

NumPy & SciPy 入門



Contents

NumPy & Scipy の機能紹介

→ ndarrayクラスの操作を中心に説明

NumPy & SciPy を使ったプログラミング

→ 線形回帰、主成分分析, K-Means

→ Pythonの特徴であるOOPも取り入れる



Contents

NumPy & Scipy の機能紹介

→ ndarrayクラスの操作を中心に説明

NumPy & SciPy を使ったプログラミング

→ 線形回帰、主成分分析, K-Means

→ Pythonの特徴であるOOPも取り入れる

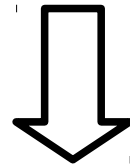


What is NumPy/SciPy?

Pythonで数値計算

数値計算 \div 繰り返し作業、ループ

Python : ループが遅い、その他諸々は書きやすい。
C, Fortran : ループが速い、冗長な表現が多い。



ループのある作業をC, Fortranで書き、
それをpythonから呼べたら便利!



What is NumPy/SciPy?

NumPy

- Cで実装されたndarray class
- Universal Function
- 乱数の生成
- f2pyによるFortranとの連携

SciPy

- NumPyを元に様々な数値計算の関数の集合
- BLAS/LAPACKを使った行列計算
- FFT, クラスタリング、関数最適化 などなど



What is NumPy & SciPy?

numpy.ndarray class

数値計算用のN次元配列

Homogeneous N-dimensional Array

各要素の型がすべて同じ → CやFortranの配列

強力なインデクシング表現

A[0:10, 3] etc

Universal functionによる直感的な数式の表現

$y[:] = 3.0 * \text{np.sin}(x[:]) + x[:]^{**}2 + e[0,:]$ etc



ndarray class

ndarrayの生成

- ✓np.array : listなどをndarrayに変換
- ✓np.empty : 任意のサイズを確保,Cのmalloc相当
- ✓np.zeros : 0で初期化,Cのcalloc相当
- ✓np.ones : 1で初期化
- ✓np.identity : 単位行列 etc



ndarray class

ndarrayの属性

- ✓dtype : 各要素の型, float64 etc
- ✓ndim : 配列の次元
- ✓shape : 各次元の要素数のtuple
- ✓size : 1次元にした時の長さ
- ✓data : 実際のデータがbuffer objectとして保持されている。



ndarray class

要素の参照、代入

- ✓ 個々の参照 : `x[2]`, `x[-1]`, `A[1, 2]`, `B[3, 2, 5]`
- ✓ スライスによる一括参照 : `x[from:to:step]`
e.g. `x[2:6]`, `x[::-1]`, `A[1, :]`, `A[:, 2, :6]`
- ✓ 1つの要素へ代入 : `x[4] = 10.0`
- ✓ スライスを使った代入 : `x[1:5] = 3.0`, `A[0, 1:7] = x[0:6]`



ndarray class

Universal Functions

配列のすべての要素に対して同じ操作を施す。(*elementwise operation*)

✓ $b[:] = 2.0 * a[:] + 1.0$

✓ $B[:, :] = \text{np.exp}(A[:, :])$

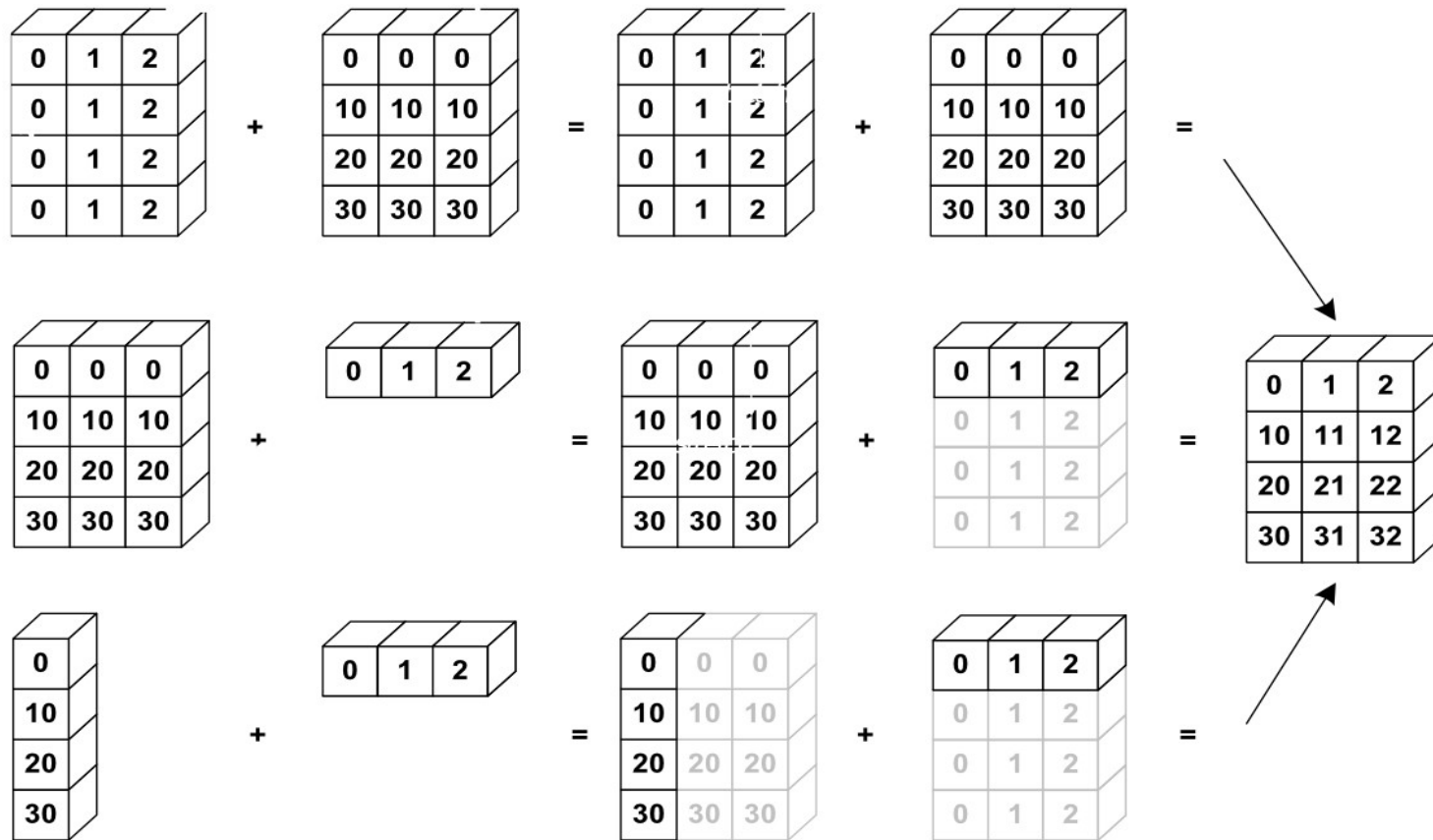
✓ $C[:, 1:10] = B[:, 3:12] + \text{np.sin}(B[:, :9])$

✓ $D = C[:, 1:10] * C[:, 1:10] \leftarrow$ 行列積ではないので注意!



ndarray class

Broadcast : 配列形状の自動調整



EuroScipy 2010, Python Scientific Notes より引用



ndarray class

Broadcastをうまく使うためのコツ

np.newaxisを使って配列の次元を揃える

✓A[10,3] に x[3] を足す → $A[:,:] + x[\text{np.newaxis},:]$

```
for i in xrange(3):  
    A[:,i] += x[i]
```

相当の計算になる。(もちろんこれより速い。)

✓B[100,4,10] に y[4] を掛ける
→ $B[:, :, :] * y[\text{np.newaxis}, :, \text{np.newaxis}]$

```
for i in xrange(4):  
    B[:,i,:] *= y[i]
```



ndarray class

Fancy Indexing

- ✓ `ixs = [0,2,2,4], x[ixs] → array(x[0],x[2],x[2],x[4])`
- ✓ `np.where(x[:]>1) → 1より大きいxのインデックスの配列`
- ✓ `np.where(x[:]>0,x,10) → xの0以上の要素をすべて10にする`
- ✓ `mask = A[:,:] > 0`
 - maskはbool型でAと同じshapeをもつndarray
 - `A[mask]`は0以上のAの要素が入った1D-ndarray
- ✓ `mask = np.sum(A[:,:]**2,1) < 0, inner_sphere = A[mask,:]`
 - 球の内側のデータ



ndarray class

io関係

- ✓np.savetxt : ファイルにndarrayを書き込む
np.savetxt("data.log",data)
- ✓np.loadtxt : ファイルからndarrayに読み込む
data = np.loadtxt("data.log")
- ✓PythonのbuiltinのPickleも使える
大きいデータをバイナリで保存する際に便利



NumPyと乱数

numpy.random

- ✓Mersenne Twisterによる乱数生成
- ✓rand : $[0,1)$ 上の一様乱数
r = rand(100,3) → 一度に生成して配列に格納
- ✓randn : 標準正規乱数、randと同様に配列に格納もできる
- ✓dirichlet, normal, t, beta, poisson 他多数
shapeを引数にとって配列に格納もできる。
rand,randnとは書式が多少違うので注意!



NumPyの統計関数

- ✓np.max, np.min : 最大最小
- ✓np.argmax, np.argmin : 最大最小の位置
- ✓np.mean, np.var, np.std : 平均、分散、標準偏差
- ✓np.sort, np.argsort : 配列のソートとそのインデックス

これらは任意次元のndarrayとaxisを引数にとって任意の軸に対して実行可

- ✓np.cov : 共分散, データは2D-ndarrayのみ
shapeは(ndim,nobs)の順なので注意!



NumPy / SciPyで行列計算

np.dot (BLASのラッパー)

`np.dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])`

例 x,y 1D-array, A,B 2D-array

- `dot(x,y)` → スカラ, BLASの***dot相当
- `dot(A,x)` → ベクター, BLASの***mv相当
- `dot(A,B)` → 行列, BLASの***mm相当(dgemmとか)

その他、outer, tensordotとかもある



NumPy / SciPyで行列計算

scipy.linalg (LAPACKのラッパー)

- det : 行列式
- inv, pinv : 逆行列、擬似逆行列
- solve, lstsq : 連立方程式の解、最小自乗解
 $Ax = b \rightarrow x = \text{solve}(A, b)$
- eig, eigh, svd : 固有値分解、特異値分解 (**ev, **svd)
 $\text{eig_val}, \text{eig_vec} = \text{eigh}(A)$
- 行列の因子分解
lu, qr, cholesky etc



NumPy / SciPyで行列計算

scipy.linalg (LAPACKのラッパー)

- det : 行列式
- inv, pinv : 逆行列、擬似逆行列
- solve, lstsq : 連立方程式の解、最小自乗解
 $Ax = b \rightarrow x = \text{solve}(A, b)$
- eig, eigh, svd : 固有値分解、特異値分解 (**ev, **svd)
 $\text{eig_val}, \text{eig_vec} = \text{eigh}(A)$
- 行列の因子分解
lu, qr, cholesky etc



特殊関数

scipy.special

- ほとんどがUniversal Functionなのでndarrayを食わせると一気に計算してくれる。
- gamma関数群 (分布関係でよく使う)
gamma, gammaln, digamma
- ベッセル関数、楕円関数など諸々 (あまりつかったことない...)



その他諸々

- `scipy.spatial` : KD-treeとか距離計算とか
- `scipy.sparse` : 疎行列とその線形代数
- `scipy.cluster` : 階層/非階層クラスタリング
- `scipy.fftpack` : Fourier変換
- `scipy.interpolate` : スプライン補完とか
- `scipy.optimize` : 関数最適化、非線形フィッティングとか
- `scipy.constants` : 科学定数集



Matplotlib

とにかくにも `import matplotlib.pyplot as plt`

- `plot(x, y, [symb, label, linewidth])` : xとyをsymbでプロット
- `legend` : labelを表示
- `xlabel, xlim, xtics` : 軸の表示設定、yも同様
- `show` : 画面にプロットの結果を表示
- `savefig` : 画像ファイルにプロットを保存 (png, eps, svg, pdf)



便利なPythonのパッケージ

- scikits.learn : NumPy&SciPyを使った機械学習のライブラリ
- PyMC : MCMCによるベイズ推定
- Networkx : グラフアルゴリズム
- OpenOpt : 関数最適化 (Pythonインターフェイスあり)
- OpenCV : 画像処理 (Pythonインターフェイスあり)
- NLTK : 自然言語処理
- pyMPI, PyCUDA : MPI, CUDAへのインターフェース
- Biopython, PyMol, MODELLER : バイオインフォマティクス
- pygresql, mysqldb, pysqlite2 : SQLへのインターフェイス
- Universal Feed Parser : RSS, ATOMのパパーサー

PyPI <<http://pypi.python.org/pypi>> で欲しいパッケージを検索できる。



Contents

NumPy & Scipy の機能紹介

→ ndarrayクラスの操作を中心に説明

NumPy & SciPy を使ったプログラミング

→ 線形回帰、主成分分析, K-Means

→ Pythonの特徴であるOOPも取り入れる



NumPy & SciPy を用いたプログラミング

大原則

- 極力ループにPythonのforではなく、NumPy対応する関数を使う。(特にデータ数に対するループは厳禁!)
- 大抵の操作はあるのでよく探すこと!
- ホットスポットの検出にはPythonのプロファイラが便利。
- それでもやっぱり遅いよ! → f2py, Cythonなどを検討



NumPy & SciPy を用いたプログラミング

Principal Component Analysis

解き方 共分散行列の固有値分解 or データ行列の特異値分解

実装方 (共分散行列版)

1. np.covを使って共分散行列を計算
2. scipy.linalg.eighを使って固有値分解
3. np.dotを使って射影



NumPy & SciPy を用いたプログラミング

Least Square Fit

$$y_n = \sum_i x_{ni} c_i + \epsilon \quad \text{or} \quad \mathbf{Y} = \mathbf{X}\mathbf{c} + \epsilon$$

解き方 $\hat{\mathbf{c}} = \underline{\text{pinv}(\mathbf{X})} \mathbf{Y}$
pseudo-inverse

実装方

- a) `scipy.linalg.svd`をコール ← 一番低レベル
- b) `scipy.linalg.pinv`をコール
- c) `scipy.linalg.lstsq`をコール



NumPy & SciPy を用いたプログラミング

K-Means (理論)

$$\ln P(x_n | c, \theta_c) = \ln \mathcal{N}(x_n; \mu_c, \Sigma_c)$$

x_n : observable

c_n : hidden variable

θ_c : model parameters

•E-Step $c_n = \operatorname{argmax}_c \ln P(x_n | c, \theta_c)$

•M-Step $\theta_c = \operatorname{argmax}_c \sum_{c_n=c} \ln P(x_n | c, \theta_c)$

$$\mu_c = \frac{1}{N_c} \sum_{c_n=c} x_n$$

$$\Sigma_c = \frac{1}{N_c} \sum_{c_n=c} (x_n - \mu_c)(x_n - \mu_c)^T$$



NumPy & SciPy を用いたプログラミング

K-Means (実装)

•E- step

各サンプルに対してInPの計算 (二次形式、Mahalanobis距離)

→ `scipy.spatial.distance.cdist`

各サンプルに対してInP最大のクラス番号を選ぶ

→ `np.argmax`

•M-step

`c_n = c` のサンプルの選択 → ndarrayのfancy indexing機能

平均、共分散の計算 → `np.mean`, `np.cov`

