Introduction to NumPy

Scientific Computation With Python

overview

- there are lots of packages available for python

- in particular for scientific use:

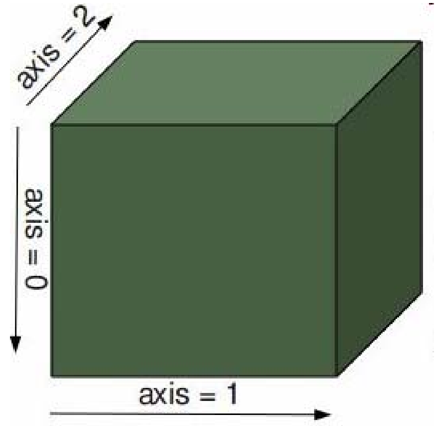
  - **Numpy** (handling and manipulation of large arrays)
  - **SciPy** (lots of user-friendly and efficient numerical routines)
  - **Matplotlib** (2D plotting library)

  => provide an open source alternative to MATLAB
  (http://www.scipy.org/NumPy_for_Matlab_Users)

- available at
  http://numpy.scipy.org/
  http://matplotlib.sourceforge.net/

numpy array

- provides a powerful N-dimensional array object:
  - table of items of **same type**
  - **more efficient** than python lists



- can be directly created from lists

```
>> import numpy as np
>> a = np.array ( [1,2,3,4] )
```

$$\vec{a} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

```
>> import numpy as np
>> M = np.array ([ [1,2],[3,4] ] )
```

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

numpy array creation

- arange() : improved range() – function

```
>> a = np.arange ( 0, 0.4, 0.1 )
```

$$\vec{a} = \begin{pmatrix} 0.0 \\ 0.1 \\ 0.2 \\ 0.3 \end{pmatrix}$$

- zeros(), ones() : fill with zeros or ones

```
>> M1 = np.ones ( (2,2) )
>> M2 = np.zeros ( (2,3) )
```

! the shape has to be specified !

$$M1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad M2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- eye() : identity

```
>> M = np.eye ( 3 )
```

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- rand(), randn() : random matrix

```
>> M = np.random.rand ( 2,2 )
```

$$M = \begin{pmatrix} 0.09833 & 0.94981 \\ 0.01581 & 0.34234 \end{pmatrix}$$

indexing & slicing

- 1-D arrays can be index, sliced and iterated like lists

```
>> a = np.arange ( 0, 0.4, 0.1 )

>> a[0]
0.0

>> a[1:3]
[0.1, 0.2]
```

$$\vec{a} = \begin{pmatrix} 0.0 \\ 0.1 \\ 0.2 \\ 0.3 \end{pmatrix}$$

- N-D arrays can have one index per axis

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

```
>> M [0,2]
2

>> M [:,1]
[1,4,7]

>> M[:-1,::-1]
[  [2,1,0],
   [5,4,3] ]
```

- not specified axes considered complete slices

```
>> M[1] # eqv. To M[1,:]
[3,4,5]
```

unary operations

- many unary operations are implemented as methods of array class:

```
>> M = np.array ( [ [1,2], [3,4] ] )
```

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

```
>> M.sum()
10

>> M.mean()
2.5

>> M.max()
4
```

"handle array like lists"

```
>> M.sum( axis=0 )
array( [4,6] )

>> M.mean( axis=0 )
array ( [2, 3] )

>> M.max( axis=0 )
array ([3,4])
```

axis can be specified

More examples: argmax(), argsort(), conjugate(), cumsum(), conj(), imag(), real(), transpose(), …

properties of arrays

```
a = np.array( [ [0,1,2,3,4], [5,6,7,8,9] ] )

a.shape
(2,5)
```

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

- access attributes of numpy array:
  - a.shape (the dimensions) is (2,5)
  - a.ndim (number of axis) is 2
  - a.size (total number of elements) is 10
  - a.dtype (datatype of elements) is int64
  - a.itemsize (size of element in bytes) is 8
- additonal datatypes:
  - int8, int16, int32, int64
  - float32, float64, float96
  - complex64, complex128, complex192
- can be specified with dtype

```
a = np.array( [ [0,1,2,3,4], [5,6,7,8,9] ], dtype=np.complex64 )
```

basic operations

- arithmetic operations apply **elementwise**
- **new** array created

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$c = 5$$

```
>> A * B
[ [5,12],
  [21,32] ]

>> A – c
[ [5,12],
  [21,32] ]
```

```
>> A ** B
[ [1, 64],
  [2187, 65536] ]

>> A<3
[ [True, False],
  [False, False] ]
```

matrix product:

```
>> np.dot (A,B)
[ [19,22],
  [43,50] ]
```

some operators act in place (similar to C++):

```
>> A *= 2
>> print A
[ [2,4],
  [6,8] ]
```

reshaping

- reshaping an array (usually **no copy**):

$$\vec{x} = \left(\; 0, 1, \ldots, 9 \; \right)$$

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

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

```
>> x = np.arange (10)

>> M = x.reshape(2,5)

>> N = x.reshape(5,-1)

>> O = x.reshape(3,-1)
==> returns an error
```

"-1" means whatever needed

modifying x, M or N modifies all objects

working with copies:

$$O = \begin{pmatrix} 40 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

```
>> x = np.arange (10)

>> O = x.copy().reshape(2,5)

>> O[0,0] = 40
```

x is unchanged

resizing

- resize an array with resize():

```
>> X = np.arange(10)
>> X.resize(3,3)
```

```
>> Y = np.arange(7)
>> Y.resize(3,3)
```

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

$$Y = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 0 & 0 \end{pmatrix}$$

- references may impede resizing:

```
>> x = np.arange (10)
>> M = x.reshape(2,5)

>> M.resize(3,3)
==> error: does not own data

>> x.resize(3,3)
==> error: referenced by other array

>> del M; x.resize(3,3)
==> this works
```

$$\vec{x} = \begin{pmatrix} 0, 1, \ldots, 9 \end{pmatrix}$$

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

alternatively use a copy:

```
>> y = x.copy().resize(3,3)
```

```
>> y = np.resize(x, (3,3) )
```

fancy indexing

- Indexing with arrays of indices

$$\vec{x} = \begin{pmatrix} 0 \\ 2 \\ 4 \\ 6 \\ 8 \end{pmatrix}$$

```
>> x = np.arange ( 0, 10, 2)

>> idx = np.array ( [0,4,4,2])

>> y = x [ idx ]

>> print y
[0, 8, 8, 4 ]
```

$$\vec{y} = \begin{pmatrix} 0 \\ 8 \\ 8 \\ 4 \end{pmatrix}$$

- also works with N-dimensional index arrays

```
>> x = np.arange ( 0, 10, 2)

>> idx = np.array ( [[0,4] , [4,2] ])

>> M = x [ idx ]

>> print M
[ [0, 8],
  [8, 4 ] ]
```

$$M = \begin{pmatrix} 0 & 8 \\ 8 & 4 \end{pmatrix}$$

fancy indexing II

- also can present higher dimensional index

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

```
>> A = np.arange (9).reshape(3,3)
>> i = np.array ( [0, 2] )
>> j = np.array ( [1,2] )
>> x = A [ i, j ]
>> B = A [ i,: ]
```

$$\vec{x} = \begin{pmatrix} 1 \\ 8 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 1 & 2 \\ 6 & 7 & 8 \end{pmatrix}$$

$$i = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$j = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

```
>> i = np.eye (2, dtype = int )
>> j = np.ones( (2,2), dtype = int)
>> C = A[ i, j]
```

$$C = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$$

fancy indexing III

- Boolean indexing, explicitly choose the elements

```
>> A = np.arange(1,5).reshape(2,2)

>> idx = np.array ( [ [True, False], [True, True] ])
>> x = A [ idx ]

>> idx = [ [True, False], [True, True] ]
>> y = A[idx]  # !be careful

>> idx = A<3
>> z = A[idx]
```

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\vec{x} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix} \qquad \vec{y} = \begin{pmatrix} 4 \\ 2 \end{pmatrix} \qquad \vec{z} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

universal functions

- Numpy provides a useful set of mathematical functons. They are called „universal functions“ and work **elementwise**.

```
>> x = np.arange(5)

>> np.sqrt (x)
[0., 1., 1.41421, 1.730225]
```

```
>> x = np.arange(5)

>> np.exp (x)
[1., 2.71828, 7.3891, 20.0855]
```

- Fast and very usefull for data processing,
  - eg. consider you have a list with data

```
>> absdata = [abs(i) for i in data  ] #have to iterate over list

>> absdata = np.abs(data) #can direclty manipulate array
```

arccos, arctan, ceil, conjugate, cos, exp, fabs, floor, fmod, log, log10, sin, sinh, sqrt, …

Scipy package

- Based on the *numpy* package *scipy* provides advanced methods for science and engineering:
  - Constants (scipy.constants)
  - Fourier transforms (scipy.fftpack)
  - Integration and ODEs (scipy.integrate)
  - Interpolation (scipy.interpolate)
  - Linear algebra (scipy.linalg)
  - Orthogonal distance regression (scipy.odr)
  - Optimization and root finding (scipy.optimize)
  - Signal processing (scipy.signal)
  - Special functions (scipy.special)
  - Statistical functions (scipy.stats)
  - C/C++ integration (scipy.weave)
  - And more …

- Check: http://docs.scipy.org/doc/

Example 1: some numpy

- The task:
  1. How many of the first million numbers are dividable by 7?

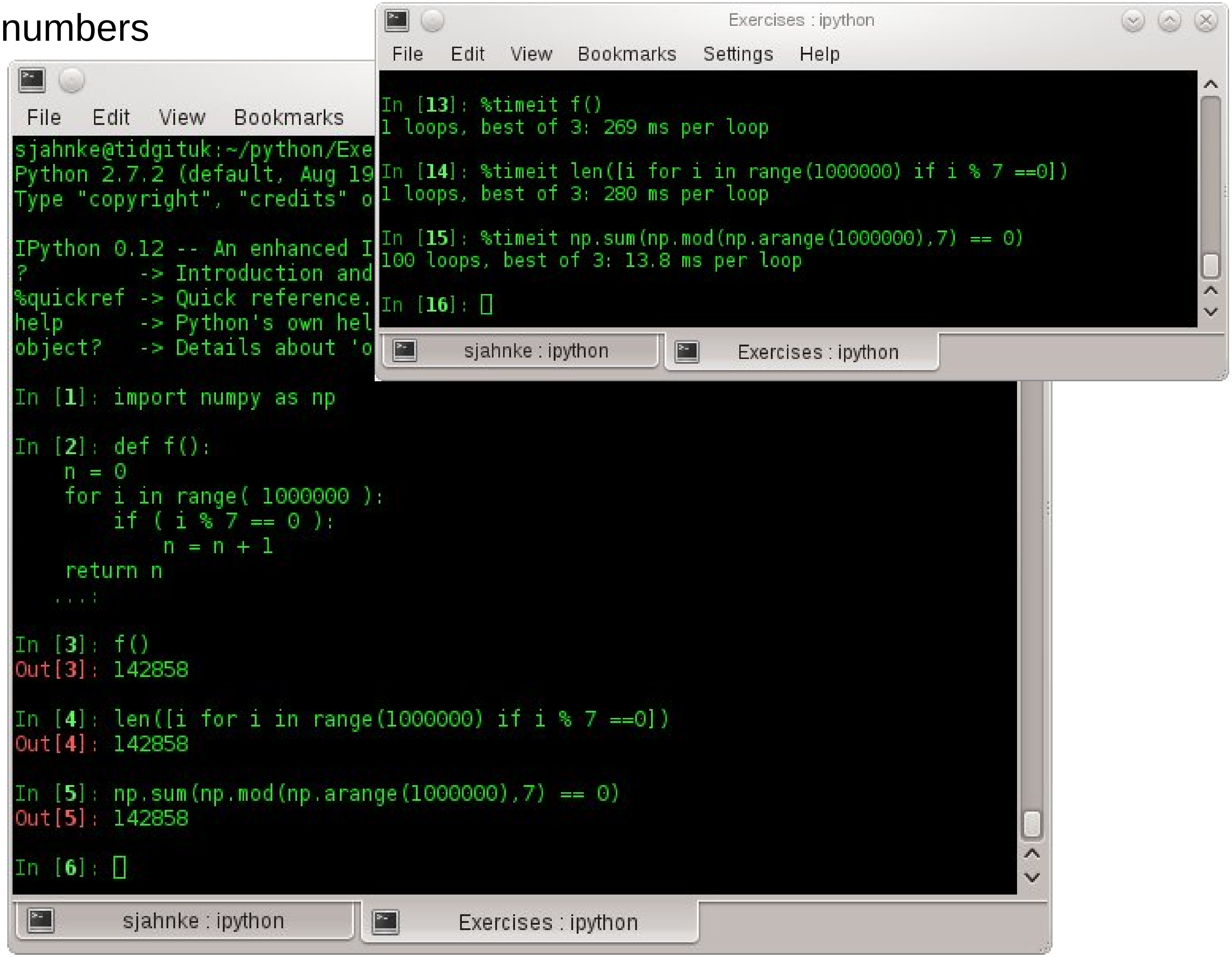
     (Yes, please really test the numbers ☺)

     (a) Implement a function using a for loop to do the task.
     (b) Use list comprehension to do the task.
     (c) Use numpy and avoid loops at all.

  2. Use the “magic function” **%timeit** to compare the runtimes of the three versions.

  3. How many of the first million numbers are dividable by 3 and 7?

numbers



Exercises : ipython
File Edit View Bookmarks Settings Help
In [13]: %timeit f()
1 loops, best of 3: 269 ms per loop
In [14]: %timeit len([i for i in range(1000000) if i % 7 ==0])
1 loops, best of 3: 280 ms per loop
In [15]: %timeit np.sum(np.mod(np.arange(1000000),7) == 0)
100 loops, best of 3: 13.8 ms per loop
In [16]:
sjahnke : ipython
Exercises : ipython
File Edit View Bookmarks
sjahnke@tidgituk:~/python/Exe
Python 2.7.2 (default, Aug 19
Type "copyright", "credits" o
IPython 0.12 -- An enhanced I
? -> Introduction and
%quickref -> Quick reference.
help -> Python's own hel
object? -> Details about 'o
In [1]: import numpy as np
In [2]: def f():
n = 0
for i in range(1000000):
if (i % 7 == 0):
n = n + 1
return n
...:
In [3]: f()
Out[3]: 142858
In [4]: len([i for i in range(1000000) if i % 7 ==0])
Out[4]: 142858
In [5]: np.sum(np.mod(np.arange(1000000),7) == 0)
Out[5]: 142858
In [6]:
sjahnke : ipython
Exercises : ipython

numbers

```
Exercises : ipython
File  Edit  View  Bookmarks  Settings  Help
sjahnke@tidgituk:~/python/Exercises> ipython
Python 2.7.2 (default, Aug 19 2011, 20:41:43) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: A = np.mod(np.arange(1000000),3)

In [3]: B = np.mod(np.arange(1000000),7)

In [4]: (A + B == 0).sum()
Out[4]: 47620

In [5]: 
Exercises : ipython
```

Ex. 2: solving system of linear eqns

- The task:

1. You find three shopping bags with following content:

   bag A: 10 kg apples, 5 kg pears, 1 kg oranges (35.35 EUR)
   bag B: 1 kg apples, 8 kg pears, 1 kg oranges (24.91 EUR)
   bag C: 9 kg apples, 3 kg pears, 5 kg oranges (40.38 EUR)

   Determine the price of apples, oranges and pears.

   (a) Formulate a linear system of equations describing the shopping bags.
   (b) Use **scipy.linalg.solve()** to solve the system of equations.
   (c) Verify the results by using **numpy.dot()**

Create the matrix

sjahnke : ipython

File Edit View Bookmarks Settings Help

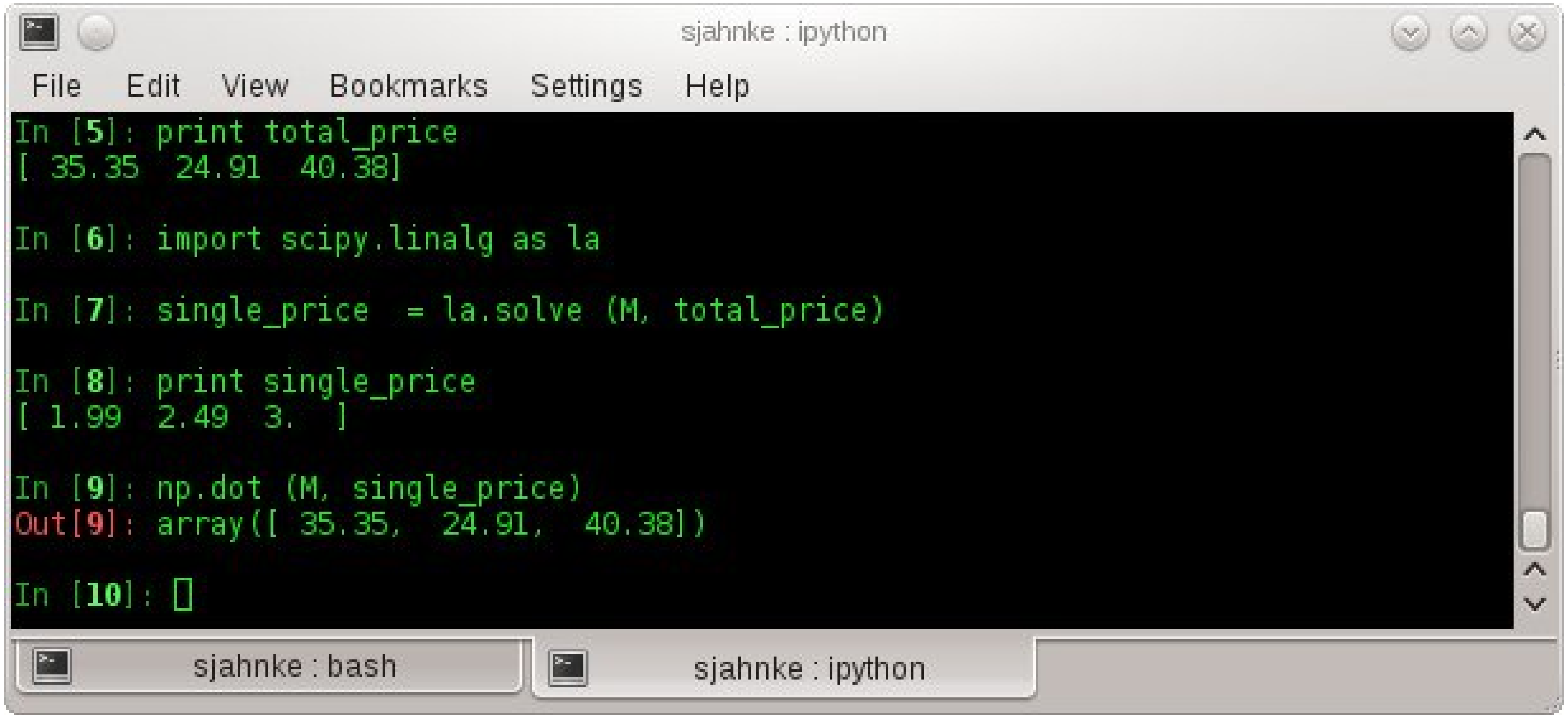
```
sjahnke@tidgituk:~> ipython
Python 2.7.2 (default, Aug 19 2011, 20:41:43) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: M = np.array([ [10, 5, 1], [1,8,1], [9,3,5] ] )

In [3]: print M
[[10  5  1]
 [ 1  8  1]
 [ 9  3  5]]

In [4]: total_price = np.array( [35.35, 24.91, 40.38] )

In [5]: print total_price
[ 35.35  24.91  40.38]

In [6]: 
```

sjahnke : bash | sjahnke : ipython

```
>> import scipy.linalg as la

>> help(la.solve)
```

```
Help on function solve in module scipy.linalg.basic:

solve(a, b, sym_pos=False, lower=False, overwrite_a=False, overwrite_b=False, debug=
False)
    Solve the equation a x = b for x

    Parameters
    ----------
    a : array, shape (M, M)
    b : array, shape (M,) or (M, N)
    sym_pos : boolean
        Assume a is symmetric and positive definite
    lower : boolean
        Use only data contained in the lower triangle of a, if sym_pos is true.
        Default is to use upper triangle.
    overwrite_a : boolean
        Allow overwriting data in a (may enhance performance)
    overwrite_b : boolean
        Allow overwriting data in b (may enhance performance)

    Returns
    -------
    x : array, shape (M,) or (M, N) depending on b
        Solution to the system a x = b

    Raises LinAlgError if a is singular
lines 1-25/25 (END)
```

Solve the equations



Example 3: polynomial fitting (polyfit)

- The task:

1. Generate data by a polynomial function

$$f(x) = -3x^3 + 2x - 8$$

2. Fit a polynomial (**scipy.polyfit()**) and recover the coefficients of the polynomial.

3. Try the same procedure with noisy data.
(use **np.random.randn()** to add noise to the data)

```
>> import scipy as sc

>> help(sc.polyfit)
```

sjahnke : ipython

File Edit View Bookmarks Settings Help

```
Help on function polyfit in module numpy.lib.polynomial:

polyfit(x, y, deg, rcond=None, full=False)
    Least squares polynomial fit.

    Fit a polynomial ``p(x) = p[0] * x**deg + ... + p[deg]`` of degree `deg`
    to points `(x, y)`. Returns a vector of coefficients `p` that minimises
    the squared error.

    Parameters
    ----------
    x : array_like, shape (M,)
        x-coordinates of the M sample points ``(x[i], y[i])``.
    y : array_like, shape (M,) or (M, K)
        y-coordinates of the sample points. Several data sets of sample
        points sharing the same x-coordinates can be fitted at once by
        passing in a 2D-array that contains one dataset per column.
    deg : int
        Degree of the fitting polynomial
    rcond : float, optional
        Relative condition number of the fit. Singular values smaller than this
        relative to the largest singular value will be ignored. The default
        value is len(x)*eps, where eps is the relative precision of the float
        type, about 2e-16 in most cases.
    full : bool, optional
        Switch determining nature of return value. When it is
        False (the default) just the coefficients are returned, when True
        diagnostic information from the singular value decomposition is also
        returned.

    Returns
    -------
    p : ndarray, shape (M,) or (M, K)
        Polynomial coefficients, highest power first.
        If `y` was 2-D, the coefficients for `k`-th data set are in ``p[:,k]``.

    residuals, rank, singular_values, rcond : present only if `full` = True
        Residuals of the least-squares fit, the effective rank of the scaled
        Vandermonde coefficient matrix, its singular values, and the specified
        value of `rcond`. For more details, see `linalg.lstsq`.

lines 1-41
```

sjahnke : ipython

Polynomial fitting

```
sjahnke@tidgituk:~> ipython
Python 2.7.2 (default, Aug 19 2011, 20:41:43) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: import scipy as sc

In [3]: x = np.arange (-4,4, 0.1)

In [4]: y = -3. * x**3 + 2. * x -8

In [5]: sc.polyfit (x, y, 3)
Out[5]:
array([ -3.00000000e+00,   6.16748406e-16,   2.00000000e+00,
        -8.00000000e+00])

In [6]: sc.polyfit (x, y, 5)
Out[6]:
array([  1.98838738e-17,   1.75451017e-16,  -3.00000000e+00,
        -1.32690083e-15,   2.00000000e+00,  -8.00000000e+00])

In [7]: y = y + np.random.randn( len(y) )

In [8]: sc.polyfit (x, y, 3)
Out[8]:
array([ -2.98841207e+00,  -7.61494306e-03,   1.83601609e+00,
        -7.92241359e+00])
```

Example 5: nonlinear fitting

- The task:

1. Define the function $f(x) = e^{-a \cdot x} + b$

2. Generate noisy data from that function with parameters

$$a = 2.0; b = 1.4$$

3. Make a simple plot of the data

```
>> import pylab as pl
>> pl.plot ( xvalues, yvalues )
>> pl.show()
```

4. Use **scipy.optimize.curve_fit()** to estimate a and b from the data and plot the results.

Generate and plot data

sjahnke : ipython

File Edit View Bookmarks Settings Help

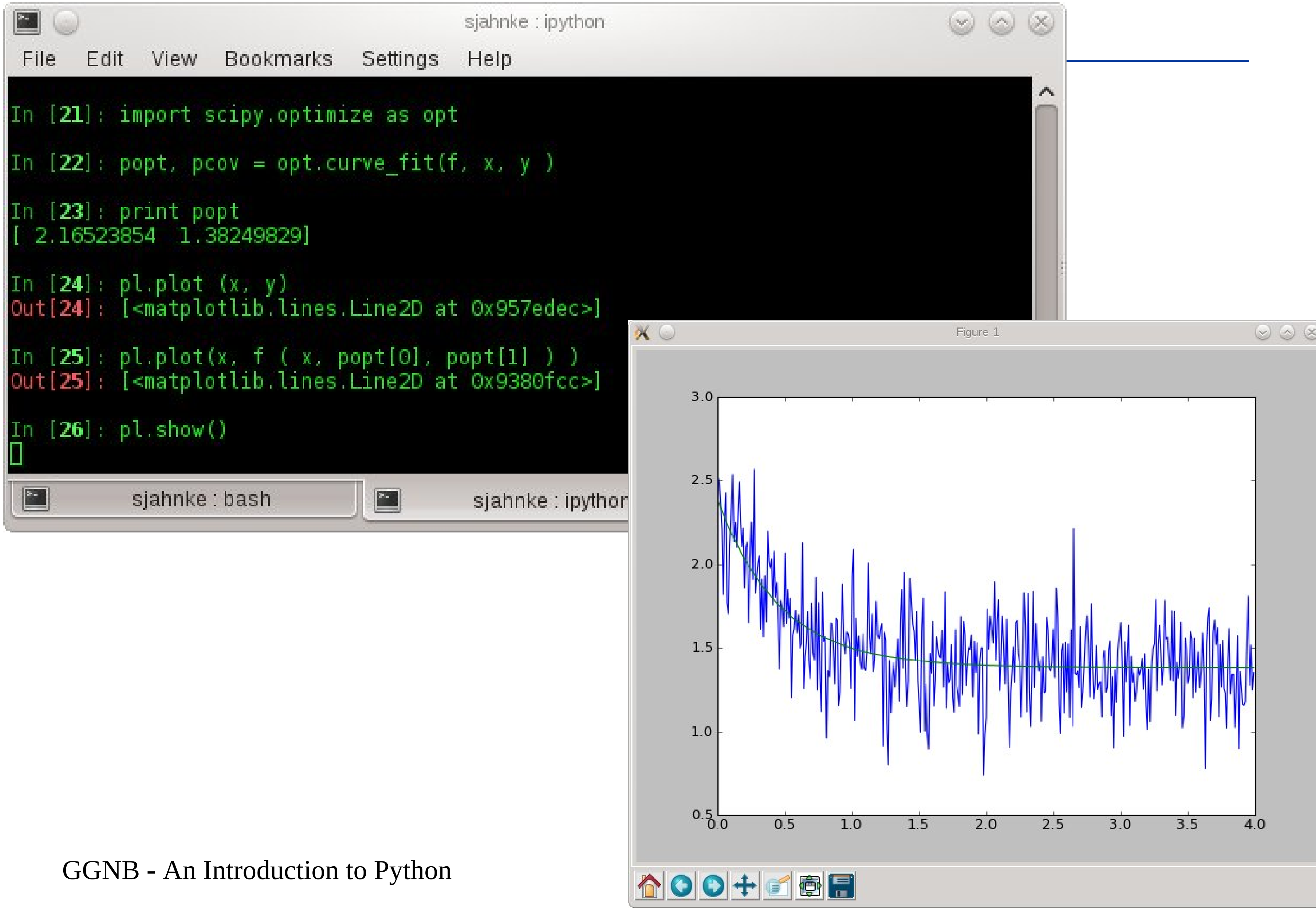
```
sjahnke@tidgituk:~> ipython
Python 2.7.2 (default, Aug 19 2011, 20:41:43) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: f = lambda x, a, b : np.exp (- a * x) + b

In [3]: x = np.arange( 0, 4, 0.01)

In [4]: y = f(x, 2.5, 1.4 )

In [5]: y = y + 0.25 * np.random.randn (len (y))

In [6]: import pylab as pl

In [7]: pl.plot (x,y)
Out[7]: [<matplotlib.lines.Line2D at 0x8f847ec>]

In [8]: pl.show()
```

sjahnke : bash   sjahnke : ipython

```
>> import scipy.optimize as opt

>> help( opt.curve_fit )
```

```
Help on function curve_fit in module scipy.optimize.minpack:

curve_fit(f, xdata, ydata, p0=None, sigma=None, **kw)
    Use non-linear least squares to fit a function, f, to data.

    Assumes ``ydata = f(xdata, *params) + eps``

    Parameters
    ----------
    f : callable
        The model function, f(x, ...).  It must take the independent
        variable as the first argument and the parameters to fit as
        separate remaining arguments.
    xdata : An N-length sequence or an (k,N)-shaped array
        for functions with k predictors.
        The independent variable where the data is measured.
    ydata : N-length sequence
        The dependent data --- nominally f(xdata, ...)
    p0 : None, scalar, or M-length sequence
        Initial guess for the parameters.  If None, then the initial
        values will all be 1 (if the number of parameters for the function
        can be determined using introspection, otherwise a ValueError
        is raised).
    sigma : None or N-length sequence
        If not None, it represents the standard-deviation of ydata.
        This vector, if given, will be used as weights in the
        least-squares problem.

    Returns
    -------
    popt : array
        Optimal values for the parameters so that the sum of the squared error
        of ``f(xdata, *popt) - ydata`` is minimized
    pcov : 2d array
        The estimated covariance of popt.  The diagonals provide the variance
        of the parameter estimate.

lines 1-37/58 73%
```

Nonlinear curve - fitting



Example 6: integration

- The task:

1. Define the function $f(x) = e^{-(x+3)^2}$
2. Vectorize the function by using **np.vectorize()** such that the function also accepts lists of xvalues.
3. Calculate the integral $\int_{-20}^{20} f(x)dx$ by "hand" using Riemann sums.
4. Use **scipy.integrate.quad()** to evaluate the integral.
5. Try to calculate $\int_{1}^{\infty} \frac{1}{x} dx$ and $\int_{1}^{\infty} \frac{1}{x^2} dx$

sjahnke : ipython

File Edit View Bookmarks Settings Help

```
Help on class vectorize in module numpy.lib.function_base:

class vectorize(__builtin__.object)
 |  vectorize(pyfunc, otypes='', doc=None)
 |
 |  Generalized function class.
 |
 |  Define a vectorized function which takes a nested sequence
 |  of objects or numpy arrays as inputs and returns a
 |  numpy array as output. The vectorized function evaluates `pyfunc` over
 |  successive tuples of the input arrays like the python map function,
 |  except it uses the broadcasting rules of numpy.
 |
 |  The data type of the output of `vectorized` is determined by calling
 |  the function with the first element of the input.  This can be avoided
 |  by specifying the `otypes` argument.
 |
 |  Parameters
 |  ----------
 |  pyfunc : callable
 |      A python function or method.
 |  otypes : str or list of dtypes, optional
 |      The output data type. It must be specified as either a string of
 |      typecode characters or a list of data type specifiers. There should
 |      be one data type specifier for each output.
 |  doc : str, optional
 |      The docstring for the function. If None, the docstring will be the
 |      `pyfunc` one.
 |
 |  Examples
 |  --------
 |  >>> def myfunc(a, b):
 |  ...     """Return a-b if a>b, otherwise return a+b"""
 |  ...     if a > b:
 |  ...         return a - b
 |  ...     else:
 |  ...         return a + b
 |
 |  >>> vfunc = np.vectorize(myfunc)
 |  >>> vfunc([1, 2, 3, 4], 2)
 |  array([3, 4, 1, 2])
 |
lines 1-42/76 58%
```

sjahnke : ipython

```
>> import numpy as np

>> help(np.vectorize)
```

Vectorize a function

```
sjahnke@tidgituk:~> ipython
Python 2.7.2 (default, Aug 19 2011, 20:41:43) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: import scipy.integrate as int

In [3]: f = lambda x : np.exp ( - (x+3)**2 )

In [4]: f([0,1,2,3])
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/home/sjahnke/<ipython-input-4-c999ad52e3e4> in <module>()
----> 1 f([0,1,2,3])

/home/sjahnke/<ipython-input-3-c9cc4f40cc2c> in <lambda>(x)
----> 1 f = lambda x : np.exp ( - (x+3)**2 )

TypeError: can only concatenate list (not "int") to list

In [5]: f = np.vectorize(lambda x : np.exp ( - (x+3)**2 ) )

In [6]: f([0,1,2,3])
Out[6]:
array([  1.23409804e-04,   1.12535175e-07,   1.38879439e-11,
         2.31952283e-16])

In [7]:
```

Calculating the Riemann sum

```
In [5]: intf = lambda fr, to, step : f ( np.arange(fr, to, step) ).sum() * step

In [6]: intf ( -20, 20, 1)
Out[6]: 1.7726372048266521

In [7]: intf ( -20, 20, 0.1)
Out[7]: 1.772453850905491

In [8]: intf ( -20, 20, 0.01)
Out[8]: 1.7724538509052388

In [9]: intf ( -20, 20, 0.001)
Out[9]: 1.7724538509033498

In [10]: intf ( -20, 20, 0.0001)
Out[10]: 1.772453850909647

In [11]:
```

```
>> import scipy.integrate as int

>> help(int.quad)
```

```
Help on function quad in module scipy.integrate.quadpack:

quad(func, a, b, args=(), full_output=0, epsabs=1.49e-08, epsrel=1.49e-08, limi
None, weight=None, wvar=None, wopts=None, maxp1=50, limlst=50)
    Compute a definite integral.

    Integrate func from a to b (possibly infinite interval) using a technique
    from the Fortran library QUADPACK.

    If func takes many arguments, it is integrated along the axis corresponding
    to the first argument. Use the keyword argument `args` to pass the other
    arguments.

    Run scipy.integrate.quad_explain() for more information on the
    more esoteric inputs and outputs.

    Parameters
    ----------

    func : function
        A Python function or method to integrate.
    a : float
        Lower limit of integration (use -scipy.integrate.Inf for -infinity).
    b : float
        Upper limit of integration (use scipy.integrate.Inf for +infinity).
    args : tuple, optional
        extra arguments to pass to func
    full_output : int
        Non-zero to return a dictionary of integration information.
        If non-zero, warning messages are also suppressed and the
        message is appended to the output tuple.

    Returns
    -------

    y : float
        The integral of func from a to b.
    abserr : float
        an estimate of the absolute error in the result.

lines 1-39
```

Calculating integrals using scipy.integrate.quad()

sjahnke : ipython

File Edit View Bookmarks Settings Help

```
In [13]: intf ( -20, 20, 0.0001)
impOut[13]: 1.772453850909647

In [14]: import scipy.integrate as int

In [15]: int.quad(f,-20,20)
Out[15]: (1.7724538509055163, 1.0654682030038008e-09)

In [16]: int.quad(lambda x: 1./x , 1. , int.Inf)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/home/sjahnke/<ipython-input-16-4708e3afcf7f> in <module>()
----> 1 int.quad(lambda x: 1./x , 1. , int.Inf)

AttributeError: 'module' object has no attribute 'Inf'

In [17]: int.quad(lambda x: 1./x , 1. , np.Inf)
/usr/lib/python2.7/site-packages/scipy/integrate/quadpack.py:288: UserWarning: The maximum n
umber of subdivisions (50) has been achieved.
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
  warnings.warn(msg)
Out[17]: (40.996012819169465, 8.156214940493738)

In [18]: int.quad(lambda x: 1./(x**2) , 1. , np.Inf)
Out[18]: (1.0000000000000002, 1.1102230246251567e-14)

In [19]: 
```

sjahnke : ipython

Example 7: Lotka-Voltera equation

- The task:

1. Consider the Lotka-Voltera equation (predator-prey-eqn)

$$\frac{dN_1}{dt} = N_1\left(\epsilon_1 - \gamma_1 N_2\right) \qquad \frac{dN_2}{dt} = -N_2\left(\epsilon_2 - \gamma_2 N_2\right)$$

   Number of preys $N_1(t)$ and predators $N_2(t)$
   $\epsilon_1, \epsilon_2, \gamma_1, \gamma_2$ parameters representing the growth and interaction

2. Find the fixed points of the system for the following parameters

$$\epsilon_1 = 1.0, \epsilon_2 = 1.5, \gamma_1 = 0.1, \gamma_2 = 0.075$$

3. Solve the system of equations for the following initial conditions:

$$N_1(0) = 10, N_2(0) = 5$$

Example 7: Find the fixed points

$$\frac{dN_1}{dt} = N_1\left(\epsilon_1 - \gamma_1 N_2\right) \qquad \frac{dN_2}{dt} = -N_2\left(\epsilon_2 - \gamma_2 N_2\right)$$

$$\epsilon_1 = 1.0, \epsilon_2 = 1.5, \gamma_1 = 0.1, \gamma_2 = 0.075$$

1. Define a function that returns the growing rates.
Because we will use **scipy.integrate.odeint()** in the next step, choose the parameters of the function accordingly.

$$f = f(N, t, \ldots)$$

2. Use **scipy.optimize.fsolve()** to find the fixed points of the system.

```
>> import scipy.integrate as int

>> help(int.odeint)
```

```
Help on function odeint in module scipy.integrate.odepack:

odeint(func, y0, t, args=(), Dfun=None, col_deriv=0, full_output=0, ml=None, mu=None, r
tcrit=None, h0=0.0, hmax=0.0, hmin=0.0, ixpr=0, mxstep=0, mxhnil=0, mxordn=12, mxords=5
    Integrate a system of ordinary differential equations.

    Solve a system of ordinary differential equations using lsoda from the
    FORTRAN library odepack.

    Solves the initial value problem for stiff or non-stiff systems
    of first order ode-s::

        dy/dt = func(y,t0,...)

    where y can be a vector.

    Parameters
    ----------
    func : callable(y, t0, ...)
        Computes the derivative of y at t0.
    y0 : array
        Initial condition on y (can be a vector).
    t : array
        A sequence of time points for which to solve for y.  The initial
        value point should be the first element of this sequence.
    args : tuple
        Extra arguments to pass to function.
    Dfun : callable(y, t0, ...)
        Gradient (Jacobian) of func.
    col_deriv : boolean
        True if Dfun defines derivatives down columns (faster),
        otherwise Dfun should define derivatives across rows.
    full_output : boolean
        True if to return a dictionary of optional outputs as the second output
    printmessg : boolean
        Whether to print the convergence message

    Returns
    -------
    y : array, shape (len(t), len(y0))
        Array containing the value of y for each desired time in t,
        with the initial value y0 in the first row.

    infodict : dict, only returned if full_output == True
        Dictionary containing additional output information
lines 1-44
```

File Edit View Bookmarks Settings Help

sjahnke : ipython

sjahnke : bash | sjahnke : ipython

```
>> import scipy.optimize as opt

>> help(opt.fsolve)
```

sjahnke : ipython

File Edit View Bookmarks Settings Help

```
Help on function fsolve in module scipy.optimize.minpack:

fsolve(func, x0, args=(), fprime=None, full_output=0, col_deriv=0, xtol=1.49012e-08, maxfev=0, band=None, ep
sfcn=0.0, factor=100, diag=None)
    Find the roots of a function.

    Return the roots of the (non-linear) equations defined by
    ``func(x) = 0`` given a starting estimate.

    Parameters
    ----------
    func : callable f(x, *args)
        A function that takes at least one (possibly vector) argument.
    x0 : ndarray
        The starting estimate for the roots of ``func(x) = 0``.
    args : tuple
        Any extra arguments to `func`.
    fprime : callable(x)
        A function to compute the Jacobian of `func` with derivatives
        across the rows. By default, the Jacobian will be estimated.
    full_output : bool
        If True, return optional outputs.
    col_deriv : bool
        Specify whether the Jacobian function computes derivatives down
        the columns (faster, because there is no transpose operation).

    Returns
    -------
    x : ndarray
        The solution (or the result of the last iteration for
        an unsuccessful call).
    infodict : dict
        A dictionary of optional outputs with the keys::

lines 1-33
```

sjahnke : bash   sjahnke : ipython

Finding the fixed points

```
sjahnke@tidgituk:~> ipython
Python 2.7.2 (default, Aug 19 2011, 20:41:43) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: import scipy.optimize as opt

In [3]: def dN_dt ( N, t0, eps1, gamma1, eps2, gamma2 ):
   ...:     dN1_dt = N[0] * ( eps1 - gamma1 * N[1] )
   ...:     dN2_dt = - N[1] * ( eps2 - gamma2 * N[0] )
   ...:     return np.array( [ dN1_dt, dN2_dt ] )
   ...:

In [4]: dN_dt( [10., 5.], 0, 1.0, 0.1, 1.5, 0.075 )
Out[4]: array([ 5.  , -3.75])

In [5]: opt.fsolve( dN_dt, x0= [0,0] , args= (0, 1.0, 0.1, 1.5, 0.075) )
Out[5]: array([ 0.,  0.])

In [6]: opt.fsolve( dN_dt, x0= [10,10] , args= (0, 1.0, 0.1, 1.5, 0.075) )
Out[6]: array([ 20.,  10.])

In [7]: dN_dt( [0., 0.], 0, 1.0, 0.1, 1.5, 0.075 )
Out[7]: array([ 0., -0.])

In [8]: dN_dt( [20., 10.], 0, 1.0, 0.1, 1.5, 0.075 )
Out[8]: array([ 0., -0.])

In [9]:
```

Example 7: Solve the system

$$\frac{dN_1}{dt} = N_1 \left(\epsilon_1 - \gamma_1 N_2\right) \qquad \frac{dN_2}{dt} = -N_2 \left(\epsilon_2 - \gamma_2 N_2\right)$$

$$\epsilon_1 = 1.0, \epsilon_2 = 1.5, \gamma_1 = 0.1, \gamma_2 = 0.075$$

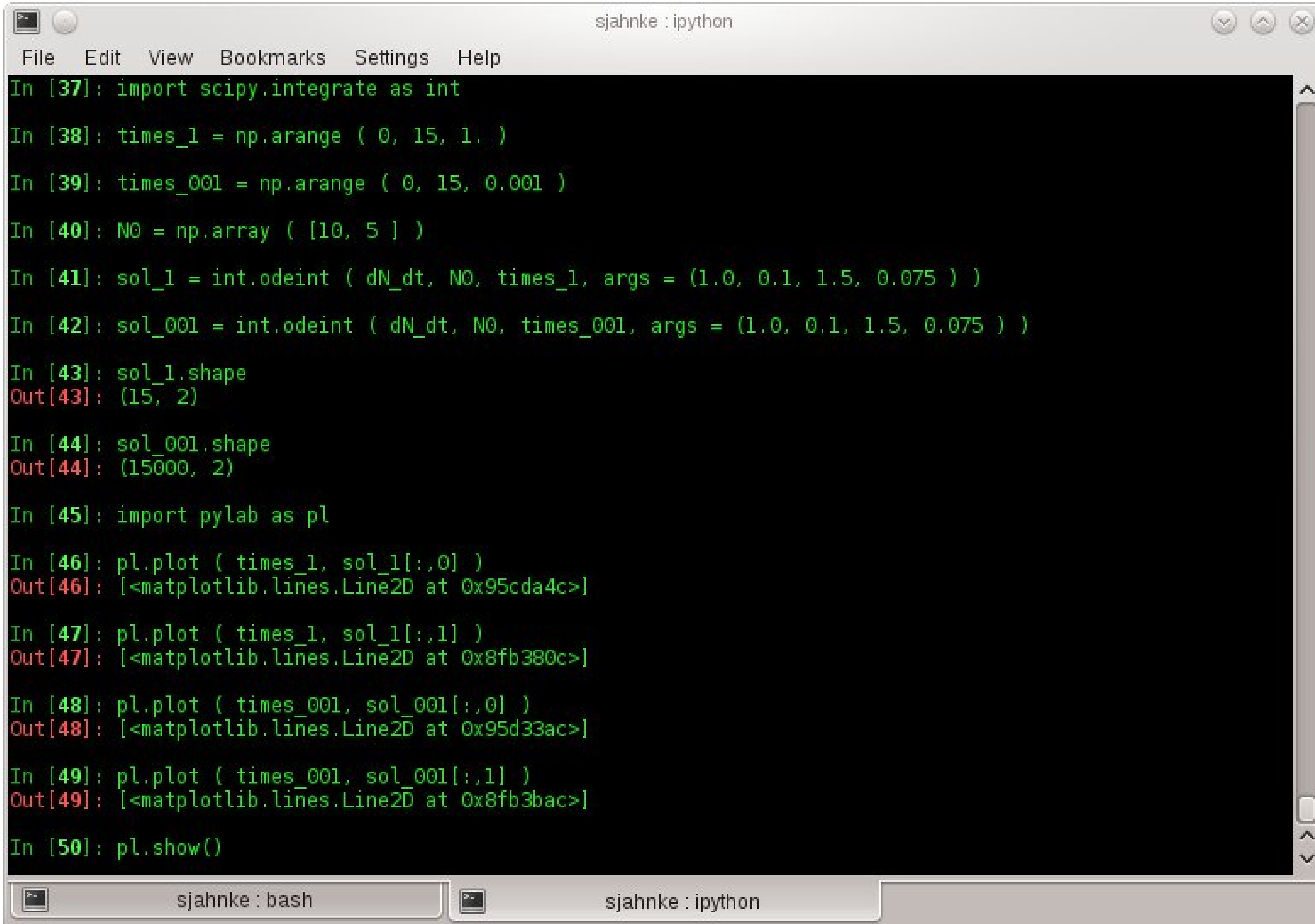
$$N_1(0) = 10, N_2(0) = 5$$

1. Define a vector containing the time steps for the integration and one for the initial conditions.

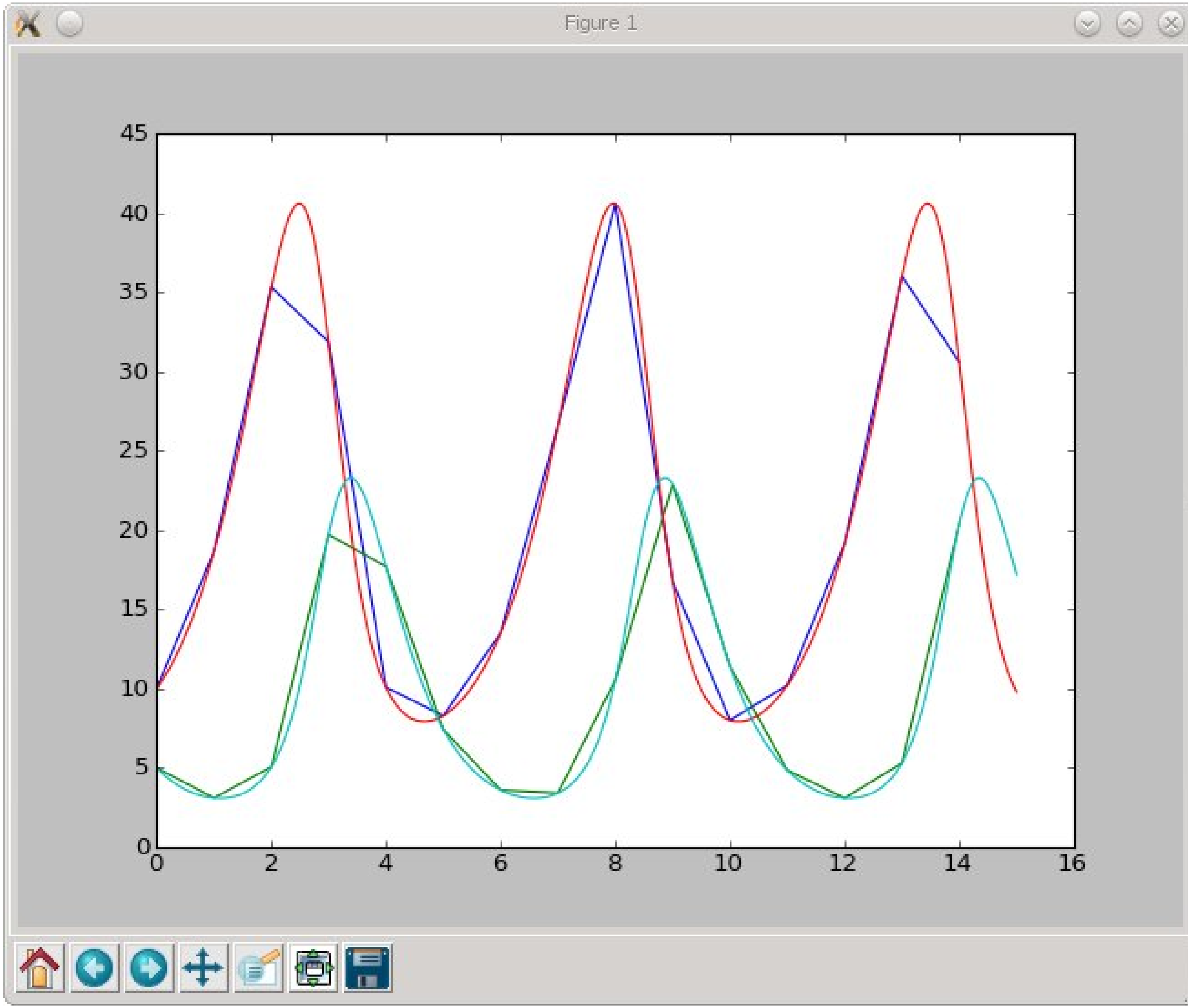
2. Solve the system, using **scipy.integrate.odeint()**

3. Plot the results for two different time step sizes.

Integration the ODE



Integration the ODE



Finally I/O functions

- read & save text files

```
>> np.savetxt ( filename, variable) #format can be specified
>> data = np.loadtxt ( filename ) #adds an .npy to filename
```

- read & save binary data

```
>> np.save ( file, variable ) #adds an .npy to filename
>> data = np.load ( file.npy ) #loading file
```

- reading matlab files

```
>> import scipy.io as io
>> matdata = io.loadmat (filename.mat) #returns a dictonary
>> print matdata[ ‘data’ ] #if the matlabfile contains data struct
```

For help take a look at the reference pages:

scipy:

- http://docs.scipy.org/doc/scipy/reference/

numpy:

- http://docs.scipy.org/doc/numpy/reference/

Have fun in the exercises!