Numerical Python

Paul F.Dubois*, Konrad Hinsen†, and James Hugunin‡
(日本語訳：山本　昇§)

日本語版：September 16, 1999

Abstract

この文書はP.F. Duboisらによる“Numerical Python”の一部をN. Yamamotoが邦訳したものである。省略された部分はPython言語の一般的な解説であり、これについては既に日本語で解説が出ているので、そちらを参照して欲しい。

1 Python

省略

2 Introducing Python

省略

3 数値機能拡張

Pythonの数値機能拡張はmailing-listによるInternet上の共同作業の結果である。Python共同体はこれまで、幾つかの“Special Interest Group”(SIG)を設けてきた。....

以下省略

Pythonに新しいオブジェクト型を付け加えることは、C 言語による拡張でも、Python自身のオブジェクト指向機能によっても可能である（しかもそれは非常に簡単である)。数値機能拡張の中心は、新しいオブジェクト型array(配列)とその配列の操作を高速に行うための新しいオブジェクト型“ofunc”(optimized function)を導入するC 言語による機能拡張である。Pythonは言語中の, +,-,*,/などの演算子を機能拡張の中で拡張する機能を有している。

3.1 配列(array)の生成

これから後の説明では、ユーザは既に

```
from Numeric import *
```

を実行してあるものと仮定する。これによって数値機能拡張の全機能を個別に宣言することなく使える様になる。基本的な配列の「生成子」は関数arrayである。この関数はPythonのsequence的な振る舞いを持つどの型のオブジェクトでも適当な型を持つ配列に変換する。Pythonの実数型はC 言語の“double”型に対応している（すなわち多くの場合64bitsのデータ幅をもつ。）Pythonの整数型はC 言語の“long int”型に対応している。数値機能拡張のarray型はさらにその他の整数型、複素数型、浮動精度型もPythonのオブジェクトのアレイと同様にサポートしている。幾つかの配列生成の例を示す。

*Paul F. Dubois is a mathematician at Lawrence Livemore National Laboratory.
†Konrad Hinsen is a physicist in the Department of Chemistry, University of Montreal.
‡James Hugunin is a graduate student in the Laboratory for Computer Science, Massachusetts Institute of Technology.
§文部省高エネルギー加速器研究機構、加速器研究施設

array([1, 2, 3]) ⇒ 整数の配列

array([1, 2.3, 4]) ⇒ 実数の配列（一つの要素が実数なので）

array([1, 2j, 3.j]) ⇒複素数の配列（一つの要素が複素数）

当たり前のことではあるが、実際の応用ではarrayの引数はこの例の様に直接入力されることはなく、何らかの計算の結果あるいはファイルからの入力値であることが普通である。よく使われる配列を用意するための関数が幾つか用意されている。

zeros(2,3,4) すべての要素が0で配列の次元が(2,3,4)である配列を作る。

zeros(2,3,4,Integer()) 上と同様の整数配列を作る。この例で使われた様な一連の関数はFortran 90のkind specifiersと同様に、異なった型と精度を持った配列を可能にする。(例：Float(32))

ones() zeros()同様であるがすべての要素を1とする。

arange(n) 0からn-1を要素とする長さｎの整数配列を生成する。

fromFunction([2, 3, 4 , lambda i, j, k: 100*i - 10*j + k])配列の次元が(2,3,4)である配列を生成する。その要素はlambda式で指定された式の値となる。もちろんlambda式の替りに、三つの引数を持つ関数の名前を与えることも出来る。

3.2 オブジェクトとしての配列

実のところarrayはオブジェクトである。実のところPythonにおいてはすべてはオブジェクトである[1]。辞書（Mapping)、リスト、関数、トレース・バック、文字列、等々すべてはオブジェクトである。であるから、それらのものは属性（データ・メンバあるいは少なくともデータ・メンバの様に振る舞うもの）とメンバ関数(method)を持っている。オブジェクト指向プログラミング言語でよく使われる”ドット”表現がこれらの属性やメンバ関数をアクセスするために使われる。例えば、リスト・データに対して二つのメンバ関数”append”と”reverse”が存在する。

```
>>> x=[1,2,3]
>>> x.append(4)
>>> x.reverse()
>>> print x
[4,3,2,1]
```

3.3 Shapes:配列の次元(形？)

夫々の配列は配列の型(shape)をもっている。この型(shape)は配列の属性として読み出せる。

```
>>>x=arange(10)
>>>print x
0 1 2 3 4 5 6 7 8 9
>>x.shape
(10,)
```

配列の型はこの属性に新しいshapeを代入することで変更できる。

```
>>>x.shape=(2,5)
>>>print x
0 1 2 3 4
5 6 7 8 9
```

このプロセスは”スマート”で新しいshapeの長さが古い長さと一致しない場合にはそれを検知する。

[1]訳注：それぞれのデータが固有のメンバとメソッドを持つという意味ではPythonにおいてはすべてのデータはオブジェクトであるとも言えるが、Listなどの構造をclass定義でInheriteすることは出来ないなどの制限があるため、これらのデータ構造をPython言語の中でいうオブジェクト(class定義とそれによって作り出されたインスタンス）とはいえない

3.4 配列を含む式

配列を含む式は配列の各要素に式を作用させた結果を各要素とする配列を生成する。この”各要素毎”の考えは配列のshapeが異なるさまざまな場合に拡張される。また、各要素毎に作用するよく使われる関数が用意されている。

例えば、二つの実数配列 a および bが同じshapeを持っていれば、

```
x=(a+b)/(a-b)+sin(a)**2+cos(b)**s
```

はa および bと同じshapeをもつ実数配列をxに割り当てる。この実数配列の各要素は対応するa および bの各要素を用いて上記の数式を評価したものになる。

一般には、a および bが同じshape を持っていない場合には、上記の表式は例外を引き起こす。この規則には、重要な例外がある。：もし配列のどの軸でも長さ 1 のものがあった場合には、この配列はその他の軸についてshapeが一致するどのような配列と組み合わせて配列を含む式の中で使える。例えば、shapeが(3,1,2)である配列は、shapeが(3,5,2)や(3,100,2)などの配列と組み合わせて、配列を含む式を構成できる[2]。この展開は、第一の配列は長さ 1 を持つ軸の方向に、他の配列のshapeと一致するまで繰り返される。（もちろん、配列の要素が実際にメモリ上でコピーされるわけではない。結果は同じだが。）

この拡張プロセスはさらにもう一歩拡張される。二つの配列が同じ配列の次元を持たない場合には、低次側の配列は高次側の配列の次元まで”upgrade”される。この”upgrade”は、長さ 1 の軸を低次配列のshapeの前方に付け加えることで行われる。例えば、shapeが (3,2) の配列が shape (5,3,2)の配列と組み合わされる場合には、低次の配列はshape(1,3,2)を持つと見なされて、shape(5,3,2)になるように繰り返される。

この拡張と繰り返しはよく使われる表式にコンパクトな表式を許す。例えば、スカラーの定数を任意の形をした配列に加えることが出来る。この表式は配列のすべての要素にスカラー定数を足すことと同じ効果を持つ。同様に、配列全体をスカラー定数倍することも可能である。次の例では、shape (3,)の配列 yを shape (4,3)の配列 xに加える。結果はyを xの各行に加えたものになる。

```
>>> x
10 11 12
13 14 15
16 17 18
19 20 21
>>> y
0 1 2
>>> x+y
10 12 14
13 15 17
16 18 20
19 21 23
```

この強力な組合わせ方法を駆使するには、しばしば長さ 1 の軸を配列のshapeの前方以外の位置に付け加えることが有効である。これを実現する方法については、次の”配列の添字”の項で説明する。

3.5 配列の添字

配列クラスは配列の一部を取り出す取捨選択された機能を有する。一つの要素を指し示す配列の添え字はスカラーの結果を返す。その他の添字は指定された部分配列へのポインタを生成する。

```
>>> from Numeric import *
>> y=arange(12)
>>> y.shape=(4,3)
>>> print y
[[ 0  1  2]
 [ 3  4  5]
```

[2] shapeが(3,5,2)の配列と(3,100,2)の配列を一つの配列を含む式の中で使うことはできない。

```
 [ 6  7  8]
 [ 9 10 11]]
>>> print y[0,0]
0
>>> print [1,0]
[1, 0]
>>> print y[1,0]
3
>>> print y[0,1]
1
>>> print y[0][1]
1
>>> print y[1]
[3 4 5]
>>> 3 4 5
```

ある次元についてすべてあるいは一部の要素を選び出すためにスライス演算子：を使う事もできる。Python では、i:j は iまたはそれ以上かつ j 以下のすべてのインデックスを意味する。i:j:kの形式を増分を指定するために使える。

```
>>> print y[1:3]
[[3 4 5]
 [6 7 8]]
>>> print y[:,1]
[ 1  4  7 10]
>>> y[:,1]=[5,6,7,8]
>>> print y
[[ 0  5  2]
 [ 3  6  5]
 [ 6  7  8]
 [ 9  8 11]]
>>>
```

3.6 配列添え字の特殊な使い方

いま二つのベクトルaとbの外積を求めようとしているとしよう。つまり、shapeが $(\mathrm{len}(a), \mathrm{len}(b))$で、$c[i,j] = a[i] * b[j]$となる配列$c$を求めようとしているとする。この配列$c$は以下のあまりエレガントとは言えない計算で求めることはできる。

```
>>> a=array([1,2,3])
>>> b=array([10,20])
>>> c=zeros((len(a),len(b)))
>>> for j in range(len(b)):
...   for i in range(len(a)):
...     c[i,j]=a[i]*b[j]
...
>>> print c
[[10 20]
 [20 40]
 [30 60]]
```

これはあまりにエレガントさを欠く。またこの方法は明らか長いベクトルを操作する際には遅い方法である。Python インタプリタはかなり速いけれども、コンパイルされた操作に比べれば大分遅い。したがって、配列操作の文法の範囲でこれを実現する方法を探す。鍵は繰返しコピーの利用である。

```
>>> a.shape=(3,1)
```

```
>>> c=a*b
>>> print c
[[10 20]
 [20 40]
 [30 60]]>
```

$a*b$と書くときに前節で説明した自動データコピーの機能を使っていることに注意して欲しい。この例に示された方法をもっとやり易くするためにやるべきことはaのshapeの変更を行うもっと良い方法を見つけることである。実際我々の求めていることは、aに長さ１の新しい軸を付け加えることである。長さ１の新しい軸を付け加えることは非常にありふれたshapeの変更であるから、Pythonは短い表記法をこの操作に用意してある。配列の添字式に、単に特別なインデックスNewAxisを新しい軸を付け加えたい場所に置くだけでよい。従って、

```
a[:,NewAxis]
```

が我々の望むものである。そして外積は簡単に、

```
>>> c= a[:,NewAxis]*b
```

となる。

別のよくある操作は、最初の軸のすべての要素について、最後の軸の第０要素を取り出すというものである。もちろんこの操作は今まで説明してきた機能で実現できる。もし配列が(2,2,3)のshape を持っているとすると、答えは：

```
a[:,:,0]
```

である。しかしこの操作を任意のshapeを持つ配列について行うためにはどうすれば良いであろうか。一つ一つの軸についてスライス演算子を使わなければならない。そのためには、軸の数を知る必要がある。解決法は、特別のインデックス...である。この特別のインデックス...は添え字式が配列のすべての軸を満たすのに必要な個数のslice演算子を意味している。（この定義より、この特殊インデックスは添字式の中で一度しか現れない。）上記の例は従って、

```
a[\ldots,0]
```

とかける。

3.7 arrayオブジェクトのMethod

夫々の配列は適応可能な一連の関数（メンバ関数）を持っている。これらの関数には、

- $x.equal(y)$あは1あるいは0を要素として持つxと同じshape を持つ配列を値として返す。この配列の要素がxとyの各要素が等しいかそうでないかに従って1あるいは0の値をとる。$equal$の他に、同系統の$notEqual, greater, greaterEqual$, 似通った論理演算を行うための$andLogical, orLogical, notlogical$関数がある。(Pythonの通常のscalar比較演算子は配列データ型には使えないことに注意せよ。)
- $x.matrixMultiply(y)$　は数学的なxとy行列の積を値として返す。
- 与えられた配列から新たな配列を作り出すための一連の関数：transpose, complex conjugate, copies, concatenations 等々が用意されている。
- $x.choose(list)$, $x.take(list)$, and $x.repeat(list)$が配列の一部を選び出す手段として用意されている。

3.8 縮約

数値機能拡張は、その他のPythonと同じように、関数を第一級のオブジェクトとして扱っている。ある種の配列に対する演算に最適化された幾つかの関数が用意されている。それらの関数の多く(sin , cos , $sqrt$など)は各要素毎に作用する単項関数である。関数オブジェクトの中には二つの引数を要求する関数(binalry function)もある。そのなかでも、add と $multiply$ は特によく使われる。$multiply(x,y)$は　$x*y$　と同じ結果をもつ。add や $multiply$がオブジェクトとして取り扱えることは重要なコンセプトである。これらの（関数）オブジェクトはまた夫々のメソッド関数をもつ。これらのメソッド関数のうちもっとも重要なメソッド関数は$reduce$である。

2項関数による縮約は配列のある軸（defaultでは最初の軸）に沿ってこの2項関数を繰り返し適用していく。この繰り返しは結果がスカラになるまで繰り返される。従って、reduceの適用の結果、次元の１低い配列が生み出される。

```
>>> z
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])
>>> add.reduce(z)
array([ 5.,  7.,  9.])
>>> add.reduce(z,1)
array([  6.,  15.])
```

省略可能な第二の引数は、取り除こうとする軸のインデックスを指定するのである。上の例ではzはshape (2,3)をもつ。既定値 (0) で縮約を行うと、結果の配列のshapeは(3,)となる。次元のインデックス 1について縮約を行うと結果のshapeは(2,)である。

この性質は簡単な"dot"積の定義を可能にする。

```
>>> def dot(a,b):
...  return add.reduce(a*b)
...
>>> w
array([ 1.,  2.,  3.])
>>> v
array([-3.,  2.,  2.])
>>> dot(w,v)
7.0
```

4 Pythonのオブジェクト指向機能

5 A Programming Strategy

これまで見てきたことは極めて素晴らしいことである。完全な配列処理の機能がユーザが作成した機能拡張としてPythonに付け加えられたのである。数値機能拡張は高性能の配列言語と洗練された道具の組み合わせである。このためにPython Parserに加えられた変更がある。虚数定数、指数演算のための**演算子、そして複数のSlicingの引数等をサポートするためである[3]。これらの拡張は文法的な見栄えをよくするために必要なだけである。例えばこの拡張によって、a[(1,2)]と書く替りにa[1,2]と書くことができるようになる。これらの変更のうち主なものはすでにPython自体のto-doリストに入っている。またこの変更によって既存のプログラムが動作しなくなることもない。

Pythonのもっとも強力な点は成長しつつあるユーザのコミュニティとユーザの協力の精神の下にユーザが開発したモジュールからなるユーザが開発したライブラリである。次のURLを見て欲しい：

```
http://www.python.org/python/Contributed.html
```

netCDFや、PDB 自己記述ファイル、PLPLOT グラフィック・パッケージ、Yorick グラフィック・パッケージ、FFTパッケージなどのインターフェイスが開発中である。あなたの興味ある点についてnewsgroup comp.lang.pythonで尋ねてみることをお奨めする。

Pythonの数値機能拡張は極めて成功した戦略の一つのケースに過ぎない。その戦略とは"Pythonをscripting 言語として使い、コンパイラ言語で書かれたオブジェクトと機能を追加する"というものである。Pythonの単純性とアプリケーションに特定のオブジェクトを容易に作り出す事ができる簡便性はPythonを科学者にたいして魅力的なツールとしている。このことは、Pythonの移植性、ライセンス フリーであること、十分な性能、そして容易な拡張性と相まって、Pythonを未来の鍵となるツールとしている。

付録には、LANLから配付されているNumerical Python Packageに含まれるNumerical extensionについてのドキュメント(doc.html)の翻訳を収めておく。

[3]訳注：これらの機能は1.5ではすでにPythonパーサの基本機能として拡張されている

A Nemerical Python document

NumPyはプログラム言語Pythonに高機能の多次元数値配列の機能を付け加えるための一連のPythonモジュールの集合体である。

A.1 Constructors - 配列オブジェクトを生成する。

以下に述べる関数が新しい配列を作る際に使われる。

A.1.1 array(sequence, typecode=None, copy=1)

引数として与えられた、**sequence**から配列(array)オブジェクトを作る。配列の形(shape)は**sequence**の入れ子構造から決定される。配列オブジェクトの要素の型は、**typecode**で明示的に示されていない場合には、**sequence**の要素が収まるような最小の型となる。

typecodeはタイプの名前とビット幅の組み合わせで指定される。以下の組み合わせは大体どのプラットフォームでも利用可能である。：Int8, Int16, Int32, Int64 (on 64-bit processors), Float32, Float64, Complex32, Complex64。Int, Float, Complexはその計算機で使用可能な最大のビット幅をもつ型として使える。いろいろな型が交じり合った配列をつくるためには、**typecode**にPyObjectを指定する。

```
>>> a=array((1,2.3,4L),PyObject)
>>> a
array([1 , 2.3 , 4L ],'O')
```

copyフラグが1にセットされていると、値として返される配列はソースとなる**sequence**のデータをコピーして作られる。**copy**フラグが0にセットされると、**sequence**が配列オブジェクトで、かつ適切なtypecodeをもっている場合には、データのコピーは作られずに、データソースとなる**sequence**と同じメモリ領域を指す配列オブジェクトが返される。

A.1.2 zeros(shape, typecode=Int)

zeros() は形が **shape** で**typecode** に指定されたtypecodeをもつすべての要素が 0 の配列を返す。zeros() とones()は、**shape**として、tuple あるいは 一次元の配列形を示すものとして一つの整数を指定できる。

A.1.3 ones(shape, typecode=Int)

ones() は形が **shape** で**typecode** に指定されたtypecodeをもつすべての要素が 1 の配列を返す。

A.1.4 arrayrange(start=0, stop, step=1, typecode=None)

array(range(**start**, **stop**, **step**), **typecode**) と等価であるが、より効率よく実行される。

A.1.5 fromstring(string, typecode)

fromstring(string, typecode) は**string**に与えられたバイナリデータから**typecode**に指定された要素の型をもつ配列オブジェクトを生成する[4]。この関数は専らバイナリデータをファイルから読み書きするために使われる。また、PILのようにPythonの文字列をバイナリデータを保存するデータ型として使うpython モジュールとバイナリデータを交換する際にもこの関数は役に立つ。

A.2 Convenience Functions

A.2.1 identity(n)

n x **n** の単位行列を返す。

[4] この関数は**string** をバイナリデータとして扱う。ASCIIで書かれた数字の文字列を読み込む様には設計されていない。

A.2.2 indices(shape)

与えられた形の配列の指標（index）を返す。一番外側の次元がどの指標かを示す。その他の次元は与えられた形の配列の指標となる。

A.2.3 fromfunction(function, shape)

与えられた形 **shape**の配列のインデックスに関数 **function**を適用した結果を要素とする配列を返す。関数は、数値演算とufuncだけをふくむものでなければならない。この制限は将来緩和されるであろう。

A.2.4 arange()

arrayrange()と等価である。

A.2.5 asarray(a, typecode=None)

この関数は必ずしも配列オブジェクトを生成しない。引数に与えられた配列が正しい型をもっているなら、この関数はその引数に与えられた配列を値として返す。もし、型が異なっていれば引数はarray(a,typecode)にわたされ、その結果が値となる。この関数は、ある型を要素とする配列が必要なときに、不要なコピーを引き起こすことなくその型の配列が使われること担保する方法を与える。

A.3 Structural Operations on Arrays (or any python sequence)

A.3.1 take(a, indices, axis=0)

配列 **a**の軸**axis** にそった指標**indices**の要素を選び出す。**indices**は整数の配列であって、負数は（Pythonの通常の指標と同じように）軸の最後から数えた指標と取り扱われる。

A.3.2 reshape(a, shape)

配列**a**の形を新しい形**shape** に変更する。**shape** は配列の次元のTupleである。形**shape** の内一つの要素が -1 であってもよい。対応する軸の次元は新しい形の要素の総数が元の形でのそれと同じになるように決められる。配列の大きさはもとの配列のそれと完全に一致せていなければならない。大きさと形をへんこうするにはresize()関数を使う必要がある。

A.3.3 resize(a, shape)

reshapeと同じように働く。ただ一点、新しい配列の大きさが元の配列と同じでなくとも良い点がreshapeとは異なっている。この操作はreshapeに比べ効率の悪い方法である。注意しなければ奇妙な結果を引き起こす。

A.3.4 transpose(a, axes=None)

配列 **a**の軸を入れ換える。axes引数として負数が許される。負の次元数は配列の軸の最後から値を割り振る。

A.3.5 repeat(a, repeats, axis=0)

与えられた配列 **a** のそれぞれの要素は **repeats** の対応する要素に示された数だけ繰り返される。結果の指定された軸にそっての長さは**repeats**の要素の和になる。その他の軸に添っての結果の形は変わらない。

A.3.6 choose(a, ($\mathbf{b}_0$, ..., $\mathbf{b}n$))

aは0 から n までの間の整数を要素とする配列である。結果の配列は**a**とおなじ形をもつ。結果の配列の各要素は対応する**a**の要素に指定された様に $\mathbf{b}_0$,...,$\mathbf{b}_n$ から選び出された値である。

A.3.7 concatenate((a_0,...,a_n), axis=0)

引数に与えられた配列はaxisに指定された軸にそって結合される。すべての配列は結合する軸以外の軸について同じ形を持たなければならない。新たに作り出された軸にそって配列を結合するためには、array((a_0,...,a_n))の形が使える。この場合にも全ての配列は同じ形をもっている必要がある。

A.3.8 diagonal(a, k=0)

配列 aの {k} 番目の軸に添った対角要素を返す。(k は主要な対角要素からのオフセットである。) この関数は２次元配列を折り扱えるように設計されている。より高次の配列に対しては、それぞれの２次元副配列の対角成分を値として返す。

A.4 より簡便な利用のために導入された関数

A.4.1 ravel(a)

reshape(a, (-1,))と等価。aを一次元配列に変換した配列を返す。

A.4.2 nonzero(a)

a中の成分が０でない要素の指標（index）を返す。一次元配列についてのみ意味をもつ。

A.4.3 where(condition, true, false)

結果はconditionの形の配列である。その値は条件(condition)が満たされているか(true)あるいは満たされていないか(false)にしたがって、true あるいはfalse の要素の値となる。

A.4.4 compress(condition, a, axis=0)

aの要素のうち対応するconditionの要素の値が非零のものを値として返す。Condition は配列a.の与えられた軸 axis と同じ大きさでなければならない。結果のaxisに沿っての形はconditionの非零の要素の数と同じになる。その他の軸については結果の配列はaxisと同じ形を持つ。

A.4.5 trace(a, k=0)

配列aのk番目の軸にそっての対角成分の和を値とする。

A.5 配列の整列と検索

A.5.1 sort(a, axis=-1)

配列aの軸axisに沿って整列した結果を返す。

A.5.2 argsort(a, axis=-1)

sort(a)の結果を与える指標(index)を要素とする配列を返す。すなわち、take(a, argsort(a)) == sort(a). となる。

A.5.3 searchsorted(a, values)

a は整列された一次元配列である。要素がvaluesと一致する位置の指標を返す。

A.5.4 argmax(a, axis=-1), argmin(a, axis=-1)

与えられた軸axisに沿っての 配列aの最大値および最小値を要素を返す。結果の配列はaより１低い次元を持つ。

A.6 配列オブジェクトのメソッド

配列オブジェクトは極めて限られた数のメソッドしか持たない。配列に対する操作はほとんどが関数として実現されている。これは、これらの関数が任意のPythonのsequence型データにたいしても働くようにするためである。たとえば、transpose([[1,2],[3,4]])は全く問題なく働くけれど、[[1,2],[3,4]].transpose()は多分働かないであろう。このやり方はまた、CやPythonで定義されたPythonの基本システムの関数と数値機能拡張の中で定義された関数の間の一様性を保つことにも役立っている。この節で紹介されるメソッドは配列オブジェクトの実現方法の詳細に大きく依存している。

A.6.1 a.astype(typecode)

astypeメソッドは配列 a の全ての要素をtypecode型に型変換した配列を返す。警告：この型変換はC言語の型変換で実現され、範囲のチェックなどは行われない。必要であれば、独自の範囲チェック等をつけくわえることが出来る。

A.6.2 a.itemsize()

配列オブジェクトaの要素のバイト単位の大きさ

A.6.3 a.byteswapped()

配列オブジェクトと同じデータを含む新たな配列を返す。ただし、この配列の要素のバイト順序は入れ替わっている。このメソッドはバイナリデータをことなるバイト順序をもつ計算機間で交換する場合に有効である。

A.6.4 a.typecode()

配列a.の要素の型を示す文字を返す。

A.6.5 a.iscontiguous()

配列aが連続領域上に確保されていれば真を火エス。array. array(a) あるいは copy.copy(a)は必要な場合には連続領域に確保されることが保証されている。

A.6.6 a.tostring()

配列a をPythonの文字列としてふくむ配列を返す。もし、aの確保する領域が連続でなければ、この関数の返すデータはあたかも連続であったようなデータを返す。

A.6.7 a.tolist()

配列a を階層的なPythonのリスト型データとして返す。明示的にPythonのリスト型データを必要とする関数に配列を実引数として渡したいときに有効である。

A.7 配列オブジェクトの属性

配列オブジェクトは四つの属性を持っている。その内二つだけが一般にも役に立つだろう。

A.7.1 a.shape

それぞれの次元に沿った配列の長さを要素とするtupleを返す。配列aの次元はlen(a.shape) で求められる。この属性(shape)は代入可能である。代入によって配列のshapeを変更出来る。この属性に代入が可能なのは、新しいshapeが元のshapeと同じ数だけの要素を含む場合のみである。この降るまいが求めるものではないなら、resize()関数を調べてみよ。

¥subsubsectiona.flat この属性は連続領域にとられた配列についてのみ意味がある。配列aの要素を一次元配列にしたものをあたいとする。

¥subsubsectiona.real この関数は配列の実数部を返す。単純なはいれつではこれはもとの全く同じである。 This returns the real part of an array. For non-complex arrays, this is exactly equal to the array itself.

¥subsubsectionsa.imaginary この関数は配列の虚数部を返す。実配列にたいしては例外を発生する。

A.8 代数的および論理的演算子

これらの演算子はumath(およびfast_umath)にふくまれるオブジェクトである。これらのオブジェクトはcallableなオブジェクトであって、通常の関数と同じように使われる。それに加えて、これらのオブジェクトは、縮約, 積算、外積などのメソッドを持っている。

A.8.1 add, subtract, multiply, divide, remainder, power

A.8.2 arccos, arccosh, arcsin, arcsinh, arctan, arctanh, cos, cosh, exp, log, log10, sin, sinh, sqrt, tan, tanh

A.8.3 maximum, minimum, conjugate

A.8.4 equal, not_equal, greater, greater_equal, less, less_equal

A.8.5 logical_and, logical_or, logical_xor, logical_not, boolean_and, boolean_or, boolean_xor, boolean_not

これらのオブジェクトの機能はその名前から容易に類推できる（と希望している)。

A.9 Ufunc メソッド

前節の全てのオブジェクトはこの節で述べるメソッドを持っている。

A.9.1 ufunc.reduce(a, axis=0)

まさに、reduce(ufunc, a, [ufunc's identity element]) とおなじ様に機能する。一点だけことなるのは、axisによって縮約を行う軸を選べることである。配列a の axis 軸に沿っての長さが零の場合には、ufuncに対する単位要素が返されることに注意すべきである。

A.9.2 ufunc.accumulate(a, axis=0)

上のreduceと同じように働く。中間の結果が保存される所だけが異なっている。

A.9.3 ufunc.outer(a, b)

a と bの外積をとる。結果の配列の形(shape)はa.shape+b.shapeとなる。（ここでの+は和ではなくTupleの結合を意味している）

A.9.4 ufunc.reduceat(a, indices, axis=0)

このメソッドはwiredなものである。ほとんどのひとは無視してよいだろう。このメソッドは配列aを与えられたindicesのそれぞれにまで縮約しaのaxisに沿った長さがindicesと同じになるようにする。

A.10 数値演算関数

A.10.1 dot(a,b)

a と bのドット積を返す。これは２次元配列の行列としての積と等価である。（転置は不要である）

A.10.2 convolve(a,b,mode=0)

a と bの畳み込み積を返す。この関数は通常の方法を用いて実現されている。FFTを使っているわけではない。この関数は大きな配列を小さな配列と畳み込むことを目的としている。（原著者は波形データをガウス型関数を使って平滑化するためにこの関数を使っている。）modeは 0,1,2の内の一つを選べる。

1. 0 - 結果の配列の長さは len(a)-len(b)+1
2. 1 - 結果の配列の長さは len(a)
3. 2 - t結果の配列の長さはlen(a)+len(b)-1

modeが１または２の場合には配列の両側に不足するデータを補うために零のデータが付け加えられる。

A.10.3 clip(a, max, min)

max と minのあいだに配列aの要素の値を制限する。

A.11 以下の関数が簡便性のために導入された。

A.11.1 sum, cumsum, product, cumproduct, alltrue, sometrue

B 数値機能拡張への標準的な拡張

B.1 離散Fourier 変換 - FFT.py

これらの関数はFFTPACK routineをf2cで変換し変更を加えられたプログラムを基にしている。

B.1.1 fft(a, n=None, axis=-1)

配列 a のn 点離散Fourier変換を返す。nが指定されない場合には、配列 a の長さが仮定される。nが2のべき乗の場合にもっとも効率がよい。n がaの長さより大きいばあいには、a はその差をうめるために零データが補充される。n がaの長さより小さい場合には、aはサイズを小さくするために折り返される。この関数は異る大きさのFFTの作業領域をnのそれぞれの値ごとに、キャッシュに保存する。そのために、この異る n で何度もこの関数を利用すると、理屈の上では memory の問題に出くわすことになる

B.1.2 inverse_fft(a, n=None, axis=-1)

配列 a.のn 点離散逆Fourier変換を返す。nが指定されない場合には、配列aの長さが仮定される。nが2のべき乗の場合にもっとも効率がよい。n がaの長さより大きいばあいには、a はその差をうめるために零データが補充される。n がaの長さより小さい場合には、aはサイズを小さくするために折り返される。この関数は異る大きさのFFTの作業領域をnのそれぞれの値ごとに、キャッシュに保存する。そのために、この異るnで何度もこの関数を利用すると、理屈の上ではmemoryの問題に出くわすことになる

B.1.3 real_fft(a, n=None, axis=-1)

実数の配列 a.のn 点離散実Fourier変換を返す。nが指定されない場合には、配列aの長さが仮定される。nが2のべき乗の場合にもっとも効率がよい。返される配列は実配列の対称複素変換の半分である。

B.1.4 fft2d(a, s=None, axes=(-2,-1))

B.2 2次元のFourier 変換 a.

B.2.1 real_fft2d(a, s=None, axes=(-2,-1))

実数の２次元配列 a.の２次元離散実Fourier変換

B.3 雑音配列 - RandomArray.py

この節の関数はranlibに基づいている。

B.3.1 seed(x=0,y=0)

二つの整数から、ランダム数生成関数の種を初期かする。双方が零の場合には、種はシステムの時計からとられる。

B.3.2 get_seed()

現在ランダム数生成関数が用いてい種を与える二つの整数を返す。

B.3.3 random(shape=[])

０から１の間に一様に分布するランダムな数字の列からなる配列を値として返す。値の配列の形(shape)は引数に指定された物になる。shapeが空の場合にはスカラを返す。

B.3.4 uniform(minimum, maximum, shape=[])

minimum と maximum の間に一様に分布するランダムな数字列を返す。

B.3.5 randint(minimum=0, maximum, shape=[])

minimum と maximum の間に分布するランダムな整数列を返す。range()関数と同じく、maximum は含まない。

B.3.6 permutation(n):

arrayrange(n)の値をランダムに再配列した配列を返す。

B.4 線形代数 - LinearAlgebra.py

この節の関数はLAPACKの関数を基にしている。

B.4.1 solve_linear_equations(a,b)

B.4.2 inverse(a)

行列aの逆行列を返す。matrixmultiply(a, inverse(a)) == identity(len(a))

B.4.3 eigenvalues(a)

行列aの固有値を返す。

B.4.4 eigenvectors(a)

eigenvectors(a) は u,v を値として返す。ここで u は固有値を成分とする配列で、vは固有ベクトルからなる行列である。固有値u[i]に対応する固有ベクトルがv[i]になっている。a,u,v は関係式、$matrixmultiply(a, v[i]) = u[i] * v[i]$を満足する。

B.4.5 singular_value_decomposition(a)

B.4.6 generalized_inverse(a)

B.4.7 determinant(a)

正方行列aの行列式を返す。

B.4.8 linear_least_squares(a,b,rcond=1.e-10)

linear_least_squares(a,b) は x,resids,rank,s. を値として返す、ここで、

- **x は 2-norm(**$| b - Ax |$**) の最小値をあたえる。**
- **resids は残差の自乗和**
- **rank はAのランク**
- **s is an rank of the singual values of A in desending order**

bが行列である場合にはxもまた行列となる。行列Aのランクが Aの列の数より小さいかあるいは行の数より大きい場合には、residualは空配列となる。それ以外の場合には、$resids = sum((b - dot(A, x) * *2)$ **である。s[0]*rcondより小さい特異値(singular value)は零として取り扱われる。**