Python. Лекция 5.
Численные алгоритмы.
Матричные вычисления.

Numeric Python[1] - это несколько модулей для вычислений с многомерными массивами, необходимых для многих численных приложений. Модуль `Numeric` вносит в Python возможности таких пакетов и систем как MatLab, Octave (аналог MatLab), APL, J, S+, IDL. Пользователи найдут Numeric достаточно простым и удобным. Стоит заметить, что некоторые синтаксические возможности Python (связанные с использованием срезов) были специально разработаны для Numeric.

Numeric Python имеет средства для:

- матричных вычислений `LinearAlgebra` ;
- быстрого преобразования Фурье `FFT` ;
- работы с недостающими экспериментальными данными `MA` ;
- статистического моделирования `RNG` ;
- эмуляции базовых функций программы MatLab.

Модуль Numeric

Модуль `Numeric` определяет полноценный тип-массив и содержит большое число функций для операций с массивами. **Массив** - это набор однородных элементов, доступных по индексам. Массивы модуля `Numeric` могут быть многомерными, то есть иметь более одной **размерности**.

Создание массива

Для создания массива можно использовать функцию `array()` с указанием содержимого массива (в виде вложенных списков) и типа. Функция `array()` делает копию, если ее аргумент - массив. Функция `asarray()` работает аналогично, но не создает нового массива, когда ее аргумент уже является массивом:

```
>>> from Numeric import *
>>> print array([[1, 2], [3, 4], [5, 6]])
[[1 2]
 [3 4]
 [5 6]]
>>> print array([[1, 2, 3], [4, 5, 6]], float)
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
>>> print array([78, 85, 77, 69, 82, 73, 67], 'c')
[N U M E R I C]
```

В качестве элементов массива можно использовать следующие типы: `Int8-Int32`, `UnsignedInt8-UnsignedInt32`, `Float8-Float64`, `Complex8-Complex64` и `PyObject`. Числа 8, 16, 32 и 64 показывают количество битов для хранения величины. Типы `Int`, `UnsignedInteger`, `Float` и `Complex` соответствуют наибольшим принятым на данной платформе значениям. В массиве можно также хранить ссылки на произвольные объекты.

[1] Сайт, посвященный Numeric Python: http://www.scipy.org/

Количество размерностей и длина массива по каждой оси называются формой массива (shape). Доступ к форме массива реализуется через атрибут `shape`:

```
>>> from Numeric import *
>>> a = array(range(15), int)
>>> print a.shape
(15,)
>>> print a
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
>>> a.shape = (3, 5)
>>> print a.shape
(3, 5)
>>> print a
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

Методы массивов

Придать нужную форму массиву можно функцией `Numeric.reshape()`. Эта функция сразу создает объект-массив нужной формы из последовательности.

```
>>> import Numeric
>>> print Numeric.reshape("абракадабр", (5, -1))
[[а б]
 [р а]
 [к а]
 [д а]
 [б р]]
```

В этом примере `-1` в указании формы говорит о том, что соответствующее значение можно вычислить. Общее количество элементов массива известно (10), поэтому длину вдоль одной из размерностей задавать не обязательно.

Через атрибут `flat` можно получить одномерное представление массива:

```
>>> a = array([[1, 2], [3, 4]])
>>> b = a.flat
>>> b
array([1, 2, 3, 4])
>>> b[0] = 9
>>> b
array([9, 2, 3, 4])
>>> a
array([[9, 2],
       [3, 4]])
```

Следует заметить, что это новый вид того же массива, поэтому присваивание значений его элементам приводит к изменениям в исходном массиве.

Функция `Numeric.resize()` похожа на `Numeric.reshape`, но может подстраивать число элементов:

```
>>> print Numeric.resize("NUMERIC", (3, 2))
[[N U]
 [M E]
 [R I]]
>>> print Numeric.resize("NUMERIC", (3, 4))
[[N U M E]
 [R I C N]
 [U M E R]]
```

Функция `Numeric.zeros()` порождает массив из одних нулей, а `Numeric.ones()` - из одних единиц. Единичную матрицу можно получить с помощью функции `Numeric.identity(n)`:

```
>>> print Numeric.zeros((2,3))
[[0 0 0]
 [0 0 0]]
>>> print Numeric.ones((2,3))
[[1 1 1]
 [1 1 1]]
>>> print Numeric.identity(4)
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]]
```

Для копирования массивов можно использовать метод `copy()`:

```
>>> import Numeric
>>> a = Numeric.arrayrange(9)
>>> a.shape = (3, 3)
>>> print a
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> a1 = a.copy()
>>> a1[0, 1] = -1   # операция над копией
>>> print a1
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Массив можно превратить обратно в список с помощью метода `tolist()`:

```
>>> a.tolist()
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

Срезы

Объекты-массивы `Numeric` используют расширенный синтаксис выделения среза. Следующие примеры иллюстрируют различные варианты записи срезов. Функция `Numeric.arrayrange()` является аналогом `range()` для массивов.

```
>>> import Numeric
>>> a = Numeric.arrayrange(24) + 1
>>> a.shape = (4, 6)
>>> print a             # исходный массив
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
>>> print a[1,2]        # элемент 1,2
9
>>> print a[1,:]        # строка 1
[ 7  8  9 10 11 12]
>>> print a[1]          # тоже строка 1
[ 7  8  9 10 11 12]
>>> print a[:,1]        # столбец 1
[ 2  8 14 20]
>>> print a[-2,:]       # предпоследняя строка
[13 14 15 16 17 18]
>>> print a[0:2,1:3]    # окно 2x2
[[2 3]
```

```
 [8 9]]
>>> print a[1,::3]      # каждый третий элемент строки 1
[ 7 10]
>>> print a[:,::-1]     # элементы строк в обратном порядке
[[ 6  5  4  3  2  1]
 [12 11 10  9  8  7]
 [18 17 16 15 14 13]
 [24 23 22 21 20 19]]
```

Срез не копирует массив (как это имеет место со списками), а дает доступ к некоторой части массива. Далее в примере меняется на 0 каждый третий элемент строки 1:

```
>>> a[1,::3] = Numeric.array([0,0])
>>> print a
[[ 1  2  3  4  5  6]
 [ 0  8  9  0 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

В следующих примерах находит применение достаточно редкая синтаксическая конструкция: срез с многоточием (`Ellipsis`). Многоточие ставится для указания произвольного числа пропущенных размерностей (`:,:,...,:`):

```
>>> import Numeric
>>> a = Numeric.arrayrange(24) + 1
>>> a.shape = (2,2,2,3)
>>> print a
[[[[ 1  2  3]
   [ 4  5  6]]
  [[ 7  8  9]
   [10 11 12]]]
 [[[13 14 15]
   [16 17 18]]
  [[19 20 21]
   [22 23 24]]]]
>>> print a[0,...]          # 0-й блок
[[[ 1  2  3]
  [ 4  5  6]]
 [[ 7  8  9]
  [10 11 12]]]
>>> print a[0,:,:,0]        # срез по первой и последней размерностям
[[ 1  4]
 [ 7 10]]
>>> print a[0,...,0]        # то же, но с использованием многоточия
[[ 1  4]
 [ 7 10]]
```

Универсальные функции

Модуль Numeric определяет набор функций для применения к элементам массива. Функции применимы не только к массивам, но и к последовательностям (к сожалению, итераторы пока не поддерживаются). В результате получаются массивы.

Функция	Описание
`add(x, y), subtract(x, y)`	Сложение и вычитание
`multiply(x, y), divide(x, y)`	Умножение и деление

`remainder(x, y), fmod(x, y)`	Получение остатка от деления (для целых чисел и чисел с плавающей запятой)
`power(x, y)`	Возведение в степень
`sqrt(x)`	Извлечение корня квадратного
`negative(x), absolute(x), fabs(x)`	Смена знака и абсолютное значение
`ceil(x), floor(x)`	Наименьшее (наибольшее) целое, большее (меньшее) или равное аргументу
`hypot(x, y)`	Длина гипотенузы (даны длины двух катетов)
`sin(x), cos(x), tan(x)`	Тригонометрические функции
`arcsin(x), arccos(x), arctan(x)`	Обратные тригонометрические функции
`arctan2(x, y)`	Арктангенс от частного аргумента
`sinh(x), cosh(x), tanh(x)`	Гиперболические функции
`arcsinh(x), arccosh(x), arctanh(x)`	Обратные гиперболические функции
`exp(x)`	Экспонента (e^x)
`log(x), log10(x)`	Натуральный и десятичный логарифмы
`maximum(x, y), minimum(x, y)`	Максимум и минимум
`conjugate(x)`	Сопряжение (для комплексных чисел)
`equal(x, y), not_equal(x, y)`	Равно, не равно
`greater(x, y), greater_equal(x, y)`	Больше, больше или равно
`less(x, y), less_equal(x, y)`	Меньше, меньше или равно
`logical_and(x, y), logical_or(x, y)`	Логические И, ИЛИ
`logical_xor(x, y)`	Логическое исключающее ИЛИ
`logical_not(x)`	Логические НЕ

`bitwise_and(x, y), bitwise_or(x, y)`	Побитовые И, ИЛИ
`bitwise_xor(x, y)`	Побитовое исключающее ИЛИ
`invert(x)`	Побитовая инверсия
`left_shift(x, n), right_shift(x, n)`	Побитовые сдвиги влево и вправо на n битов

Перечисленные функции являются объектами типа `ufunc` и применяются к массивам поэлементно. Эти функции имеют специальные методы:

`accumulate()`	Аккумулирование результата.
`outer()`	Внешнее "произведение".
`reduce()`	Сокращение.
`reduceat()`	Сокращение в заданных точках.

Пример с функцией `add()` позволяет понять смысл универсальной функции и ее методов:

```
>>> from Numeric import add
>>> add([[1, 2], [3, 4]],  [[1, 0], [0, 1]])
array([[2, 2],
       [3, 5]])
>>> add([[1, 2], [3, 4]],  [1, 0])
array([[2, 2],
       [4, 4]])
>>> add([[1, 2], [3, 4]],  1)
array([[2, 3],
       [4, 5]])
>>> add.reduce([1, 2, 3, 4])                    # т.е. 1+2+3+4
10
>>> add.reduce([[1, 2], [3, 4]], 0)             # т.е. [1+3 2+4]
array([4, 6])
>>> add.reduce([[1, 2], [3, 4]], 1)             # т.е. [1+2 3+4]
array([3, 7])
>>> add.accumulate([1, 2, 3, 4])                    # т.е. [1 1+2 1+2+3
1+2+3+4]
array([ 1,  3,  6, 10])
>>> add.reduceat(range(10), [0, 3, 6])              # т.е. [0+1+2 3+4+5
6+7+8+9]
array([ 3, 12, 30])
>>> add.outer([1,2], [3,4])                       # т.е. [[1+3 1+4] [2+3
2+4]]
array([[4, 5],
       [5, 6]])
```

Методы `accumulate()`, `reduce()` и `reduceat()` принимают необязательный аргумент - номер размерности, используемой для соответствующего действия. По умолчанию применяется нулевая размерность.

Универсальные функции, помимо одного или двух необходимых параметров, позволяют задавать и еще один аргумент, для приема результата функции. Тип третьего аргумента должен строго соответствовать типу результата. Например, функция `sqrt()` даже от целых чисел имеет тип `Float`.

```
>>> from Numeric import array, sqrt, Float
>>> a = array([0, 1, 2])
>>> r = array([0, 0, 0], Float)
>>> sqrt(a, r)
array([ 0.        ,  1.        ,  1.41421356])
>>> print r
[ 0.          1.          1.41421356]
```

Предупреждение:

Не следует использовать в качестве приемника результата массив, который фигурирует в предыдущих аргументах функции, так как при этом результат может быть испорчен. Следующий пример показывает именно такой вариант:

```
>>> import Numeric
>>> m = Numeric.array([0, 0, 0, 1, 0, 0, 0, 0])
>>> add(m[:-1], m[1:], m[1:])
array([0, 0, 1, 1, 1, 1, 1])
```

В таких неоднозначных случаях необходимо использовать промежуточный массив.

Функции модуля Numeric

Следующие функции модуля `Numeric` являются краткой записью некоторых наиболее употребительных сочетаний функций и методов:

Функция	Аналог функции
`sum(a, axis)`	`add.reduce(a, axis)`
`cumsum(a, axis)`	`add.accumulate(a, axis)`
`product(a, axis)`	`multiply.reduce(a, axis)`
`cumproduct(a, axis)`	`multiply.accumulate(a, axis)`
`alltrue(a, axis)`	`logical_and.reduce(a, axis)`
`sometrue(a, axis)`	`logical_or.reduce(a, axis)`

Функции для работы с массивами

Функций достаточно много, поэтому подробно будут рассмотрены только две из них, а остальные сведены в таблицу.

Функция Numeric.take()

Функция `Numeric.take()` позволяет взять часть массива по заданным на определенном измерении индексам. По умолчанию номер измерения (третий аргумент) равен нулю.

```
>>> import Numeric
>>> a = Numeric.reshape(Numeric.arrayrange(25), (5, 5))
>>> print a
[[ 0  1  2  3  4]
```

```
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
>>> print Numeric.take(a, [1], 0)
[ [5 6 7 8 9]]
>>> print Numeric.take(a, [1], 1)
[[ 1]
 [ 6]
 [11]
 [16]
 [21]]
>>> print Numeric.take(a, [[1,2],[3,4]])
[[[ 5  6  7  8  9]
  [10 11 12 13 14]]
 [[15 16 17 18 19]
  [20 21 22 23 24]]]
```

В отличие от среза, функция `Numeric.take()` сохраняет размерность массива, если конечно, структура заданных индексов одномерна. Результат `Numeric.take(a, [[1,2],[3,4]])` показывает, что взятые по индексам части помещаются в массив со структурой самих индексов, как если бы вместо `1` было написано `[5 6 7 8 9]`, а вместо `2` - `[10 11 12 13 14]` и т.д.

Функции Numeric.diagonal() и Numeric.trace()

Функция `Numeric.diagonal()` возвращает диагональ матрицы. Она имеет следующие аргументы:

`a`	Исходный массив.
`offset`	Смещение вправо от "главной" диагонали (по умолчанию 0).
`axis1`	Первое из измерений, на которых берется диагональ (по умолчанию 0).
`axis2`	Второе измерение, образующее вместе с первым плоскость, на которой и берется диагональ. По умолчанию `axis2=1`.

Функция `Numeric.trace()` (для вычисления `следа матрицы`) имеет те же аргументы, но суммирует элементы на диагонали. В примере ниже рассмотрены обе эти функции:

```
>>> import Numeric
>>> a = Numeric.reshape(Numeric.arrayrange(16), (4, 4))
>>> print a
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
>>> for i in range(-3, 4):
...   print "Sum", Numeric.diagonal(a, i), "=", Numeric.trace(a, i)
...
Sum [12] = 12
Sum [ 8 13] = 21
Sum [ 4  9 14] = 27
Sum [ 0  5 10 15] = 30
Sum [ 1  6 11] = 18
Sum [2 7] = 9
Sum [3] = 3
```

Функция Numeric.choose()

Эта функция использует один массив с целыми числами от `0` до `n` для выбора значения из одного из заданных массивов:

```
>>> a = Numeric.identity(4)
>>> b0 = Numeric.reshape(Numeric.arrayrange(16), (4, 4))
>>> b1 = -Numeric.reshape(Numeric.arrayrange(16), (4, 4))
>>> print Numeric.choose(a, (b0, b1))
[[  0   1   2   3]
 [  4  -5   6   7]
 [  8   9 -10  11]
 [ 12  13  14 -15]]
```

Свод функций модуля Numeric

Следующая таблица приводит описания функций модуля Numeric.

Функция и ее аргументы	Назначение функции
`allclose(a, b[, eps[, A]])`	Сравнение `a` и `b` с заданными относительными `eps` и абсолютными `A` погрешностями. По умолчанию `eps` равен `1.0e-1`, а `A = 1.0e-8`.
`alltrue(a[, axis])`	Логическое И по всей оси `axis` массива `a`
`argmax(a[, axis])`	Индекс максимального значения в массиве по заданному измерению `axis`
`argmin(a[, axis])`	Индекс минимального значения в массиве по заданному измерению `axis`
`argsort(a[, axis])`	Индексы отсортированного массива, такие, что `take(a,argsort(a, axis),axis)` дает отсортированный массив `a`, как если бы было выполнено `sort(a, axis)`
`array(a[, type])`	Создание массива на основе последовательности `a` данного типа `type`
`arrayrange(start[, stop[, step[, type]]])`	Аналог `range()` для массивов
`asarray(a[, type[, savespace]])`	То же, что и `array()`, но не создает новый массив, если `a` уже является массивом.
`choose(a, (b0,...,bn))`	Создает массив на основе элементов, взятых по индексам из `a` (индексы от `0` до `n` включительно). Формы массивов `a`, `b1`, ..., `bn` должны совпадать
`clip(a, a_min, a_max)`	Обрубает значения массива `a` так, чтобы они находились

	между значениями из `a_min` и `a_max` поэлементно
`compress(cond, a[, axis])`	Возвращает массив только из тех элементов массива `a`, для которых условие `cond` истинно (не нуль)
`concatenate(a[, axis])`	Соединение двух массивов (конкатенация) по заданному измерению `axis` (по умолчанию - по нулевой)
`convolve(a, b[, mode])`	Свертка двух массивов. Аргумент `mode` может принимать значения `0`, `1` или `2`
`cross_correlate(a, b[, mode])`	Взаимная корреляция двух массивов. Параметр `mode` может принимать значения `0`, `1` или `2`
`cumproduct(a[, axis])`	Произведение по измерению `axis` массива `a` с промежуточными результатами
`cumsum(a[, axis])`	Суммирование с промежуточными результатами
`diagonal(a[, k[, axis1[, axis2]]])`	Взятие `k` -й диагонали массива `a` в плоскости измерений `axis1` и `axis2`
`dot(a, b)`	Внутреннее (матричное) произведение массивов. По определению: `innerproduct(a, swapaxes(b, -1, -2))`, т.е. с переставленными последними измерениями, как и должно быть при перемножении матриц
`dump(obj, file)`	Запись массива `a` (в двоичном виде) в открытый файловый объект `file`. Файл должен быть открыт в бинарном режиме. В файл можно записать несколько объектов подряд
`dumps(obj)`	Строка с двоичным представлением объекта `obj`
`fromfunction(f, dims)`	Строит массив, получая информацию от функции `f()`, в качестве аргументов которой выступают значения кортежа индексов. Фактически является сокращением для `f(*tuple(indices(dims)))`
`fromstring(s[, count[, type]])`	Создание массива на основе бинарных данных, хранящихся в строке
`identity(n)`	Возвращает двумерный массив формы `(n, n)`
`indices(dims[, type])`	Возвращает массив индексов заданной длины по каждому измерению с изменением поочередно по каждому изменению. Например, `indices([2, 2])[1]` дает двумерный массив `[[0, 1], [0, 1]]`.
`innerproduct(a, b)`	Внутреннее произведение двух массивов (по общему измерению). Для успешной операции `a.shape[-1]` должен

	быть равен `b.shape[-1]`. Форма результата будет `a.shape[:-1] + b.shape[:-1]`. Элементы пропадающего измерения попарно умножаются и получающиеся произведения суммируются
`load(file)`	Чтение массива из файла `file`. Файл должен быть открыт в бинарном режиме
`loads(s)`	Возвращает объект, соответствующий бинарному представлению, заданному в строке
`nonzero(a)`	Возвращает индексы ненулевых элементов одномерного массива
`ones(shape[, type])`	Массив из единиц заданной формы `shape` и обозначения типа `type`
`outerproduct(a, b)`	Внешнее произведение `a` и `b`
`product(a[, axis])`	Произведение по измерению `axis` массива `a`
`put(a, indices, b)`	Присваивание частям массива, `a[n] = b[n]` для всех индексов `indices`
`putmask(a, mask, b)`	Присваивание `a` элементов из `b`, для которых маска `mask` имеет значение истина
`ravel(a)`	Превращение массива в одномерный. Аналогично `reshape(a, (-1,))`
`repeat(a, n[, axis])`	Повторяет элементы массива `a` `n` раз по измерению `axis`
`reshape(a, shape)`	Возвращает массив нужной формы (нового массива не создает). Количество элементов в исходном и новом массивах должно совпадать
`resize(a, shape)`	Возвращает массив с произвольной новой формой `shape`. Размер исходного массива не важен
`searchsorted(a, i)`	Для каждого элемента из `i` найти место в массиве `a`. Массив `a` должен быть одномерным и отсортированным. Результат имеет форму массива `i`
`shape(a)`	Возвращает форму массива `a`
`sometrue(a[, axis])`	Логическое ИЛИ по всему измерению `axis` массива `a`
`sort(a[, axis])`	Сортировка элементов массива по заданному измерению
`sum(a[, axis])`	Суммирование по измерению `axis` массива `a`

`swapaxes(a, axis1, axis1)`	Смена измерений (частный случай транспонирования)
`take(a, indices[, axis])`	Выбор частей массива `a` на основе индексов `indices` по измерению `axis`
`trace(a[, k[, axis1[, axis2]]])`	Сумма элементов вдоль диагонали, то есть `add.reduce(diagonal(a, k, axis1, axis2))`
`transpose(a[, axes])`	Перестановка измерений в соответствии с `axes`, либо, если `axes` не заданы - расположение их в обратном порядке
`where(cond, a1, a2)`	Выбор элементов на основании условия `cond` из `a1` (если не нуль) и `a2` (при нуле) поэлементно. Равносилен `choose(not_equal(cond, 0), (y, x))`. Формы массивов-аргументов `a1` и `a2` должны совпадать
`zeros(shape[, type])`	Массив из нулей заданной формы `shape` и обозначения типа `type`

В этой таблице в качестве обозначения типа `type` можно указывать рассмотренные выше константы: `Int`, `Float` и т.п.

Модуль `Numeric` также определяет константы `e` (число e) и `pi` (число пи).

Модуль LinearAlgebra

Модуль `LinearAlgebra` содержит алгоритмы линейной алгебры, в частности нахождение определителя матрицы, решений системы линейных уравнений, обращение матрицы, нахождение собственных чисел и собственных векторов матрицы, разложение матрицы на множители: Холецкого, сингулярное, метод наименьших квадратов.

Функция `LinearAlgebra.determinant()` находит определитель матрицы:

```
>>> import Numeric, LinearAlgebra
>>> print LinearAlgebra.determinant(
...     Numeric.array([[1, -2],
...                    [1, 5]]))
7
```

Функция `LinearAlgebra.solve_linear_equations()` решает линейные уравнения вида `ax=b` по заданным аргументам `a` и `b`:

```
>>> import Numeric, LinearAlgebra
>>> a = Numeric.array([[1.0, 2.0],  [0.0, 1.0]])
>>> b = Numeric.array([1.2, 1.5])
>>> x = LinearAlgebra.solve_linear_equations(a, b)
>>> print "x =", x
x = [-1.8  1.5]
>>> print "Проверка:", Numeric.dot(a, x) - b
Проверка: [ 0.  0.]
```

Когда матрица `a` имеет нулевой определитель, система имеет не единственное решение и возбуждается исключение `LinearAlgebraError`:

```
>>> a = Numeric.array([[1.0, 2.0],  [0.5, 1.0]])
>>> x = LinearAlgebra.solve_linear_equations(a, b)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File                                  "/usr/local/lib/python2.3/site-
packages/Numeric/LinearAlgebra.py", line 98,
    in solve_linear_equations raise LinAlgError, 'Singular matrix'
LinearAlgebra.LinAlgError: Singular matrix
```

Функция `LinearAlgebra.inverse()` находит обратную матрицу. Однако не следует решать линейные уравнения с помощью `LinearAlgebra.inverse()` умножением на обратную матрицу, так как она определена через `LinearAlgebra.solve_linear_equations()`:

```
def inverse(a):
    return solve_linear_equations(a, Numeric.identity(a.shape[0]))
```

Функция `LinearAlgebra.eigenvalues()` находит собственные значения матрицы, а `LinearAlgebra.eigenvectors()` - пару: собственные значения, собственные вектора:

```
>>> from Numeric import array, dot
>>> from LinearAlgebra import eigenvalues, eigenvectors
>>> a = array([[-5, 2],  [2, -7]])
>>> lmd = eigenvalues(a)
>>> print "Собственные значения:", lmd
Собственные значения: [-3.76393202 -8.23606798]
>>> (lmd, v) = eigenvectors(a)
>>> print "Собственные вектора:"
Собственные вектора:
>>> print v
[[ 0.85065081  0.52573111]
 [-0.52573111  0.85065081]]
>>> print "Проверка:", dot(a, v[0]) - v[0] * lmd[0]
Проверка: [ -4.44089210e-16   2.22044605e-16]
```

Проверка показывает, что тождество выполняется с достаточно большой точностью (числа совсем маленькие, практически нули): собственные числа и векторы найдены верно.

Модуль RandomArray

В этом модуле собраны функции для генерации массивов случайных чисел различных распределений и свойств. Их можно применять для математического моделирования.

Функция `RandomArray.random()` создает массивы из псевдослучайных чисел, равномерно распределенных в интервале `(0, 1)`:

```
>>> import RandomArray
>>> print RandomArray.random(10)  # массив из 10 псевдослучайных чисел
[ 0.28374212  0.19260929  0.07045474  0.30547682  0.10842083  0.14049676
      0.01347435  0.37043894  0.47362471  0.37673479]
>>> print RandomArray.random([3,3])   # массив 3x3 из псевдослучайных
чисел
[[ 0.53493741  0.44636754  0.20466961]
 [ 0.8911635   0.03570878  0.00965272]
 [ 0.78490953  0.20674807  0.23657821]]
```

Функция `RandomArray.randint()` для получения массива равномерно распределенных чисел из заданного интервала и заданной формы:

```
>>> print RandomArray.randint(1, 10, [10])
[8 1 9 9 7 5 2 5 3 2]
>>> print RandomArray.randint(1, 10, [10])
```

```
[2 2 5 5 7 7 3 4 3 7]
```

Можно получать и случайные перестановки с помощью `RandomArray.permutation()`:

```
>>> print RandomArray.permutation(6)
[4 0 1 3 2 5]
>>> print RandomArray.permutation(6)
[1 2 0 3 5 4]
```

Доступны и другие распределения для получения массива нормально распределенных величин с заданным средним и стандартным отклонением:

```
>>> print RandomArray.normal(0, 1, 30)
[-1.0944078   1.24862444  0.20415567 -0.74283403  0.72461408 -0.57834256
 0.30957144  0.8682853   1.10942173 -0.39661118  1.33383882  1.54818618
 0.18814971  0.89728773 -0.86146659  0.0184834  -1.46222591 -0.78427434
 1.09295738 -1.09731364  1.34913492 -0.75001568 -0.11239344  2.73692131
 -0.19881676  -0.49245331     1.54091263  -1.81212211     0.46522358  -
0.08338884]
```

Следующая таблица приводит функции для других распределений:

Функция и ее аргументы	Описание
`F(dfn, dfd, shape=[])`	F-распределение
`beta(a, b, shape=[])`	Бета-распределение
`binomial(trials, p, shape=[])`	Биномиальное распределение
`chi_square(df, shape=[])`	Распределение хи-квадрат
`exponential(mean, shape=[])`	Экспоненциальное распределение
`gamma(a, r, shape=[])`	Гамма-распределение
`multivariate_normal(mean, cov, shape=[])`	Многомерное нормальное распределение
`negative_binomial(trials, p, shape=[])`	Негативное биномиальное
`noncentral_F(dfn, dfd, nconc, shape=[])`	Нецентральное F-распределение
`noncentral_chi_square(df, nconc, shape=[])`	Нецентральное хи-квадрат распределение
`normal(mean, std, shape=[])`	Нормальное распределение
`permutation(n)`	Случайная перестановка
`poisson(mean, shape=[])`	Пуассоновское распределение
`randint(min, max=None, shape=[])`	Случайное целое

`random(shape=[])`	Равномерное распределение на интервале (0, 1)
`random_integers(max, min=1, shape=[])`	Случайное целое
`standard_normal(shape=[])`	Стандартное нормальное распределение
`uniform(min, max, shape=[])`	Равномерное распределение

Заключение

В этой лекции рассматривался набор модулей для численных вычислений. Модуль `Numeric` определяет тип многомерный массив и множество функций для работы с массивами. Также были представлены модули для линейной алгебры и моделирования последовательностей случайных чисел различных распределений.