NumPy / SciPy and Array-Oriented Computing

Travis Oliphant, PhD
PyCodeConf

Oct 7, 2011
Python Fits Your Brain

**Thesis:** Software engineering today is more about neuroscience than computer science.
Even the Brain of Scientists
Thinking differently

Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

\[ y_0 = 0 \]
\[ y_1 = 1 \]
\[ y_i = y_{i-1} + y_{i-2} \]
How an engineer may see Fibonacci

• Unstable Infinite Impulse Response (IIR) linear filter

```python
from scipy.signal import lfilter, lfilteric
from numpy import zeros

b = [0]
a = [1, -1, -1]
yinit = [0, 1]
zi = lfilteric(b, a, yinit)

def fibonacci(N):
    x = zeros(N)
y, zf = lfilter(b, a, x, zi=zi)
    return y.astype(int)
```
Demo
Conway’s game of Life

- Dead cell with exactly 3 live neighbors will come to life
- A live cell with 2 or 3 neighbors will survive
- With too few or too many neighbors, the cell dies
Interesting Patterns emerge
APL: the first array-oriented language

• Appeared in 1964
• Originated by Ken Iverson
• Direct descendants (J, K, Matlab) are still used heavily and people pay a lot of money for them
• NumPy is a descendant
Conway’s game of Life using Arrays

APL

Φ' [0, ε N] ⊂ S ← [0, (3 = T) ∨ M ∧ 2 = T ⊃ +/(V Φ'' ⊂ M), (V Φ'' ⊂ M), (V, ΦV)Φ''(V, V ← 1 - 1) Φ'' ⊂ M]

NumPy

Initialization

size = 100
GRID = (rand(size, size) > 0.5).astype(uint8)
# The world is round
indx = r_[0:size]
up = roll(indx, -1)
down = roll(indx, 1)

Update Step

neighbors = GRID[up, :] + GRID[down, :] + GRID[:, up] + GRID[:, down] + Γ
              GRID[ix_(up, up)] + GRID[ix_(up, down)] + GRID[ix_(down, up)] + GRID[ix_(down, down)]
GRID = (neighbors == 3) | (GRID & (neighbors==2))
Derivative Calculations

\[
f'(x) = \frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}
\]

NumPy

# calculate the derivative dy/dx numerically.
# First, calculate the distance between adjacent pairs of
# x and y values.
dy = y[1:] - y[:-1]
dx = x[1:] - x[:-1]

# Now divide to get "rise" over "run" for each interval.
dy_dx = dy/dx

# Assuming central differences, these derivative values
# centered in-between our original sample points.
centers_x = (x[1:] + x[:-1]) / 2.0
A Little NumPy / SciPy History
# Python origins

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9.0</td>
<td>Feb. 1991</td>
</tr>
<tr>
<td>0.9.4</td>
<td>Dec. 1991</td>
</tr>
<tr>
<td>0.9.6</td>
<td>Apr. 1992</td>
</tr>
<tr>
<td>0.9.8</td>
<td>Jan. 1993</td>
</tr>
<tr>
<td>1.0.0</td>
<td>Jan. 1994</td>
</tr>
<tr>
<td>1.2</td>
<td>Apr. 1995</td>
</tr>
<tr>
<td>1.4</td>
<td>Oct. 1996</td>
</tr>
<tr>
<td>1.5.2</td>
<td>Apr. 1999</td>
</tr>
</tbody>
</table>

Early Contributors

Jim Fulton

Jim Hugunin
Numeric

Paul Dubois

Konrad Hinsen
Key Language Changes from Array-oriented Computing

- $a[0,1]$ instead of $a[(0,1)]$
- $a[::2]$ instead of just $a[:$
- Ellipsis object
- Complex numbers $1j$
Aside: We need a few more!!!


- Infix array operator (matrix multiplication is not domain specific)
- Use of slice notation inside function calls
- Array overloading of `and` and `or`
- DSL blocks?

- In general, the collaboration between Python core and Scientific developers needs to be tighter
\[ \rho_0 (2\pi f)^2 U_i (a, f) = [C_{ijkl} (a, f) U_k, l (a, f)]_{,j} \]
Finding Derivatives of 5-d data

\[ U_X (a, f) \]

\[ U_Y (a, f) \]

\[ U_Z (a, f) \]
Finding Derivatives of 5-d data

$$
\Xi = \nabla \times U
$$

$$
\Xi_x (a, f)
$$

$$
\Xi_y (a, f)
$$

$$
\Xi_z (a, f)
$$
Found Python and Numeric in 1997

I was a fairly proficient MATLAB user, but it was not memory efficient enough.

- Loved the expressive syntax of Python
- Loved the fact that slicing didn’t make copies
- Loved the existing multiple data-types
- Loved how much more flexible it was to extend than MATLAB was
- Loved that I could read the source code and extend it
First problem: Efficient Data Input

“It’s All About the Data”

TableIO
April 1998

Reference Counting Essay
http://www.python.org/doc/ess/
May 1998

Michael A. Miller

Guido van Rossum

NumPyIO
June 1998
Early pieces of SciPy

**fftw wrappers**
June 1998

**cephesmodule**
November 1998

**stats.py**
December 1998

Gary Strangman
1999 : Early SciPy emerges


In response on 15 Jan, 1999, I posted to matrix-sig a list of routines I felt needed to be present and began wrapping / writing in earnest. On **6 April 1999**, I announced I would be creating this uber-package which eventually became SciPy.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian quadrature</td>
<td>5 Jan 1999</td>
</tr>
<tr>
<td>cephes 1.0</td>
<td>30 Jan 1999</td>
</tr>
<tr>
<td>sigtools 0.40</td>
<td>23 Feb 1999</td>
</tr>
<tr>
<td>Numeric docs</td>
<td>March 1999</td>
</tr>
<tr>
<td>cephes 1.1</td>
<td>9 Mar 1999</td>
</tr>
<tr>
<td>multipack 0.3</td>
<td>13 Apr 1999</td>
</tr>
<tr>
<td>Helper routines</td>
<td>14 Apr 1999</td>
</tr>
<tr>
<td>multipack 0.6 (leastsq, ode, fsolve, quad)</td>
<td>29 Apr 1999</td>
</tr>
<tr>
<td>sparse plan described</td>
<td>30 May 1999</td>
</tr>
<tr>
<td>multipack 0.7</td>
<td>14 Jun 1999</td>
</tr>
<tr>
<td>SparsePy 0.1</td>
<td>5 Nov 1999</td>
</tr>
<tr>
<td>cephes 1.2 (vectorize)</td>
<td>29 Dec 1999</td>
</tr>
</tbody>
</table>

Helping with f2py

Plotting??

Gist
XPLOT
DISLIN
Gnuplot
Early contributors in 1999

Pearu Peterson

Hosting of first Multipack CVS repository (June 1999)
Amazing makefiles
Interface to FITPACK
Wrote f2py as he watched my brute-force approach (July 1999)

Janko Hauser

(IPP) Early IPython interactive environment (27 Apr 1999)
Matlab file reader (24 Apr 1999)

Robert Kern

Created windows binaries of multipack, cephesmodule, fftw, and signaltools (June 1999 while still in high school!)
Facets of SciPy

Conferences

Collection of Tools

Community
Now a Community effort

- Chuck Harris
- Pauli Virtanen
- Mark Wiebe
- David Cournapeau
- Stefan van der Walt
- Jarrod Millman
- Josef Perktold
- Anne Archibald
- Dag Sverre Seljebotn
- Robert Kern
- Matthew Brett
- Warren Weckesser
- Ralf Gommers
- Joe Harrington --- Documentation effort
- Andrew Straw --- www.scipy.org

many, many others --- forgive me!
SciPy is a **Large** community Project

A great way to get involved with Python
In 2009

North America
In 2009

Europe
SciPy Unique Visits – 2009 (~400K total)
(first half of)

<table>
<thead>
<tr>
<th>Universities</th>
<th>Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT</td>
<td>2965</td>
</tr>
<tr>
<td>Harvard</td>
<td>1553</td>
</tr>
<tr>
<td>Stanford</td>
<td>1534</td>
</tr>
<tr>
<td>Cambridge</td>
<td>1214</td>
</tr>
<tr>
<td>CalTech</td>
<td>1121</td>
</tr>
<tr>
<td>Cornell</td>
<td>1109</td>
</tr>
<tr>
<td>U Wash</td>
<td>1011</td>
</tr>
<tr>
<td>Berkeley</td>
<td>958</td>
</tr>
<tr>
<td>Oxford</td>
<td>811</td>
</tr>
<tr>
<td>Univ. BC</td>
<td>795</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Labs</th>
<th>Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>NASA</td>
<td>2378</td>
</tr>
<tr>
<td>LANL</td>
<td>508</td>
</tr>
<tr>
<td>LLNL</td>
<td>420</td>
</tr>
<tr>
<td>LBNL</td>
<td>368</td>
</tr>
<tr>
<td>ORNL</td>
<td>205</td>
</tr>
<tr>
<td>Brookhaven</td>
<td>172</td>
</tr>
<tr>
<td>Argonne</td>
<td>158</td>
</tr>
<tr>
<td>PNL</td>
<td>100</td>
</tr>
</tbody>
</table>

680K total visits in first half of 2010!!
### SciPy Unique Visits – 2009 (~400K total)

(first half of)

#### Tech Sampling

<table>
<thead>
<tr>
<th>Location</th>
<th>Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seagate</td>
<td>1018</td>
</tr>
<tr>
<td>Intel</td>
<td>390</td>
</tr>
<tr>
<td>Microsoft</td>
<td>246</td>
</tr>
<tr>
<td>HP</td>
<td>241</td>
</tr>
<tr>
<td>IBM</td>
<td>146</td>
</tr>
<tr>
<td>Google</td>
<td>85</td>
</tr>
<tr>
<td>Freescale</td>
<td>64</td>
</tr>
</tbody>
</table>

#### Industrial Sampling

<table>
<thead>
<tr>
<th>Location</th>
<th>Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing</td>
<td>286</td>
</tr>
<tr>
<td>Caterpillar</td>
<td>205</td>
</tr>
<tr>
<td>Airbus</td>
<td>158</td>
</tr>
<tr>
<td>P&amp;G</td>
<td>51</td>
</tr>
<tr>
<td>Rolls Royce</td>
<td>30</td>
</tr>
</tbody>
</table>

680K total visits in first half of 2010!!
### Financial Sampling

<table>
<thead>
<tr>
<th>Location</th>
<th>Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE Shaw</td>
<td>693</td>
</tr>
<tr>
<td>Citadel</td>
<td>373</td>
</tr>
<tr>
<td>JP Morgan</td>
<td>340</td>
</tr>
<tr>
<td>Softbank</td>
<td>315</td>
</tr>
<tr>
<td>Goldman Sachs</td>
<td>273</td>
</tr>
<tr>
<td>AQR Capital</td>
<td>171</td>
</tr>
<tr>
<td>Susquehanna</td>
<td>159</td>
</tr>
<tr>
<td>UBS</td>
<td>160</td>
</tr>
<tr>
<td>Tradelink</td>
<td>129</td>
</tr>
<tr>
<td>Deutsche Bank</td>
<td>78</td>
</tr>
<tr>
<td>Merrill Lynch</td>
<td>77</td>
</tr>
<tr>
<td>CitiGroup</td>
<td>62</td>
</tr>
<tr>
<td>Morgan Stanley</td>
<td>40</td>
</tr>
<tr>
<td>Renaissance</td>
<td>20</td>
</tr>
</tbody>
</table>

### Oil Sampling

<table>
<thead>
<tr>
<th>Location</th>
<th>Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shell</td>
<td>160</td>
</tr>
<tr>
<td>Stat Oil</td>
<td>155</td>
</tr>
<tr>
<td>Schlumberger</td>
<td>110</td>
</tr>
<tr>
<td>Exxon</td>
<td>76</td>
</tr>
<tr>
<td>ConocoPhillips</td>
<td>27</td>
</tr>
<tr>
<td>Aramco</td>
<td>27</td>
</tr>
</tbody>
</table>

680K total visits in first half of 2010!!
started January 2005

NumPy

Version 1.0 October 2006

Key contributions from:
Numarray
Numeric
Chuck Harris
Robert Kern
David Cooke
Pierre GM
NumPy: so what? (speed and expressiveness)

• Data: the array object
  – slicing
  – shapes and strides
  – data-type generality

• Fast Math:
  – vectorization
  – broadcasting
  – aggregations
NumPy Array

A NumPy array is an N-dimensional homogeneous collection of “items” of the same kind. The kind can be any arbitrary structure of bytes and is specified using the data-type.
Array Slicing

SLICING WORKS MUCH LIKE STANDARD PYTHON SLICING

>>> a[0,3:5]
array([[3, 4]])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2, 12, 22, 32, 42, 52])

STRIDES ARE ALSO POSSIBLE

>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])
Fancy Indexing in 2-D

```python
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([[ 1, 12, 23, 34, 45]])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                dtype=bool)

>>> a[mask,2]
array([2,22,52])
```

Unlike slicing, fancy indexing creates copies instead of a view into original array.
Statistics Array Methods

MEAN

```python
>>> a = array([[1, 2, 3],
             [4, 5, 6]], float)

# mean value of each column
>>> a.mean(axis=0)
array([ 2.5,  3.5,  4.5])

>>> mean(a, axis=0)
array([ 2.5,  3.5,  4.5])

>>> average(a, axis=0)
array([ 2.5,  3.5,  4.5])
```

# average can also calculate
# a weighted average
```python
>>> average(a, weights=[1,2],
             axis=0)
array([ 3.,  4.,  5.])
```
Dimensional reduction (sum)
Demo
## NumPy dtypes

<table>
<thead>
<tr>
<th>Basic Type</th>
<th>Available NumPy types</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>bool</td>
<td>Elements are 1 byte in size.</td>
</tr>
<tr>
<td>Integer</td>
<td>int8, int16, int32, int64, int128, int</td>
<td>int defaults to the size of int in C for the platform.</td>
</tr>
<tr>
<td>Unsigned Integer</td>
<td>uint8, uint16, uint32, uint64, uint128, uint</td>
<td>uint defaults to the size of unsigned int in C for the platform.</td>
</tr>
<tr>
<td>Float</td>
<td>float32, float64, float, longfloat,</td>
<td>float is always a double precision floating point value (64 bits). longfloat represents large precision floats. Its size is platform dependent.</td>
</tr>
<tr>
<td>Complex</td>
<td>complex64, complex128, complex, longcomplex</td>
<td>The real and complex elements of a complex64 are each represented by a single precision (32 bit) value for a total size of 64 bits.</td>
</tr>
<tr>
<td>Strings</td>
<td>str, unicode</td>
<td></td>
</tr>
<tr>
<td>Object</td>
<td>Object</td>
<td>Represent items in array as Python objects.</td>
</tr>
<tr>
<td>Records</td>
<td>Void</td>
<td>Used for arbitrary data structures.</td>
</tr>
</tbody>
</table>
“Structured” Arrays

Elements of an array can be any fixed-size data structure!

name char[10]
age int
weight double

Elements of an array can be any fixed-size data structure!

>>> from numpy import dtype, empty
# structured data format
>>> fmt = dtype([('name', 'S10'),
               ('age', int),
               ('weight', float)])
>>> a = empty((3,4), dtype=fmt)
>>> a.itemsize
22
>>> a['name'] = [['Brad', ... ,'Jill']]  
>>> a['age'] = [[33, ... , 54]]
>>> a['weight'] = [[135, ... , 145]]  
>>> print a

[['('Brad', 33, 135.0)
  ...
  ('Jill', 54, 145.0)]

<table>
<thead>
<tr>
<th>Brad</th>
<th>Jane</th>
<th>John</th>
<th>Fred</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>25</td>
<td>47</td>
<td>54</td>
</tr>
<tr>
<td>135.0</td>
<td>105.0</td>
<td>225.0</td>
<td>140.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Henry</th>
<th>George</th>
<th>Brian</th>
<th>Amy</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>61</td>
<td>32</td>
<td>27</td>
</tr>
<tr>
<td>154.0</td>
<td>202.0</td>
<td>137.0</td>
<td>187.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ron</th>
<th>Susan</th>
<th>Jennifer</th>
<th>Jill</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>33</td>
<td>18</td>
<td>54</td>
</tr>
<tr>
<td>188.0</td>
<td>135.0</td>
<td>88.0</td>
<td>145.0</td>
</tr>
</tbody>
</table>
## Nested Datatype

<table>
<thead>
<tr>
<th>Time</th>
<th>Size</th>
<th>Position</th>
<th>Gain</th>
<th>Samples (2048) ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Az</td>
<td>EI</td>
<td>Type</td>
</tr>
<tr>
<td>1172581077060</td>
<td>4108</td>
<td>0.715594</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077091</td>
<td>4108</td>
<td>0.706876</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077123</td>
<td>4108</td>
<td>0.698157</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077153</td>
<td>4108</td>
<td>0.689423</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077184</td>
<td>4108</td>
<td>0.680683</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077215</td>
<td>4108</td>
<td>0.671956</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077245</td>
<td>4108</td>
<td>0.663232</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077276</td>
<td>4108</td>
<td>0.654511</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077306</td>
<td>4108</td>
<td>0.645787</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077339</td>
<td>4108</td>
<td>0.637058</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077370</td>
<td>4108</td>
<td>0.628321</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077402</td>
<td>4108</td>
<td>0.619599</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077432</td>
<td>4108</td>
<td>0.61087</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077463</td>
<td>4108</td>
<td>0.590148</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077547</td>
<td>4108</td>
<td>0.584696</td>
<td>-0.148407</td>
<td>1</td>
</tr>
<tr>
<td>1172581077599</td>
<td>4108</td>
<td>0.575972</td>
<td>-0.148407</td>
<td>1</td>
</tr>
</tbody>
</table>
Nested Datatype

dt = dtype([('time', uint64),
            ('size', uint32),
            ('position', [('az', float32),
                          ('el', float32),
                          ('region_type', uint8),
                          ('region_ID', uint16)]),
            ('gain', np.uint8),
            ('samples', np.int16, 2048)])

data = np.fromfile(f, dtype=dt)
## Structured Arrays

Elements of array can be any fixed-size data structure!

<table>
<thead>
<tr>
<th>Name</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSFT_profit</td>
<td>10</td>
<td>6.20</td>
</tr>
<tr>
<td>GOOG_profit</td>
<td>12</td>
<td>-1.08</td>
</tr>
<tr>
<td>MSFT_profit</td>
<td>18</td>
<td>8.40</td>
</tr>
<tr>
<td>INTC_profit</td>
<td>25</td>
<td>-0.20</td>
</tr>
</tbody>
</table>

### Example

```python
>>> import numpy as np
>>> fmt = np.dtype([('name', 'S12'),
                  ('time', np.int64),
                  ('value', np.float32)])
>>> vals = [('MSFT_profit', 10, 6.20),
          ('GOOG_profit', 12, -1.08),
          ('MSFT_profit', 18, 8.40),
          ('INTC_profit', 25, -0.20),
          ('GOOG_profit', 1000325, 3.20),
          ('GOOG_profit', 1000350, 4.50),
          ('INTC_profit', 1000385, -1.05),
          ('MSFT_profit', 1000390, 5.60)]
>>> arr = np.array(vals, dtype=fmt)
# or
>>> arr = np.fromfile('db.dat', dtype=fmt)
# or
>>> arr = np.memmap('db.dat', dtype=fmt,
                  mode='c')
```
Memory Mapped Arrays

• Methods for Creating:
  – **memmap**: subclass of ndarray that manages the memory mapping details.
  – **frombuffer**: Create an array from a memory mapped buffer object.
  – **ndarray constructor**: Use the `buffer` keyword to pass in a memory mapped buffer.

• Limitations:
  – Files must be < 2GB on Python 2.4 and before.
  – Files must be < 2GB on 32-bit machines.
  – Python 2.5 on 64 bit machines is theoretically "limited" to 17.2 billion GB (17 Exabytes).
Memmap Timings (3D arrays)

<table>
<thead>
<tr>
<th>Operations (500x500x1000)</th>
<th>Linux</th>
<th>OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In Memory</td>
<td>Memory Mapped</td>
</tr>
<tr>
<td>read</td>
<td>2103 ms</td>
<td>11 ms</td>
</tr>
<tr>
<td>x slice</td>
<td>1.8 ms</td>
<td>4.8 ms</td>
</tr>
<tr>
<td>y slice</td>
<td>2.8 ms</td>
<td>4.6 ms</td>
</tr>
<tr>
<td>z slice</td>
<td>9.2 ms</td>
<td>13.8 ms</td>
</tr>
<tr>
<td>downsample 4x4</td>
<td>0.02 ms</td>
<td>125 ms</td>
</tr>
</tbody>
</table>

All times in milliseconds (ms).

Linux: Ubuntu 4.1, Dell Precision 690, Dual Quad Core Zeon X5355 2.6 GHz, 8 GB Memory
OS X: OS X 10.5, MacBook Pro Laptop, 2.6 GHz Core Duo, 4 GB Memory
## SciPy [Scientific Algorithms]

<table>
<thead>
<tr>
<th>linalg</th>
<th>stats</th>
<th>interpolate</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster</td>
<td>special</td>
<td>maxentropies</td>
</tr>
<tr>
<td>io</td>
<td>fftpack</td>
<td>odr</td>
</tr>
<tr>
<td>ndimage</td>
<td>sparse</td>
<td>integrate</td>
</tr>
<tr>
<td>signal</td>
<td>optimize</td>
<td>weave</td>
</tr>
</tbody>
</table>

## NumPy [Data Structure Core]

<table>
<thead>
<tr>
<th>fft</th>
<th>random</th>
<th>linalg</th>
</tr>
</thead>
</table>

### NDArray
- multi-dimensional array object

### UFunc
- fast array math operations
from scipy.optimize import curve_fit
from scipy.stats import norm

def function(x, a, b, f, phi):
    result = a * exp(-b * sin(f * x + phi))
    return result

actual_params = [3, 2, 1, pi/4]
x = linspace(0, 2*pi, 25)
exact = function(x, *actual_params)
noisy = exact + 0.3*norm.rvs(size=len(x))

initial_guess = [1, 1, 1, 1]
estimated_params, err_est = curve_fit(function, x, noisy, p0=initial_guess)
estimated_params
array([3.1705, 1.9501, 1.0206, 0.7034])

estimated_params
array([3.1705, 1.9501, 1.0206, 0.7034])

err_est is an estimate of the covariance matrix of the estimates
(i.e. how good of a fit is it)
2D RBF Interpolation

```python
>>> from scipy.interpolate import Rbf
>>> from numpy import hypot, mgrid
>>> from scipy.special import j0
>>> x, y = mgrid[-5:6,-5:6]
>>> z = j0(hypot(x,y))
>>> newfunc = Rbf(x, y, z)
>>> xx, yy = mgrid[-5:5:100j, -5:5:100j]
# xx and yy are both 2-d
# result is evaluated
# element-by-element
>>> zz = newfunc(xx, yy)
>>> from enthought.mayavi import mlab
>>> mlab.surf(x, y, z*5)
>>> mlab.figure()
>>> mlab.surf(xx, yy, zz*5)
>>> mlab.points3d(x,y,z*5,
 scale_factor=0.5)
```
Brownian Motion

Brownian motion (Wiener process):

\[ X(t+dt) = X(t) + N(0, \sigma^2 dt, t, t+dt) \]

where \( N(a,b,t_1,t_2) \) is normal with mean \( a \) and variance \( b \), and independent on disjoint time intervals.

```python
>>> from scipy.stats import norm
>>> x0 = 100.0
>>> dt = 0.5
>>> sigma = 1.5
>>> n = 100
>>> steps = norm.rvs(size=n, scale=sigma*sqrt(dt))
>>> steps[0] = x0  # Make i.c. work
>>> x = steps.cumsum()
>>> t = linspace(0, (n-1)*dt, n)
>>> plot(t, x)
```
# Edge detection using Sobel filter

```python
>>> from scipy.ndimage.filters import sobel
>>> imshow(lena)
>>> edges = sobel(lena)
>>> imshow(edges)
```
Zen of NumPy

- strided is better than scattered
- contiguous is better than strided
- descriptive is better than imperative (e.g. data-types)
- array-oriented is better than object-oriented
- broadcasting is a great idea -- use where possible
- vectorized is better than an explicit loop
- unless it’s complicated --- then use Cython / weave
- think in higher dimensions
Recent Developments in NumPy / SciPy

• Community growth (github)
• Addition of .NET (IronPython) support
  – NumPy as a core C-library -- needs work
  – NumPy and SciPy using Cython to build all extension modules
  – better tests and bugs closed
• Re-factoring of the ufunc-implementation as iterators, date-time, masks (Mark Wiebe)
  – expose the calculation pipeline that was only previously used in “worst-case” scenario.
  – first stages of calculation structure refactoring
NumPy Roadmap

• NumPy Object Structure
  – generalize shape --- data-array (labeled arrays)
  – generalize strides --- integrate Pandas, PIL, and more

• Data improvements
  – support for indexes on structured arrays
  – data-base-style persistence
  – improved data-types
    • date-times
    • missing data
    • enumerated data-types
    • reference data-types
    • derived data-types
Data Type Examples

```python
from numpy import dtype

class Stock(dtype):
    symbol = np.Str(4)
    open = np.Int(2)
    close = np.Int(2)
    high = np.Int(2)
    low = np.Int(2)
    @np.Int(2)
    def mid(self):
        return (self.high + self.low) / 2.0

class Person(dtype):
    weight = np.Float(64)
    age = np.Int(16)
    name = np.Str(100)
    gender = np.Enum('male', 'female')

    def isfat(self):
        return self.weight > 200

    def isold(self):
        return self.age > 39
```
Calculation improvements

• Functional iterators (Peter Wang)
• Iterator pipelines (Mark Wiebe)
• Generalized kernels for ufuncs
• Automatic compilation (mini JIT)
  – Call-sites
  – (fast) vectorize
  – more....
• Delayed computation
SciPy next steps

- Roadmap generation (move to github)
- Finish Cython migration
- Module cleanup continuing
- Merging of SciPy and Scikits into “Sci” packages
- Lots of work happening…
  - errorbars in polyfit
  - spectral algorithms in signal
  - improvements to ode and sparse
  - addition of pre-conditioners to sparse
High Level Languages Made Faster

- Cython / Weave
- FENICS / DOLFIN
- PyCuda (GPU)
- Copperhead (GPU)
- CorePy
Example Problem: Laplace equation

\[
\frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2} = 0
\]

\[
\begin{align*}
  u[0, :] &= 1 \\
  u[-1, :] &= 0 \\
  u[::, 0] &= 0 \\
  u[::, -1] &= 0
\end{align*}
\]
Current winner: Pure Fortran 90

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (sec)</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Python</td>
<td>202</td>
<td>0.03</td>
</tr>
<tr>
<td>NumPy</td>
<td>5.56</td>
<td>1</td>
</tr>
<tr>
<td>PyPy</td>
<td>4.71</td>
<td>1.18</td>
</tr>
<tr>
<td>Weave</td>
<td>2.42</td>
<td>2.23</td>
</tr>
<tr>
<td>Cython</td>
<td>2.21</td>
<td>2.52</td>
</tr>
<tr>
<td>Looped Fortran</td>
<td>2.19</td>
<td>2.53</td>
</tr>
<tr>
<td>Vectorized Fortran</td>
<td>1.42</td>
<td>3.92</td>
</tr>
<tr>
<td>Main Fortran (vec)</td>
<td>~0.8</td>
<td>~7</td>
</tr>
</tbody>
</table>
How is it written in Fortran 90?

```fortran
subroutine for_update2(u, dx2, dy2)
  real(dp), intent(inout) :: u(:,:),
  real(dp), intent(in) :: dx2, dy2
  integer :: nx, ny
  nx = size(u, 1)
  ny = size(u, 2)
  u(2:nx-1,2:ny-1) = ((u(3:,2:ny-1)+u(:,ny-2,2:ny-1))*dy2 + &
                      (u(2:nx-1,3:) + u(2:nx-1,:ny-2))*dx2) / (2*(dx2+dy2))
end subroutine
```

Compare NumPy

```python
def num_update(u, dx2, dy2):
    u[1:-1,1:-1] = ((u[2:,1:-1]+u[:-2,1:-1])*dy2 + &
                    (u[1:-1,2:] + u[1:-1,:-2])*dx2) / (2*(dx2+dy2))
```
Question?

SciPy the community

PyPy
Question?

SciPy the Community

PyPy