix}$$

NB. $B_3 = A \times (B_2 - \_11I)$

$p_3 = \frac{1}{3} Tr[B_3] = \frac{1}{3} \times 18 = 6$

NB.$\frac{1}{3} \times B_2$の対角行列

特性方程式を作成する

$P[\lambda] = \lambda^3 - \sum_{i=1}^{3} p_i \lambda^{n-1}$

$p[\lambda] = -6 + 11\lambda - 6\lambda^2 + \lambda^3$

NB. $- p_3 - p_2\lambda - p_1\lambda^2 + \lambda^3$

特性方程式 $\rightarrow -6 + 11x - 6x^2 + x^3$

```
  p. _6 11 _6 1
+-+-----+
|1|3 2 1|
+-+-----+
```

逆行列

$A^{-1} = \frac{1}{p_3}(B_{3-1} - p_{3-1}I)$

$$A^{-1} = \frac{1}{6} \times \left[\begin{pmatrix} -6 & 1 & 3 \\ 3 & -8 & -3 \\ -1 & 1 & -8 \end{pmatrix} - \begin{pmatrix} -11 & 0 & 0 \\ 0 & -11 & 0 \\ 0 & 0 & -11 \end{pmatrix} \right] = \frac{1}{6} \times \begin{pmatrix} 5 & 1 & -3 \\ 3 & 3 & -3 \\ -1 & 1 & 3 \end{pmatrix} = \begin{pmatrix} 5/6 & 1/6 & -1/2 \\ 1/2 & 1/2 & -1/2 \\ -1/6 & 1/6 & 1/2 \end{pmatrix}$$

NB.$A^{-1} = \frac{1}{P_3} \times (B_2 - P_2 I)$

John Randoll に啓発されて *LF* 法のスクリプトを作ってみた。

```
tr=: (<0 1)&|: NB. diag
NB. umatrix=: (=/~)@i.@#
char_lf=: 3 : 0
ANS=. TR_SUM=. +/ tr MAT=. y NB. sum of trace
UMAT=. =/~ i. # y
for_LF. i.<: # y do.
MAT=. y +/ . * MAT - UMAT *  TR_SUM
```

```
TR_SUM=. (% 2+LF)* +/  tr MAT
ANS=. ANS,TR_SUM
end.
(p. POL), (<POL=.(-ANS),1)
)
```

ループは $n-1$ 回。*LF* がカウンターになっている。ここで $\frac{1}{2}$ からの p_k を *2+LF* とする。係数は *p.* 用に次数の高い方を右にする。最高次の λ の係数 p^n は *1* である。

```
   char_lf a0
+-+-----+----------+
|1|3 2 1|_6 11 _6 1|
+-+-----+----------+
```

固有値　*3*　*2*　*1*
特性方程式 $-6+11x-6x^2+x^3$
解は *p.* で求めた固有値を先にした。[*3]

[*3] 最初のマスの 1 は p. が多項式を解いたときのループを表している。精度を見るとき以外は特に必要としない

第章

Indices

.1 アルファベット順索引

.2 あいうえお順索引

.3 プリミティブの機能一覧

■= イコール

	説明 (英語)	説明 (日本語)	品詞	章節
$=Y$	$Self-classify$	自己分類を行なう	動詞	1.2.1
$X=Y$	$Equal$	左右が等しければ 1 (論理演算)	動詞	1.2.1
$X=.Y$	$Is(Local)$	局所定義	接続詞	1.6.1
$X=:Y$	$Is(Gloval)$	大局定義	接続詞	1.6.1

■. ＜より小

$<Y$	Box	ボックスで囲む	動詞	1.4
$X<Y$	$Lesserthan$	X が Y より小なら 1 (論理演算)	動詞	1.2.1
$<.Y$	$Flooroval)$	切り捨てた整数値	動詞	1.1.1
$X<.Y$	$Lesso f$	小さいほうの値を与える	動詞	1.1.1
$<:Y$	$Decrement$	数値から 1 を引く	動詞	1.1.1
$X<:Y$	$LessorEqual$	小さいか等しい (論理演算)	動詞	1.2.1

■. ＞より大

$<Y$	$Open$	ボックスを開く	動詞	1.4
$X<Y$	$Larger\ than$	X が Y より大なら 1(論理演算)	動詞	1.2.1
$<.Y$	$Ceiling$	切り捨てた整数値	動詞	1.1.1
$X<.Y$	$Largeo f$	大きいほうの値を与える	動詞	1.1.1
$<:Y$	$Increment$	数値に 1 を加える	動詞	1.1.1
$X<:Y$	$Large\ or\ Equal$	大きいか等しい (論理演算)	動詞	1.2.1

■_ アンダーバー

$_Y$	$NegativeSign$	マイナス符号	動詞	1.1.3
$_$	$Infinity$	無限大	副詞	1.1.3
$_.$	$Indefinite$	不定	副詞	1.1.3
$_:$	$Infinity$	無限大	動詞	1.1.3

■+ プラス

$+Y$	$Conjugate$	共役複素数	動詞	1.1.1(5)
$X+Y$	$Plus$	加算	動詞	1.1.1
$+.Y$	$Real/Imaginary$	複素数の実部と虚部を与える	動詞	1.1.5
$X+.Y$	GCD	最大公約数	動詞	1.1.1
$X+.Y$	Or	論理和 (論理演算)	動詞	1.2.2
$+:Y$	$Double$	２倍にする	動詞	1.1.1
$X+:Y$	$Not-Or$	否定論理和 (論理演算)	動詞	1.2.2

■− マイナス

$-Y$	$Negative$	逆符号	動詞	1.1.1
$X-Y$	$Minus$	減算	動詞	1.1.1
$-.Y$	Not	論理否定	動詞	1.2.2
$X-.Y$	$Less$	Y に含まれない X の要素を出力	動詞	1.2.2
$-:Y$	$Halve$	半分にする	動詞	1.1.1
$X-:Y$	$Match$	一致すれば 1(論理演算)	動詞	1.2.1

■* スター

∗*Y*	*Signum*	符号複素数	動詞	1.1.1
X ∗ *Y*	*Multiply*	乗算	動詞	1.1.1
∗.*Y*	*Length/Angle*	複素数の極座標表示を与える	動詞	1.1.5
X ∗ .*Y*	*LCM*	最小公倍数	動詞	1.1.1
X ∗ .*Y*	*And*	論理積 (論理演算)	動詞	1.2.2
∗ : *Y*	*Square*	平方する	動詞	1.1.1
X∗ : *Y*	*Not – And*	否定論理積 (論理演算)	動詞	1.2.2

■% パーセント

% Y	*Reciprocal*	逆数	動詞	1.1.1
X % Y	*Divide*	除算	動詞	1.1.1
%. Y	*Matrix – Inverse*	(一般化) 逆行列を与える	動詞	1.1.6
X %. Y	*Matrix – Devide*	行列の除算	動詞	1.1.6
%: Y	*Square – Root*	平方根	動詞	1.1.1
X %: Y	*Root*	累乗根	動詞	1.1.1

■ ^ ハット

^ Y	*Exponential*	指数	動詞	1.1.1
X ^/Y	*Power*	*X* の数値の *Y* 乗	動詞	1.1.1
^. Y	*Natural – Log*	自然対数	動詞	1.1.1
X ^. Y	*Logarithm*	*X* を底とする対数値	動詞	1.1.1
^: Y	*Power*	反復計算	接続詞	1.5.3
X ^: Y	*Power*	反復計算)	接続詞	1.5.3

■. $ ドル

$ Y	*Shape of*	*Y* の形を与える	動詞	1.3.2
X $ Y	*Shape*	*X* で指定した形に	動詞	1.3.2
$.	*Sparce*	大型粗行列	動詞	1.3.2
X $: Y	*Self – Reference*	再帰の繰返し演算	動詞	1.5.3

■~ チルド

~ *Y*	*Reflex*	両側化	動詞	1.3. 2
X ~ *Y*	*Passive Evoke*	*X* と *Y* の交換	動詞	1.3.2
~ .*Y*	*Nub*	重複した要素を排除する	動詞	1.2.1
~: *Y*	*NubSieve*	重複した要素に 0 を与える	接続詞	1.2.1
X ~: *Y*	*Not – Equal*	等しくない (論理演算)	接続詞	1.2.1

■| 縦棒

|*Y*	*Magnitude*	(実数や複素数の) 絶対値	動詞	1.1.1
X|*Y*	*Residue*	整数の除算の剰余	動詞	1.1.1
|.*Y*	*Reverse*	アレイ (ベクトルや行列) の逆順	動詞	1.3.2
X|.*Y*	*Rotate*	アレイ (ベクトルや行列) の回転	動詞	1.3.2
| : *Y*	*Transpose*	アレイ (ベクトルや行列) の転置	動詞	1.3.2
X| : *Y*	*Transpose*	*X* で指定したセルの転置	動詞	1.3.2

■. . ピリオッド

−/. ∗ *Y*	*Determinant*	一般行列式	接続詞	1.1.6
+/. ∗ *Y*	*Dot − Product*	一般内積	接続詞	1.1.6
X..*Y*	*Even*	偶数部分の取り出し	接続詞	1.6.1
X. : *Y*	*Odd*	奇数部分の取り出し	接続詞	1.6.1

■: コロン

u : .*v*	*Obverse*	逆定義	接続詞	1.6.1
u :: *v*	*Adverse*	エラーが無ければ *u*、あれば *v*	接続詞	1.5.3

■, , コンマ

, *Y*	*Ravel*	リスト化	動詞	1.3.2
X, *Y*	*Append*	アイテム同士の結合	動詞	1.3.2
, .*Y*	*RavelItem*	テーブル化	動詞	1.3.2
X, .*Y*	*Stitch*	高いランクの方向に結合	動詞	1.3.2
, : *Y*	*Itemize*	アレイのランクを 1 つ増加	動詞	1.3.2
X, : *Y*	*Laminate*	高いランクの形に合わせて結合	動詞	1.3.2

■; セミコロン

; *Y*	*Raze*	リストにほぐす	動詞	1.4
X; *Y*	*Link*	ボックスで囲んで接続	動詞	1.4
; .*Y*	*Cut*	文字列やブロック等を区切る	接続詞	1.4
X; .*Y*	*Cut*	*X* に応じて文字列等を区切る	接続詞	1.4
; : *Y*	*WordFormation*	文字列の単語を区分	動詞	1.7
X; : *Y*	*Sequential Maschine*	単語を区分	動詞	1.7

■# シャープ

# Y	*Tally*	アレイのアイテム数	動詞	1.3.2
X # Y	*Copy*	*X* で指定した個数をコピーする	動詞	1.3.2
#. Y	*Base*2	2 進数の 10 進数化	動詞	1.2.3
X ;. Y	*Base*	*X* で指定した底で 10 進数値	動詞	1.2.3
#: Y	*Antibase*2	10 進数の 2 進数化	動詞	1.2.3
X #. Y	*Antibase*	10 進数の *X* 進数化	動詞	1.2.3

■! 感嘆符

!*Y*	*Factorial*	階乗の値を求める	動詞	1.1.9
X!*Y*	*Out − of*	2 項係数を求める	動詞	1.1.9
!.*Y*	*Fit*/*Customize*	^ と数値を接続した階乗型関数	接続詞	1.6.1
X! : *Y*	*Foreign*	外部接続詞	動詞	第 3 章

■/ スラッシュ

u/*Y*	*Insert*	演算子をアイテム間に挿入する	副詞	1.4
Xu/*Y*	*Table*(*Insert*)	左右のクロス演算 (一般外積)	副詞	1.1.6
/.*Y*	*Oblique*	対角線上に演算子を挿入する	副詞	1.3.6
X/.*Y*	*Key*(*Append*)	*Y* の *X* に等しい部分に作動する	動詞	1.3.5
/ : *Y*	*Gradeup*	アレイの昇順のインデクス	動詞	1.3.5
X/ : *Y*	*Sort*	昇順ソート	副詞	1.3.5

■\ 逆スラッシュ

u & v	*Atop*	動詞・名詞の接続	接続詞	1.5.1
u &. v	*Under*	動詞の接続 (アンダー)	接続詞	1.5.1
u &: v	*Appose*	動詞・名詞の接続	接続詞	1.5.1
u &.: v	*Under*(*Dual*)	動詞・名詞の接続	接続詞	1.5.1

■? 疑問符

?*Y*	*Roll*	重複を許した乱数の発生	動詞	1.1.7
X?*Y*	*Deal*	重複を許さぬ乱数の発生	動詞	1.1.7
?.*Y*	*Roll*(*fixseed*)	重複を許した乱数 (シード固定)	動詞	1.1.7
X?.*Y*	*Deal*(*fixseed*)	重複を許さぬ乱数 (シード固定)	動詞	1.1.7

■A .

A.Y	*AnagramIndex*	組合せ	動詞	1.1.9
XA.Y	*Anagram*	辞書式順序	動詞	1.1.9
a.Y	*Alphabet*	等の (256) の生成	名詞	1.7
a :	*Ace*	空のボックス	名詞	1.7
ad	*ad*	度数での複素数表示	名詞	1.1.2
ar	*ar*	ラジアンでの複素数表示	名詞	1.1.2

■B,C .

b. Y	*Boolean*	ブール数へ変換	副詞	1.1.2
u b. n	*Basic*	ランク、逆関数、原始定義	副詞	1.2.3
n b m	*Character*	mのn進数の 10 進数の値	名詞	1.1.2
C.Y	*Cycle*	巡回置換	動詞	1.1.9
X C. Y	*Permute*	置換	動詞	1.1.9

■D,E .

u D. n	*Derivative*	多項式の微分係数を与える	接続詞	1.1.8
X D : *Y*	*SecantSlope*	平均変化率を与える	接続詞	1.1.8
u d. n	*Derivative*	ランク 0 に固定「*uD.n*」と同じ	接続詞	1.1.8
X E. Y	*Member*	*Y* の *X* を含む位置に 1 を与える	動詞	1.2.1
e. Y	*Razein*	アトムの含合積「– /〜」と同じ	動詞	1.2.1
X e. Y	*MemberofInt.*	*Y* が *X* を含めば 1(論理演算)	動詞	1.2.1
x e n	*Exponent*	数値の指数表示	名詞	1.1.2

■F,H,I .

X f. Y	*Fix*	定義関数 (代動詞) の固定	副詞	1.5.6
H. Y	*Hypergeometric*	超幾何級数	接続詞	1.1.8
i. Y	*Integer*	負でない整数列の生成	動詞	1.3.1
X i. Y	*Indexof*	*Y* の要素の *X* の位置インデクス	動詞	1.3.1
i : *Y*	*Steps*	マイナスを含めた整数列を生成	動詞	1.3.2
i : *Y*	*Index of Last*	後の位置インデックス	動詞	1.2.1

■J,L,I .

j.Y	*Imaginary*	９０度回転した複素数を与える	動詞	1.1.5
X j. Y	*Complex*	複素数生成	動詞	1.1.5
mjn	*Complexnumber*	複素数	名詞	1.1.5
L.n Y	*Level*	ボックスのレベルを表示する	動詞	1.4
u L : *n Y*	*Level At*	ボックスのレベルでの演算	接続詞	1.4

Quick Reference for J Language

Edited by J Research Group(Japan APL Association)

Contents