Initiation à la programmation en Python: Calcul Scientifique

Zhentao Li

École Normale Supérieure

16 mars 2016

Une parenthèse sur les algorithmes avant de commencer

Aujourd'hui nous allons voir des manipulations de valeurs matricielle pour essayer d'arriver à un résultat voulu. Les étapes utilisées pour arriver à ce but ne sont par évidents (sauf peut-être pour les gens qui ont utilisés un autre langage comme MATLAB basé sur le même principe).

Le style que nous verrons aujourd'hui est plutôt déclaratif et les résultats sont exprimées en terme d'opérations sur matrices.

Pour comprendre ce que cela veut dire, examinons le contexte dans lequel on se place.

Une parenthèse sur les algorithmes avant de commencer

Tout opération d'un ordinateur peut être vu comme

- des **interactions** avec le monde extérieur (clavier, souris, écran, imprimante, etc) et
- des **algorithmes** qui consiste purement en la transformation de données.

La plupart des exercise de ce cours demandent implicitement pour des algorithmes (en plus de demander pour un programme syntactiquement et sémantiquement correcte).

Une parenthèse sur les algorithmes avant de commencer

Tout opération d'un ordinateur peut être vu comme

- des **interactions** avec le monde extérieur (clavier, souris, écran, imprimante, etc) et
- des **algorithmes** qui consiste purement en la transformation de données.

La plupart des exercise de ce cours demandent implicitement pour des algorithmes (en plus de demander pour un programme syntactiquement et sémantiquement correcte).

Algorithmes

Un algorithme consiste en une **spécification** (définit prochainement) et un ensemble d'**étapes** qui satisfait cette spécification. La spécification d'un algorithme comprends une **entrée** et une **sortie**.

Un exemple

Entrée: Une liste.
Sortie: Le plus petit élément de la liste.
Étapes:

1. Initialiser un minimum temporaire initialisée au premier élément de la liste.
2. Traverser la liste et pour chaque élément, remplacer le minimum temporaire par celui-ci si l'élément est plus petit.
3. Retourner le minimum temporaire.

Un algorithme resemble beaucoup à la définition d'une fonction que nous avons vu en cours, mais celle-ci n'est pas nécéssairement liée à un langage particulier. Un algorithme concrétisé dans un langage de programmation est appelée une *implémentation* de celle-ci.

Algorithmes

Un algorithme consiste en une **spécification** (définit prochainement) et un ensemble d'**étapes** qui satisfait cette spécification. La spécification d'un algorithme comprends une **entrée** et une **sortie**.

Un exemple

Entrée: Une liste.
Sortie: Le plus petit élément de la liste.
Étapes:

1. Initialiser un minimum temporaire initialisée au premier élément de la liste.
2. Traverser la liste et pour chaque élément, remplacer le minimum temporaire par celui-ci si l'élément est plus petit.
3. Retourner le minimum temporaire.

Un algorithme resemble beaucoup à la définition d'une fonction que nous avons vu en cours, mais celle-ci n'est pas nécéssairement liée à un langage particulier. Un algorithme concrétisé dans un langage de programmation est appelée une *implémentation* de celle-ci.

Algorithmes concrets

Beaucoup d'algorithmes utilsés contrètement se ressemblent. Et pour cette raison, connaître l'entrée et la sortie voulue suffit pour en déduire les étapes (d'un algorithme ≪ canonique ≫).
C'est notamment une des explications plausibles de la compréhension de liste en python:

Rappel

```
carres = [i*i for i in range(10)]
```

Cette ligne peut être vue comme une déclaration de la spécification: on veut à partir d'une liste (entrée) créer une deuxième liste.
Ce style plutôt *déclaratif* qu'impératif.
Et voilà l'explication sur la remarque au départ.

Le style que nous verrons aujourd'hui est plutôt déclaratif et les résultats sont exprimées en terme d'opérations sur matrices.

Calcul scientifique: SciPy et Pylab

Syntaxe

```
from scipy import *
from pylab import *
```

Ou bien

Syntaxe

```
import scipy as sp
import pylab as pl
```

Documentation :

- http://docs.scipy.org/doc/
- http://matplotlib.sourceforge.net/

Vecteurs et matrices

Type de base : `array`

Vecteur

```
x = sp.array([1,2,3])
```

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

Matrice

```
M = sp.array([[1,2,3],[4,5,6]])
```

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Construction

Vecteur/Matrice de 1

```
x = sp.ones(5)
M = sp.ones((3,2))
```

$$x = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Matrice identité et matrice diagonale

```
M = sp.identity(3)
N = sp.diag([1,2,3])
```

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad N = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Construction

Vecteur de valeurs consécutives

```
x = sp.arange(10)
y = sp.linspace(0,1,11)
```

$$x = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

$$y = \begin{pmatrix} 0 & 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1 \end{pmatrix}$$

Opérations

Éléments par éléments

```
x+y
x-y
x*y
x/y
x**n
sp.exp(x)
sp.sqrt(x)
```

Produit scalaire

```
sp.dot(x,y)
sp.dot(M,x)
sp.dot(M,N)
```

Opérations

Transpositon

```
M.T
```

Maximum

```
M.max()
```

Somme

```
M.sum()
```

Nombre d'éléments

```
sp.size(x)
M.shape()
```

Vecteurs booléens

```
x = sp.array([1,2,3,4])
x > 2.5
```

```
[False, False, True, True]
```

```
A[B]
```

= tous les a_{ij} tels que b_{ij} est `True`

```
x[x > 2.5] += 1
```

ajoute 1 à chaque élément > 2.5

```
M = sp.rand(3,3)
sp.where(M > 0.5)
```

indices (ligne, colonne) des éléments > 0.5

```
i = sp.where(x > 2.5)[0]
```

indices des éléments > 2.5

```
>>> dir(scipy)
['ALLOW_THREADS', 'BUFSIZE', 'CLIP', 'ERR_CALL', 'ERR_DEFAULT', 'ERR_DEFAULT2', 'ERR_IGNORE', 'ERR_LOG', 'ERR_PRINT', 'ERR_RAISE',
 'ERR_WARN', 'FLOATING_POINT_SUPPORT', 'FPE_DIVIDEBYZERO', 'FPE_INVALID', 'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf',
'Infinity', 'MAXDIMS', 'MachAr', 'NAN', 'NINF', 'NZERO', 'NaN', 'NumpyTest', 'PINF', 'PZERO', 'PackageLoader', 'RAISE', 'RankWarning',
 'True_', 'UFUNC_BUFSIZE_DEFAULT', 'UFUNC_PYVALS_NAME', 'WRAP', '__all__', '__builtins__', '__config__', '__doc__', '__file__',
'__name__', '__numpy_version__', '__path__', '__version__', 'absolute', 'add', 'add_docstring', 'add_newdoc', 'add_newdocs', 'alen',
 'all', 'allclose', 'alltrue', 'alterdot', 'amax', 'amin', 'angle', 'any', 'append', 'apply_along_axis', 'apply_over_axes', 'arange',
 'arccos', 'arccosh', 'arcsin', 'arcsinh', 'arctan', 'arctan2', 'arctanh', 'argmax', 'argmin', 'argsort', 'argwhere', 'around', 'array
 'ascontiguousarray', 'asfarray', 'asfortranarray', 'asmatrix', 'asscalar', 'atleast_1d', 'atleast_2d', 'atleast_3d', 'average',
'bartlett', 'base_repr', 'binary_repr', 'bincount', 'bitwise_and', 'bitwise_not', 'bitwise_or', 'bitwise_xor', 'blackman', 'bmat',
'bool8', 'bool_', 'broadcast', 'byte', 'c_', 'can_cast', 'cast', 'cdouble', 'ceil', 'central_diff_weights', 'cfloat', 'char', 'charact
 'complex128', 'complex192', 'complex64', 'complex_', 'complexfloating', 'compress', 'concatenate', 'conj', 'conjugate', 'convolve',
 'copy', 'corrcoef', 'correlate', 'cos', 'cosh', 'cov', 'cross', 'csingle', 'ctypeslib', 'cumprod', 'cumproduct', 'cumsum', 'delete',
 'deprecate', 'derivative', 'diag', 'diagflat', 'diagonal', 'diff', 'digitize', 'disp', 'divide', 'dot', 'double', 'dsplit', 'dstack',
 'dtype', 'e', 'ediff1d', 'emath', 'empty', 'empty_like', 'equal', 'errstate', 'exp', 'expand_dims', 'expm1', 'extract', 'eye', 'fabs
 'factorial', 'factorial2', 'factorialk', 'fastCopyAndTranspose', 'fft', 'fftpack', 'finfo', 'fix', 'flatiter', 'flatnonzero', 'flexib
 'fromfile', 'fromfunction', 'fromiter', 'frompyfunc', 'fromstring', 'generic', 'get_array_wrap', 'get_include', 'get_numarray_include
 'get_numpy_include', 'get_printoptions', 'getbuffer', 'getbufsize', 'geterr', 'geterrcall', 'geterrobj', 'gradient', 'greater',
'greater_equal', 'hamming', 'hanning', 'histogram', 'histogram2d', 'histogramdd', 'hsplit', 'hstack', 'hypot', 'i0', 'identity', 'ifft
'imag', 'index_exp', 'indices', 'inexact', 'inf', 'info', 'infty', 'inner', 'insert', 'int0', 'int16', 'int32', 'int64', 'int8','int_
 'int_asbuffer', 'intc', 'integer', 'integrate', 'interpolate', 'intersect1d', 'intersect1d_nu', 'intp', 'invert', 'io', 'iscomplex',
'iscomplexobj', 'isfinite', 'isfortran', 'isinf', 'isnan', 'isneginf', 'isposinf', 'isreal', 'isrealobj', 'isscalar', 'issctype',
'issubclass_', 'issubdtype', 'issubsctype', 'iterable', 'ix_', 'kaiser', 'kron', 'ldexp', 'left_shift', 'lena', 'less', 'less_equal',
'lexsort', 'lib', 'linalg', 'linsolve', 'linspace', 'little_endian', 'load', 'loads', 'log', 'log10', 'log1p', 'log2', 'logical_and',
'logical_not', 'logical_or', 'logical_xor', 'logn', 'logspace', 'longdouble', 'longfloat', 'longlong', 'ma', 'mat', 'math', 'matrix',
'maxentropy', 'maximum', 'maximum_sctype', 'mean', 'median', 'memmap', 'meshgrid', 'mgrid', 'minimum', 'mintypecode', 'misc', 'mod',
'modf', 'msort', 'multiply', 'nan', 'nan_to_num', 'nanargmax', 'nanargmin', 'nanmax', 'nanmin', 'nansum', 'nbytes', 'ndarray',
'ndenumerate', 'ndim', 'ndimage', 'ndindex', 'negative', 'newaxis', 'newbuffer', 'nonzero', 'not_equal', 'number', 'obj2sctype',
'object0', 'object_', 'ogrid', 'oldnumeric', 'ones', 'ones_like', 'optimize', 'outer', 'pade', 'pi', 'piecewise', 'pkgload', 'place',
'poly', 'poly1d', 'polyadd', 'polyder', 'polydiv', 'polyfit', 'polyint', 'polymul', 'polysub', 'polyval', 'power', 'prod', 'product',
'ptp', 'put', 'putmask', 'r_', 'rand', 'randn', 'random', 'rank', 'ravel', 'real', 'real_if_close', 'rec', 'recarray', 'reciprocal',
'record', 'remainder', 'repeat', 'require', 'reshape', 'resize', 'restoredot', 'right_shift', 'rint', 'roll', 'rollaxis', 'roots',
'rot90', 'round_', 'row_stack', 's_', 'sctype2char', 'sctypeDict', 'sctypeNA', 'sctypes', 'searchsorted', 'select', 'set_numeric_ops',
 'set_printoptions', 'set_string_function', 'setbufsize', 'setdiff1d', 'seterr', 'seterrcall', 'seterrobj', 'setmember1d', 'setxor1d',
 'shape', 'short', 'show_config', 'show_numpy_config', 'sign', 'signal', 'signbit', 'signedinteger', 'sin', 'sinc', 'single', 'sinh',
 'size', 'sometrue', 'sort', 'sort_complex', 'source', 'sparse', 'special', 'split', 'sqrt', 'square', 'squeeze', 'stats', 'std', 'str
 'string0', 'string_', 'subtract', 'sum', 'swapaxes', 'take', 'tan', 'tanh', 'tensordot', 'test', 'tile', 'trace', 'transpose', 'trapz
'tri', 'tril', 'trim_zeros', 'triu', 'true_divide', 'typeDict', 'typeNA', 'typecodes', 'typename', 'ubyte', 'ufunc', 'uint', 'uint0',
 'uint16', 'uint32', 'uint64', 'uint8', 'uintc', 'uintp', 'ulonglong', 'unicode0', 'unicode_', 'union1d', 'unique', 'unique1d',
'unravel_index', 'unsignedinteger', 'unwrap', 'ushort', 'vander', 'var', 'vdot', 'vectorize', 'version', 'void', 'void0', 'vsplit',
 'vstack', 'where', 'who', 'zeros', 'zeros_like']
```

Tracé de courbes 1D

Le tracé de toutes les courbes "scientifiques" se fait à l'aide de

```
import pylab as pl
```

Pour tracer une sinusoïde :

```
x = pl.linspace(-5,5,101) # coordonnées de -5 à 5 avec 101 valeurs
y = pl.sin(x)
pl.plot(x,y) # Tracé de la courbe !
```

Pour tracer plusieurs courbes, on peut les mettre les unes à la suite des autres, par exemple :

```
pl.plot(x, y, "r-", x, cos(x), "g.")
```

Tracé de courbes 2D

Pour cela on utilise `imshow(z)` ou `pcolor(x,y,z)`.

```
z = pl.identity(10)
pl.imshow(z)   # Affiche les entrées de la matrice z en 2D
```

```
x = pl.linspace(-5,5,201)
y = pl.linspace(-7,7,201)[:, pl.newaxis]
   # newaxis indique que ce vecteur est selon la 2ème dimension
z = pl.sin(x**2 + y**2)
pl.imshow(z) # Affiche l'image en 2D

pl.imshow(z, extent=(x.min(), x.max(), y.min(), y.max()))
   # On précise les coordonnées des axes
```

Diagramme circulaire

Exemple (de Matplotlib)

```
import pylab as pl

pl.figure(1, figsize=(6,6))
ax = pl.axes([0.1, 0.1, 0.8, 0.8])

labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
fracs = [15,30,45, 10]
explode = (0, 0.05, 0, 0)
pl.pie(fracs, explode=explode, labels=labels, autopct='%1.1f%%',
shadow=True)
pl.title('Raining Hogs and Dogs', bbox={'facecolor':'0.8', 'pad':5})
pl.show()
```

Diagramme circulaire