• speeding up Python with inline C++
• Weave
• Platform-independent Scientific Plotting
• Chaco
• Computational Libraries for Python
• Scipy

Overview

Scientific Computing with Python

Eric Jones

Enthought, Inc.
<table>
<thead>
<tr>
<th>Supported Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Linear Algebra (scipy.linalg)</td>
</tr>
<tr>
<td>- Clustering Algorithms (scipy.cluster)</td>
</tr>
<tr>
<td>- Fast Execution (scipy.compiler)</td>
</tr>
<tr>
<td>- Distributed Computing (scipy.computing)</td>
</tr>
<tr>
<td>- Special Algorithms (scipy.special)</td>
</tr>
<tr>
<td>- General Algorithms (scipy.ga)</td>
</tr>
<tr>
<td>- Numerical Integration (scipy.integrate)</td>
</tr>
<tr>
<td>- General Plotting (scipy.pit and scipy.plot)</td>
</tr>
<tr>
<td>- Optimization (scipy.optimize)</td>
</tr>
<tr>
<td>- Fourier Transforms (scipy.fft)</td>
</tr>
<tr>
<td>- Signal Processing (scipy.signal)</td>
</tr>
</tbody>
</table>

Available at [www.scipy.org](http://www.scipy.org)  
Version 0.2 Alpha  
Open Source Python Style License  
(Many thanks to Travis Oliphant & Pearu Peterson)  
Developed by Enthought and Partners

**Overview**

Slides by Travis Oliphant and Eric Jones
>>> scipy.stats.mean(a)

>>> from scipy import *

>>> b = scipy.atleast_2d(a)

6

FINDING THE ROOTS OF A POLYNOMIAL (PROCEDURAL)

# 1.3x^2 + 4x + .6

>>> polynomial = [1.3,4,.6]

# plot values of the polynomial

>>> x = arange(-4,1,.05)

>>> y = polyval(polynomial,x)

>>> plt.plot(x,y,'-')

>>> plt.hold('on')

# find the roots and plot them

>>> r = roots(polynomial)

>>> r

array([-2.9187968+.0j,
       -0.15812627+.0j])

>>> s = polyval(polynomial,r)

# we need to plot real parts

# because numerical error causes

# tiny imaginary part

# we need to plot real parts

# Find the roots and plot them

# Plot the values of the polynomial

# Find the roots

# Plot the roots

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial

# Plot the polynomial
Bessel Example

\[ j_0(x) = \text{special.j0}(x) \]

Dense Matrix Functionality

- Inheritance
- Matrix, class
- Dense matrix
- Matrix class
- Also includes low-level sub-packages
- High-level linear algebra routines
- Based on ATLAS implementation of LAPACK

Special Functions

Jacobi, Hermite, Chebyshev, Legendre, Laguerre

Orthogonal weighting functions for Gaussian quadrature

 airy, ellipj, bessel, gamma, hypergeom, zeta, sin, etc.

Included over 100 functions.
• Supported distributions (50+):

  - Return mean, variance, and optionally skewness and kurtosis of distribution
  - Inverse survival function
  - Percent point function (inverse of CDF)
  - Survival function (1-CDF)
  - Cumulative distribution function
  - Probability distribution function

Statistics

- decomposition
  - Cholesky - cholesky
  - expm - matrix exponential
  - norm
  - norm - vector and matrix
  - qr - decomposition
  - lu decomposition
  - decomposition
  - svd - singular value
  - linalg - least-squares solver
- norm
  - eig - eigen vectors and
  - expm - matrix exponential
  - det - matrix determinant
  - inv - matrix inverse

numpy

Solve

Array

array([[0, 0.5, 1.0, 0.0],
       [0.0, 0.5, 0.0, 1.0],
       [1.0, 0.0, 0.0],
       [0.0, 1.0, 0.0]])

Array

ax = array([[1.0]], 0.0)

Array

a = 1.0

Array

b = 2.0

Array

a = 1.0
```
>>> def hat(xy):
    # -sinc function with minimum at (0,0)
    return \n    -\frac{1}{\sqrt{\pi}} \exp\left( \frac{-x^2}{\pi} \right)
    \frac{1}{\sqrt{\pi}} \exp\left( \frac{-y^2}{\pi} \right)
    \frac{1}{\sqrt{\pi}^2} \exp\left( \frac{-xy}{\pi} \right)

    # multiple gradient

    # -steepest descent with minimum at

    # constrained minimization

    # -modified Powell hybrid method

    # Non-linear equation solver

    # scalar function minimization only

    # 2D minimization using fmin()
MINIMIZE BESSEL FUNCTION

```python
>>> plt.plot(x,j1_min,'ro')
```

CODE

Solve for $x_1$, $x_2$, and $x_3$ in a non-linear system of equations.

Non-Linear Solver

Bounded Minimization
- Fourier transforms
  • Integrating a function given a callable Python object
    - integrate.romb()  
    - integrate.quad()  
    - integrate.simps()  
    - integrate.trapz()  
    - integrate.dblquad()  
  • Filtering window functions
    - fftshift()
  • Integrating an ordinary differential equation
    - integrate.odeint()
    - integrate.odepack()
  • Integrating a function from samples
    - integrate.tplquad()  
    - integrate.quadrature()  
    - integrate.romberg()
Filter Design

– remez()
– iirdesign()
– iirfilter()
– butter(), elliptic(), cheby1(), cheby2(), bessel()

Frequency Response

– freqz()
– freqs()

Image Processing

# Blurring using a median filter

```python
>>> lena = plt.imread('lena.png')
>>> lena = lena.astype('float32')
>>> plt.imshow(lena)
>>> fl = signal.medfilt2d(lena, [15, 15])
>>> plt.imshow(fl)
```

LENA IMAGE

MEDIAN FILTERED IMAGE
Eric Jones and Patrick Miller

Weave

```python
# Noise removal using wiener filter
>>> from RandomArray import normal
>>> ln = lena + normal(0, 32, shape(lena))
>>> cleaned = signal.wiener(ln)
>>> plt.plot(cleaned)

NOISY IMAGE
FILTERED IMAGE
```

Image Processing
Basic Pieces of Weave

- Handle dynamic/static typing issues.
- To be done once.
- Persist compiled functions so that it only has
  - Compile and load extension function
  - Error Handling
  - Convert Python types to C++/C++ Types (Type Converters)
  - Boilerplate Module Code
  - C/C++ Code Generation

Classes for building C++ extension modules in Python

- weave.ext-tools
- weave.accelerate()
- weave.blitz()

Translation of Numeric Array Expressions to C++ for Fast

- weave.blitte()
- weave´tirrte()

Compile (a subset of) Python code directly to C++

- weave.accelerate()
weave.polyline('polyline_code', dc, line)

"""
# call the weave version of the function.
"""

HDC hdc = (HDC) dc->GetHDC

polyline_code = ""

line = line.typeconv(Int)

if (line.typecode() != Int or not line.iscontiguous())
    assert (len(shp) == 2 and shp[1] == 2)

    shape = line.shape

"""

def polyline(dc, line):

example

```cpp
>>> a='qwerty'
1
>>> weave.inline(polyline_code,
['dc','line'])
```

```python
if (line.typecode() != Int or not line.iscontiguous())
    assert (len(shp) == 2 and shp[1] == 2)

    shape = line.shape

```
MAXWELL'S EQUATIONS: FINITE DIFFERENCE TIME DOMAIN (FDTD),

\[\begin{align*}
\frac{\partial E}{\partial t} &= \nabla \times H - j\omega \mu_0 \sigma E, \\
\frac{\partial H}{\partial t} &= -\nabla \times E - j\omega \varepsilon_0 \mu_0 c^2 H.
\end{align*}\]

**Python Version of Same Equation, Pre-Calculated Constants**

```python
for i in range(len_x):
    for j in range(len_y):
        for k in range(len_z):
            tmp2[j] = tmp1[j] + c[j];
```

**C Code**

```c
for(i=0; i < len_a; i++)
    for(j=0; j < len_b; j++)
        for(k=0; k < len_c; k++)
            c[i] = a[i] + b[i] + c[i];
```

**Update of X Component of Electric Field**

```c
tmp2 = malloc(len_c * el_sz);
```
>>> import parser
weave.blitz
['testlist',
['test',
['and_test',
['not_test',
['factor',
['power',
['atom',
['NAME', 'd']]
]
]]]]],
['NEWLINE', ''
]
['ENDMARKER', ''
]
>>> expr = "ex[:,1:,1:] = ca_x[:,1:,1:] * ex[:,1:,1:]"
['NEWLINE', ''
]
['ENDMARKER', ''
]
>>> pprint.pprint(sym_list)
['EQUAL', '='
]
['file_input', 'foo.py']

## NUMERIC EXPRESSION AST

entourage

2.

3. Load completed module and execute code

2. Complete with gc using array variables in local scope

1. Translating expression to Blitz++ expression

```blitz
\n\n```

```c

```

```

```c

```

WeaveBlitz

WeaveBlitz compiles array expressions to C++ code using the Blitz++ library.

WeaveBlitz version of same equation

WeaveBlitz

entourage
Weave blitz benchmarks

**enough**

```c++
[[T-::T::]A = [:T::][:T::]A * [:T::][:T::]x - [:T::][:T::]A' * [:T::][:T::]x - 
  [:T::][:T::]x' * [:T::][:T::]x - [:T::][:T::]x - [:T::][:T::]x - [:T::][:T::]x]
```

**Python**

```python
[[T':T::]A = [:T':T::]A * [:T':T::]x - [:T':T::]A' * [:T':T::]x - 
  [:T':T::]x' * [:T':T::]x - [:T':T::]x - [:T':T::]x - [:T':T::]x]
```
\[
\begin{align*}
\n\text{expr} &= \sum_{x, y} \left( u_{1:-1, 0:-2} + u_{1:-1, 2:} \right) dx^2 \\
&\quad + \sum_{x, y} \left( u_{1:-1, 1:-1} \right) = \sum_{x, y} \text{expr} \\
\text{old} &= \text{old} + \text{diff}^2
\end{align*}
\]
### Laplace Benchmarks

<table>
<thead>
<tr>
<th>Method</th>
<th>Run Time (sec)</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure C++ Program</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Python/Fortran (with f2py)</td>
<td>2.90</td>
<td>2.4</td>
</tr>
<tr>
<td>weave.inline (easy)</td>
<td>6.74</td>
<td>0.4</td>
</tr>
<tr>
<td>weave.blitz</td>
<td>28.4</td>
<td>0.4</td>
</tr>
<tr>
<td>Numeric</td>
<td>10.0</td>
<td>0.4</td>
</tr>
<tr>
<td>Python</td>
<td>187.0</td>
<td>0.02</td>
</tr>
</tbody>
</table>

**Speed**

**Up**

### Laplace and Laplace's Equation

The image contains code examples and performance data for benchmarking. The code snippet is written in Python and includes comments indicating performance metrics. The comments mention that the code was written to solve the Laplace equation, and the performance is compared to other methods such as Python/Fortran and C++. The table provides a comparison of run times for different methods. The Python implementation, when compiled with f2py, outperforms the C++ version by a factor of 2.4.

The code snippet includes a function `compute` that defines a formula for calculating a value based on input parameters. The function is then used to compute a value and print the result. The code also contains comments indicating the usage of debugging tools and mention of the code being used for writing the benchmark script.

### Compile and Run

To compile and run the code, you would need to have Python and f2py installed. The `make` command can be used to compile the code, and the resulting `laplace` executable can be run to see the performance results.

```python
# This is only used for debugging (this is only used for debugging)
# #time 120 "laplace.py" (this is only used for debugging)

# The code = "...

# ... Enthought
```

The image also contains a table that compares the speed and run time for different methods. The table shows that the Python/Fortran implementation is faster than the C++ implementation, with a speedup of 2.4 times. The table also includes comments about the use of `weave` and `f2py` in the Python/Fortran implementation.

The results of the benchmark are displayed at the bottom of the image, indicating that the Python implementation runs in 4.3 seconds, which is faster than the C++ implementation.

The code snippet also includes comments about the use of `f2py` and `weave` in the Python/Fortran implementation, and the use of debugging tools such as `time` and `gdb` for profiling and debugging.

Overall, the image provides a comparison of different methods for solving the Laplace equation, with a focus on performance and the use of Python and Fortran with appropriate libraries for optimization.
def sortedDictValues3(adict):
    keys = adict.keys()
    keys.sort()
    return map(adict.get, keys)

def c_sort(adict):
    assert(type(adict) == type({}))
    code =
    #line 21 "dict_sort.py"
    Py::List keys = adict.keys();
    Py::List items(keys.length());
    items = PyDict_GetItem(adict.ptr(),item);
    Py_XINCREF(item);
    PyList_SetItem(items.ptr(),i,item
    return_val= Py::new_reference_to(items);
    
    return inline_tools.inline(code,['adict'])
Chaco structure

- PDF
- Properties
- OpenGL
- PDF
- Tkinter
- wxPython
- Chaco

Support for high-quality hard copy output (via PDF).

- Works with different widget sets (wxPython, Tkinter).
- Cross-platform (Windows, Mac, Unix).
- Uses the Kiwa graphics API.
- Python based.
- A high-level object oriented plotting toolkit.
Properties are easy to define. Properties are defined as a class level dictionary:

```python
class person ( HasProperties ):
    __properties__ = {
        'name':   '',  
        'age':    Property( 1, PropertyRange( 0, 130 ) ),
        'weight': Property( 160.0, PropertyRange( 0.0, 1000.0 ) )
    }  
```

Properties are used like normal Python attributes:

```python
bill = person()
bill.name = 'William'
bill.age = 34
bill.weight = 'medium'  # Generates an exception, illegal value
```

Properties are "strongly typed" Python attributes.

- Easy to define.
- Provide a "user friendly" API.
- Provide excellent error diagnostics.
- Are flexible and open-ended.
- Support a powerful delegation (i.e. containment) model.
- Allow for easy creation of GUI "property sheets".
- Generate an exception, illegal value.
Properties are easy to define.

Some property examples:

```python
# Accepts any character string; default is '':
'\text': '',

# Can be any of the listed strings; default is 'small':
'size': [ 'small', 'medium', 'large' ],

# Can be 'small', 'sma', 's', 'me', …; default is 'small':
'size2': Property( 'medium', PropertyPrefixList( [ 'small', 'medium', 'large' ] ) ),

# Can be any integer value; default is 1:
'width': 1,

# Can be 'None' or a float between 0 and 10; default is 1:
'width2': Property( 1.0, None, PropertyRange( 0.0, 10.0 ) ),

# Provide excellent error diagnostics

class hat ( HasProperties ):
    __properties__ = {
        'size': Property( 'medium', PropertyPrefixList( [ 'small', 'medium', 'large' ] ) )
    }

bowler = hat()

bowler.size = 's'          # This is OK
bowler.size = 'very large' # Produces this exception:
                          # File "test.py", line 10, in ?
                          # bowler.size = 'very large'
                          # File "properties.py", line 1114, in __setattr__
                          # raise PropertyError, excp
                          # properties.PropertyError: The 'size' property of a hat instance
                          # must be 'small' or 'medium' or 'large' (or any unique prefix),
                          # but a value of 'very large' was specified.
                          # raise PropertyError, excp
                          # File "properties.py", line 1114, in __setattr__
                          # raise PropertyError, excp
                          # File "properties.py", line 1114, in __setattr__
                          # raise PropertyError, excp
                          # File "test.py", line 10, in ?
                          # bowler.size = 'very large'
                          # File "test.py", line 10, in ?
                          # Traceback (most recent call last):
                          # File "test.py", line 10, in ?
                          # bowler.size = 'very large'
                          # File "properties.py", line 1114, in __setattr__
                          # raise PropertyError, excp
                          # properties.PropertyError: The 'size' property of a hat instance
                          # must be 'small' or 'medium' or 'large' (or any unique prefix),
                          # but a value of 'very large' was specified.
```
Support a powerful delegation model

- Objects can set `local` values which override object layers.
- Delegation can be chained through multiple object layers if desired.
- Each property can delegate to a different another object.
- The value of a property can be delegated to the delegated value.
The high-level Chaco classes

• PlotWindow
• PlotGroup
• PlotIndexed
• PlotCanvas
• PlotValue
• PlotTitle
• PlotOverlay
• PlotAxis

High-level property sheet editors available

Although all, or a specified subset, of an object's properties can be edited interactively, a pop-up editor is provided.

User sees a pop-up property sheet editor for an object's properties.

The property sheet is created by delegating to the containing PlotWindow object, which is GUI toolkit specific.

All Chaco objects have an `edit` method for interactive editing of object properties.
Each Chaco object can have its own context.

PDF files can be generated for the entire window.

Script files.

Plot object configurations can be loaded from Python statements.

Command shell window can be used to enter editors.

Flexible context-sensitive pop-up menus and property editors.

Chaco objects can be modified using context-sensitive pop-up menus and property sheets.

Interactive demonstration of the Chaco toolkit.

• Chaco objects can be modified using context-sensitive pop-up menus and property sheet editors.

• Plot object configurations can be loaded from 'script' files.

• PDF files can be generated for the entire window or for any Chaco object in the window.

• Command shell window can be used to enter Python statements.

• Top level PlotWindow allows choice of:

  - Editing top-most object's properties.
  - Displaying/merging all stacked object's context menus.
  - Selecting/editing all stacked object's properties.
  - Displaying top-most object's context menu.
  - Top-level PlotWindow allows choice of: