Python in HPC

Numpy, MatPlotLib, SciPy

Andy R. Terrel

The Texas Advanced Computing Center

April 23, 2012

- Introduction
- NumPy: Array Datastructures
- MatPlotLib: Easy Visualization
- SciPy: Scientific toolkit

- Introduction
- NumPy: Array Datastructures
- MatPlotLib: Easy Visualization
- SciPy: Scientific toolkit

Instead of hello world, let's see the MatPlotLib histogram.

From ipython --pylab or EPD on windows

```
In [1]: hist(randn(10000), 100)
```

or

From regular python

```
>>> from numpy.random import *
>>> from pylab import *
>>> hist(randn(10000), 100)
>>> show()
```

Help commands in IPython

```
In [2]: hist ?

In[3]: hist ??

In[4]: import pylab

In[5]: pylab ??

#Learn more about IPython
In[6]: %magic
```

Help commands in Python

```
>>> help(hist)

>>> import pylab

>>> help(pylab)
```

Timeline

- Began in 1989 at CWI as a successor of "ABC"
- Initially developed by Guido van Rossum, now Benevolent Dictator for Life
- Primarily thought of initally as a teaching language
- Early release cylce every 6 months or so, currently every 2 years
- Currently core is moving to 3.0, but most libraries only work on 2.7

Technical Specifications

- Python is a "multi-paradigm" language:
  - Imperative
  - Object Oriented
  - Dynamically Typed – Interpreted
  - Almost Functional
- This means that programmers can work in a variety of styles, freely intermixing constructs from different paradigms

Duck Typing

- If it looks like a duck, then it is a duck.
- Dynamically evaluates types
- Type (and name) errors not caught until runtime
  - Unit testing becomes more important
  - Good IDE can catch many standard errors

Speed

- Running Python code is usually 10X slower than C
- Writing Python code is usually much much much faster than writing C
- Reading Python code is usually much much much faster than reading C
- Write Python, profile, refine in C
- Several projects are working to speed up Python (PyPy, Cython, ...)

Gotchas

- No multithreading support
- Import problem

But who cares, let's start hacking!

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.

- Special cases aren't special enough to break the rules.
  - Although practicality beats purity.
- Errors should never pass silently.
  - Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one– and preferably only one –obvious way to do it.
  - Although that way may not be obvious at first unless you're Dutch.

- Now is better than never.
  - Although never is often better than right now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- NameSpaces are one honking great idea – let's do more of those!

- Introduction
- NumPy: Array Datastructures
- MatPlotLib: Easy Visualization
- SciPy: Scientific toolkit

How slow is Python? Let's add one to a million numbers.

Using lists

```
In [15]: lst = range(1000000)

In [16]: %timeit [i + 1 for i in lst]
10 loops, best of 3: 65.6 ms per loop
```

Why is Python slow?

- Dynamic typing requires lots of metadata around variable.
- Python uses heavy frame objects during iteration

Solution:

- Make an object that has a single type and continuous storage.
- Implement common functionality into that object to iterate in C.

How slow is Python? Let's add one to a million numbers.

Using lists

```
In [15]: lst = range(1000000)

In [16]: %timeit [i + 1 for i in lst]
10 loops, best of 3: 65.6 ms per loop
```

Using NumPy

```
In [18]: arr = arange(1000000)

In [19]: %timeit arr + 1
100 loops, best of 3: 2.91 ms per loop
```

History of NumPy

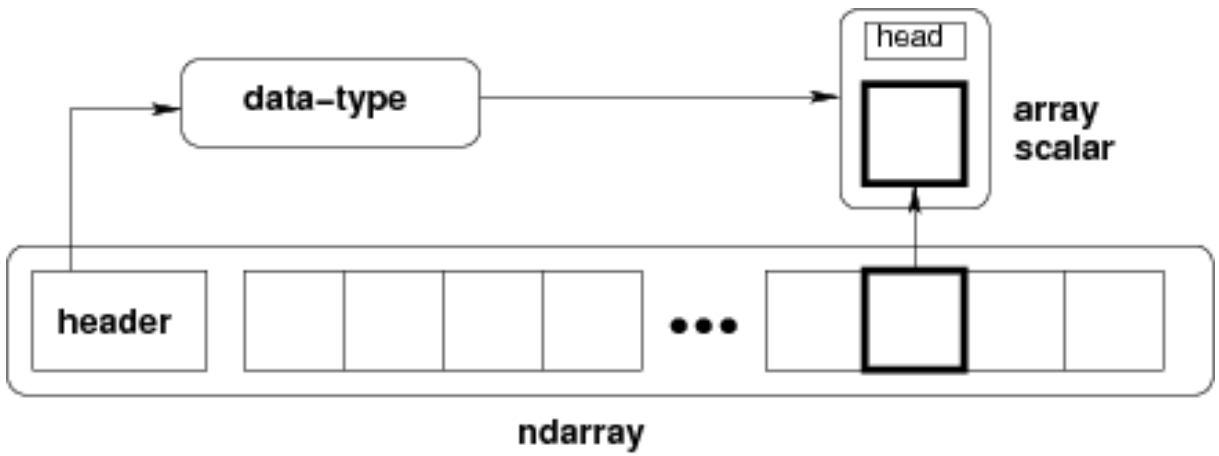
- Features
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities
- Development
  - Based originally on `Numeric` by Jim Hugunin
  - Also based on `NumArray` by Perry Greenfield
  - Written by Travis Oliphant to bring both feature sets together

What makes an array so much faster?

- Data layout
  - homogenous: every item takes up the same size block of memory
  - single data-type objects
  - powerful array scalar types
- universal function (ufuncs)
  - function that operates on ndarrays in an element-by-element fashion
  - vectorized wrapper for a function
  - built-in functions are implemented in compiled C code

Data layout

- homogenous: every item takes up the same size block of memory
- single data-type objects
- powerful array scalar types



universal function (ufuncs)

- function that operates on ndarrays in an element-by-element fashion
- vectorized wrapper for a function
- built-in functions are implemented in compiled C code

python function v ufunc

```
In [31]: %timeit [sin(i)**2 for i in arr]
1 loops, best of 3: 4.32 s per loop

In [32]: import numpy as np

In [33]: %timeit np.sin(arr)**2
10 loops, best of 3: 20.8 ms per loop
```

Creating array

```
In [5]: x = array([2, 3, 12]) # Create from list

# mix of tuple, lists, and types
In [6]: x = np.array([[1,2.0],[0,0],(1+1j,3.)])

In [7]: x = arange(-10, 10, 2, dtype=float)

In [8]: np.linspace(1., 4., 6)

In [9]: np.indices((3,3))

In [10]: fromfile('foo.dat')
```

Array API

```
In [16]: x = arange(9).reshape(3,3)
In [19]: x[:, 0]
Out[19]: array([0, 3, 6])
In [20]: x.shape
Out[20]: (3, 3)
In [21]: x.strides
Out[21]: (24, 8)
In [22]: y = x[::2, ::2]
In [25]: y[0,0] = 100
In [26]: x[0, 0]
Out[26]: 100
```

Finite Differences

computing blocks

```
In [38]: x = np.arange(0, 20, 2)

In [40]: y = x**2

In [42]: dy_dx = ((y[1:] - y[:-1]) / (x[1:] - x[:-1]))
In [43]: dy_dx
Out[43]: array([ 2,  6, ... 30, 34])

In [44]: dy_dx_c = ((y[2:] - y[:-2])/(x[2:] - x[:-2]))
In [45]: dy_dx_c
Out[45]: array([ 4,  8, ... 28, 32])
```

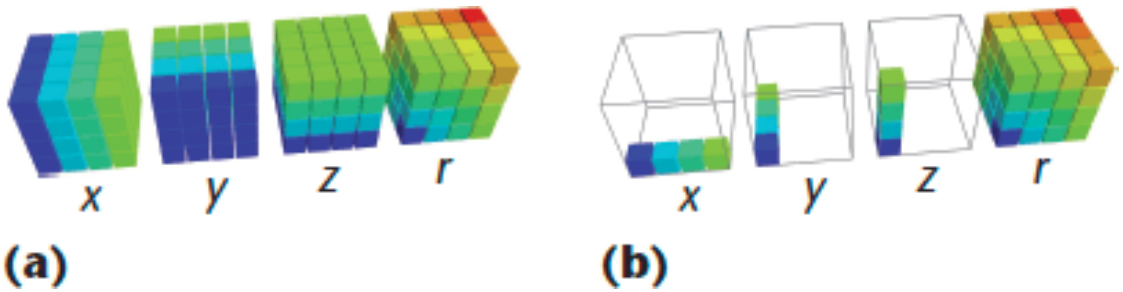
Computing a 3D grid of distances $R_{ijk} = \sqrt{(i^2 + j^2 + k^2)}$

With temporary matrices

```
In [44]: i, j, k = np.mgrid[-100:100, -100:100, -100:100]
In [45]: print(i.shape, j.shape, k.shape)
((200, 200, 200), (200, 200, 200), (200, 200, 200)

In [46]: R = np.sqrt(i**2 + j**2 + k**2)
In [47]: R.shape
Out[47]: (200, 200, 200)
```

Computing a 3D grid of distances $R_{ijk} = \sqrt{(i^2 + j^2 + k^2)}$



Computing a 3D grid of distances $R_{ijk} = \sqrt{(i^2 + j^2 + k^2)}$

With temporary matrices

```
# Construct the row vector that runs from -100 to +100
In [47]: i = np.arange(-100, 100).reshape(200, 1, 1)
#  Construct the column vector
In [48]: j = np.reshape(i, (1, 200, 1))
# Construct the depth vector
In [49]: k = np.reshape(i, (1, 1, 200))

In [50]: R = np.sqrt(i**2 + j**2 + k**2)
In [51]: R.shape Out[49]: (200, 200, 200)
```

Computing a 3D grid of distances $R_{ijk} = \sqrt{(i^2 + j^2 + k^2)}$

With temporary matrices

```
# Shorthand for i, j, k broadcasting
In [52]: i, j, k = ogrid[-100:100, -100:100, -100:100]

In [53]: R = np.sqrt(i**2 + j**2 + k**2)
In [53]: R.shape Out[49]: (200, 200, 200)
```

Numpy has a sophisticated view of data.

bool	int	int8
int16	int32	int64
uint8	uint16	uint32
uint64	float	float16
float32	float64	complex
complex64	complex128	

Using NumPy types

```
In[55]: np.array([1, 2, 3], dtype='f')
array([ 1.,  2.,  3.], dtype=float32)
In[56]: z.astype(float)
array([  0.,  1.,  2.])
In[57]: np.int8(z)
array([0, 1, 2], dtype=int8)
In[58]: d = np.dtype(int)
In[59]: d
dtype('int32')
In[60]: np.issubdtype(d, int)
True
In[61]: np.issubdtype(d, float)
False
```

Structured data

```
In[62]: x = np.zeros((2,),dtype=('i4,f4,a10'))
In[63]: x[:] = [(1,2.,'Hello'),(2,3.,"World")]
In[64]: dt = dtype([('time', uint64),
...         ('pos', [
...            ('x', float),
...            ('y', float))])
In[65]: x = np.array([
...         (100, ( 0, 0.5),
...         (200, ( 0, 10.3),
...         (300, (5.5, 15.1)],
...         dtype=dt)
In[66]: x['time']
In[67]: x[ x['time'] >= 200 ]
```

Linear Algebra

```
In [33]: dot(arange(10), arange(10))
Out[33]: 285

In [35]: dot(arange(9).reshape(3,3), arange(3))
Out[35]: array([ 5, 14, 23])

In [36]: dot(arange(9).reshape(3,3),
...           arange(9).reshape(3,3))
Out[36]:
array([[ 15,  18,  21],
       [ 42,  54,  66],
       [ 69,  90, 111]])
```

Other libraries

```
In [37]: fft ?

In [38]: linalg ?

In [39]: random ?
```

The `array` object has an order flag, see `array?`.

1. Create a 100x100 random row and column 2D arrays.
2. Time the dot of two row arrays.
3. Time the dot of two column arrays.
4. Time the dot of the combination.

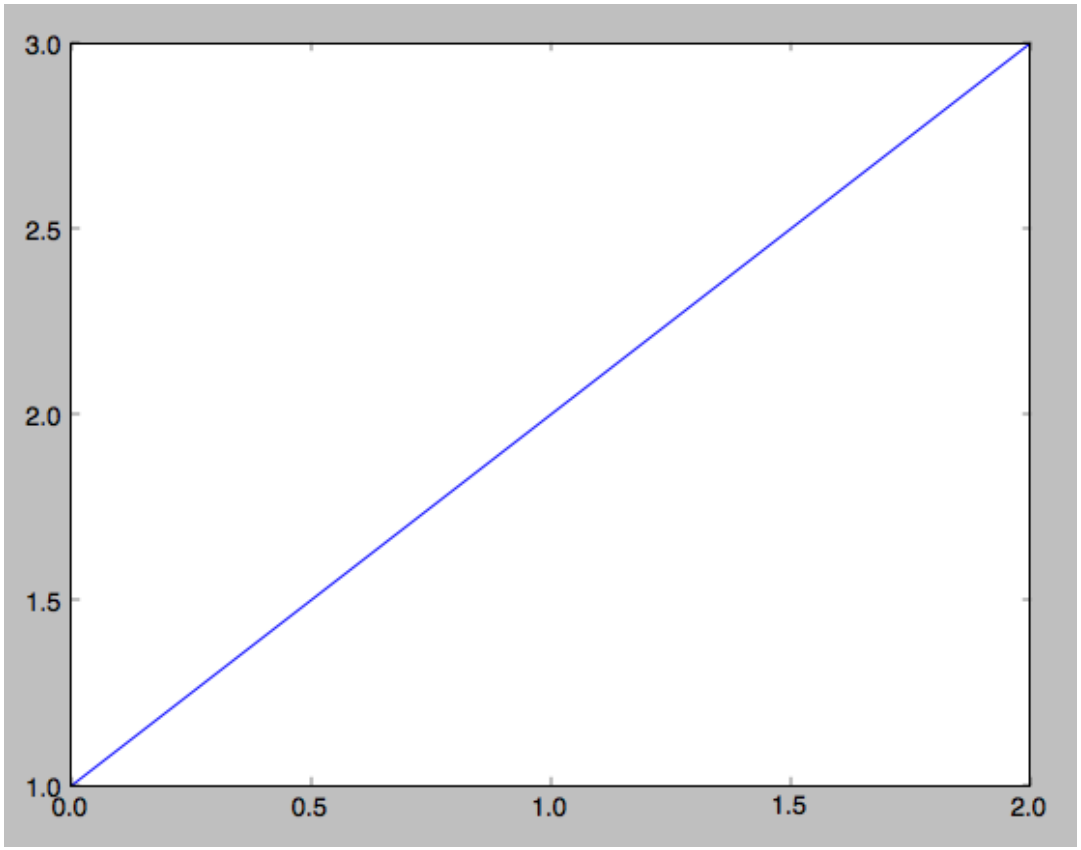
Using only slicing and indexing. Create a 2D laplacian operator. How do you handle the boundaries?

- Introduction
- NumPy: Array Datastructures
- MatPlotLib: Easy Visualization
- SciPy: Scientific toolkit

matplotlib is a python 2D plotting library which:

- produces publication quality figures,
- interactive environments,
- generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc
- customize all look and feel

```
In [62]: plot([1,2,3])
In [63]: show()
```

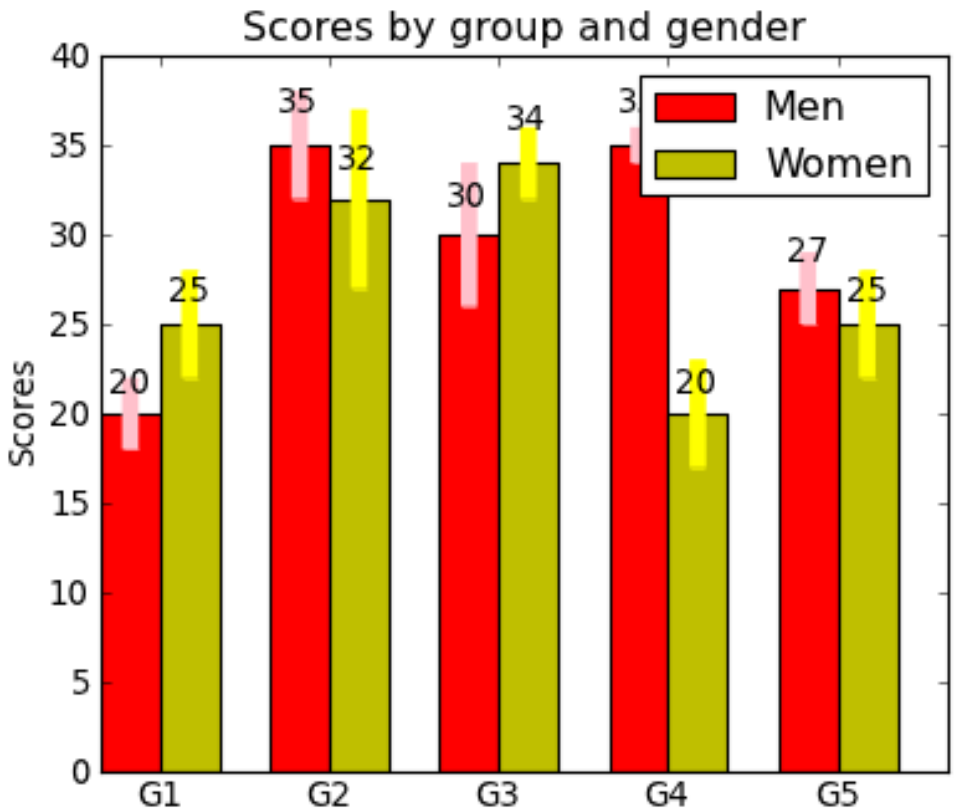


```
In [69]: hold(False)

In [70]: plot([1,2,3], [.2, .3, .4], 'g+')

In [71]: plot([1,2,3], [.2, .3, .4], 'g+',
...            [2, 4, 6], [.2, .3, .4], 'bt')

In [72]: plot([1,2,3], [1,2,3], 'go-', label='line 1',
...            linewidth=2)
In [74]: hold(True)
In [75]: plot([1,2,3], [1,4,9], 'rs',  label='line 2')
In [76]: axis([0, 4, 0, 10])
In [77]: legend()
In [78]: save_fig(``my_file.png'')
```


Scores by group and gender
Men
Women
Scores
0
5
10
15
20
25
30
35
40
G1
G2
G3
G4
G5
20
25
35
32
30
34
3
20
27
25

```
N = 5
menMeans = (20, 35, 30, 35, 27)
menStd =   (2, 3, 4, 1, 2)

ind = np.arange(N)  # the x locations for the groups
width = 0.35        # the width of the bars

rects1 = bar(ind, menMeans, width,
         color='r',
         yerr=menStd,
         error_kw=dict(elinewidth=6, ecolor='pink'))
```

```
womenMeans = (25, 32, 34, 20, 25)
womenStd =   (3, 5, 2, 3, 3)
rects2 = plt.bar(ind+width, womenMeans, width,
                 color='y',
                 yerr=womenStd,
                 error_kw=dict(elinewidth=6, ecolor='yellow'))
```

```
# add some
plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.xticks(ind+width, ('G1', 'G2', 'G3', 'G4', 'G5') )

plt.legend( (rects1[0], rects2[0]), ('Men', 'Women') )
```

```
# add some
plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.xticks(ind+width, ('G1', 'G2', 'G3', 'G4', 'G5') )

plt.legend( (rects1[0], rects2[0]), ('Men', 'Women') )
```

```
def autolabel(rects):
    # attach some text labels
    for rect in rects:
        height = rect.get_height()
        plt.text(rect.get_x()+rect.get_width()/2.,
                 1.05*height, '%d'%int(height),
                 ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

plt.show()
```

- Using the plot command make a scatter plot (hint just use 'bo' as the format).
- Use the semilogx, semilogy, and loglog plots.
- See http://matplotlib.sourceforge.net/gallery.html and make a plot that you think is interesting.

- Introduction
- NumPy: Array Datastructures
- MatPlotLib: Easy Visualization
- SciPy: Scientific toolkit

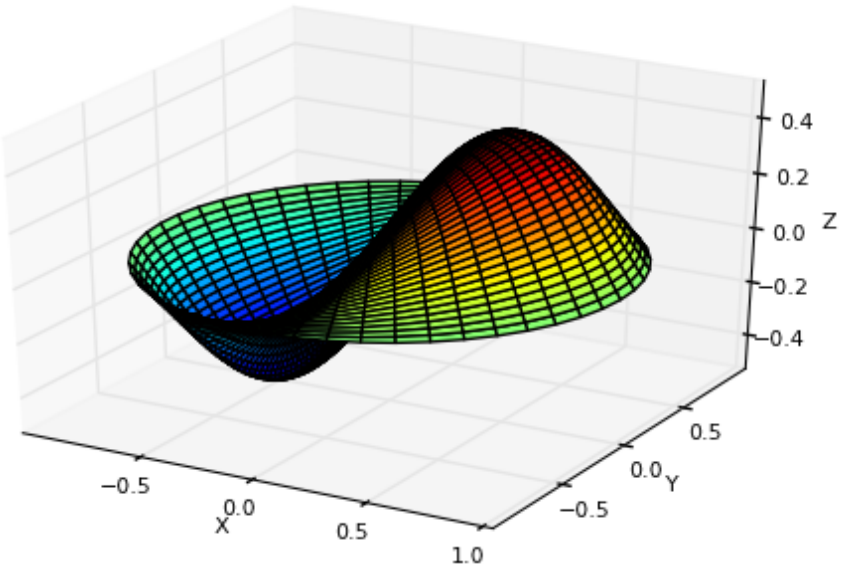
SciPy

- mathematical algorithms and convenience functions built on the Numpy,
- organized into subpackages covering different scientific computing domains,
- a data-processing and system-prototyping environment rivaling sytems such as MATLAB, IDL, Octave, R-Lab, and SciLab

- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Sparse Eigenvalue Problems with ARPACK
- Statistics (scipy.stats)
- Multi-dimensional image processing (scipy.ndimage)
- File IO (scipy.io)
- Weave (scipy.weave)

Bessel functions

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0 \quad (1)$$



```
>>> from scipy import *
>>> from scipy.special import jn, jn_zeros
>>> def drumhead_height(n, k, distance, angle, t):
...     nth_zero = jn_zeros(n, k)
...     return cos(t)*cos(n*angle)*\
...            jn(n, distance*nth_zero)
>>> theta = r_[0:2*pi:50j]
>>> radius = r_[0:1:50j]
>>> x = array([r*cos(theta) for r in radius])
>>> y = array([r*sin(theta) for r in radius])
>>> z = array([drumhead_height(1, 1, r, theta, 0.5) \
...             for r in radius])
```

```
>>> import pylab
>>> from mpl_toolkits.mplot3d import Axes3D
>>> from matplotlib import cm
>>> fig = pylab.figure()
>>> ax = Axes3D(fig)
>>> ax.plot_surface(x, y, z, \
...     rstride=1, cstride=1, cmap=cm.jet)
>>> ax.set_xlabel('X')
>>> ax.set_ylabel('Y')
>>> ax.set_zlabel('Z')
>>> pylab.show()
```

Using quad

```
>>> from scipy.integrate import quad
>>> def integrand(t,n,x):
...     return exp(-x*t) / t**n

>>> def expint(n,x):
...     return quad(integrand, 1, Inf, args=(n, x))[0]

>>> vec_expint = vectorize(expint)

>>> vec_expint(3,arange(1.0,4.0,0.5))
array([ 0.1097,  0.0567,  0.0301,  0.0163,  0.0089,  0.004
>>> special.expn(3,arange(1.0,4.0,0.5))
array([ 0.1097,  0.0567,  0.0301,  0.0163,  0.0089,  0.004
```

Using dblquad

```
>>> result = quad(lambda x: expint(3, x), 0, inf)
>>> print result
(0.33333333324560266, 2.8548934485373678e-09)

>>> I3 = 1.0/3.0
>>> print I3
0.333333333333

>>> print I3 - result[0]
8.7730656073 1e-11
```

Using dblquad

```
>>> from scipy.integrate import quad, dblquad
>>> def I(n):
...     return dblquad(lambda t, x: exp(-x*t)/t**n,
..                    0, Inf,
...                   lambda x: 1, lambda x: Inf)

>>> print I(4)
(0.25000000000435768, 1.0518245707751597e-09)
>>> print I(3)
(0.33333333325010883, 2.8604069919261191e-09)
>>> print I(2)
(0.49999999999857514, 1.8855523253868967e-09)
```