intro_Scipy

November 25, 2015

```
In [1]: # Just to know last time this was run:
        import time
        print time.ctime()
```

```
Mon Sep 14 16:59:08 2015
```

1 E Introduction to Scipy

This is part of the Python lecture given by Christophe Morisset at IA-UNAM. More informations at: http://python-astro.blogspot.mx/

Scipy is a library with a lot of foncionalities, we will not cover everything here, but rather point to some of them with examples. Some useful links about scipy:

- https://scipy-lectures.github.io/intro/scipy.html
- http://docs.scipy.org/doc/scipy/reference/tutorial/

```
In [2]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [3]: import scipy # This imports a lot of numpy stuff, but not the important modules
```

1.0.1 Some usefull methods

```
In [4]: from scipy.stats import nanmean
```

```
In [6]: a = np.array([-2, -1, 1., 2, 3])
        b = np.log10(a)
        mask = np.isfinite(b)
        print a
        print b
        print mask
        print a.mean()
        print b.mean()
        print b[mask].mean()
        print nanmean(b)
```

```
[-2. -1.  1.  2.  3.]
[       nan        nan  0.         0.30103    0.47712125]
[False False  True  True  True]
0.6
nan
0.259383750128
0.259383750128
```

```
In [7]: from scipy.special import gamma
        print gamma(10.3)
```

```
716430.689062
```

```
In [8]: from scipy import constants as cst
        print cst.astronomical_unit # A lot of constants
        from scipy.constants import codata # a lot more, with units. From NIST
        print codata.value('proton mass'), codata.unit('proton mass')
```

```
1.49597870691e+11
1.672621777e-27 kg
```

1.0.2 Integrations

```
In [9]: from scipy.integrate import trapz, cumtrapz, simps
        #help(scipy.integrate) # a big one...
        print '-------------------------------------------------------------------------------'
        help(trapz)
        print '-------------------------------------------------------------------------------'
        help(cumtrapz)
        print '-------------------------------------------------------------------------------'
        help(simps)
```

```
-------------------------------------------------------------------------------
Help on function trapz in module numpy.lib.function_base:

trapz(y, x=None, dx=1.0, axis=-1)
    Integrate along the given axis using the composite trapezoidal rule.

    Integrate `y` (`x`) along given axis.

    Parameters
    ----------
    y : array_like
        Input array to integrate.
    x : array_like, optional
        If `x` is None, then spacing between all `y` elements is `dx`.
    dx : scalar, optional
        If `x` is None, spacing given by `dx` is assumed. Default is 1.
    axis : int, optional
        Specify the axis.

    Returns
    -------
    trapz : float
        Definite integral as approximated by trapezoidal rule.

    See Also
    --------
    sum, cumsum

    Notes
    -----
    Image [2]_ illustrates trapezoidal rule -- y-axis locations of points
```

```
    will be taken from `y` array, by default x-axis distances between
    points will be 1.0, alternatively they can be provided with `x` array
    or with `dx` scalar.  Return value will be equal to combined area under
    the red lines.


    References
    ----------
    .. [1] Wikipedia page: http://en.wikipedia.org/wiki/Trapezoidal_rule

    .. [2] Illustration image:
           http://en.wikipedia.org/wiki/File:Composite_trapezoidal_rule_illustration.png

    Examples
    --------
    >>> np.trapz([1,2,3])
    4.0
    >>> np.trapz([1,2,3], x=[4,6,8])
    8.0
    >>> np.trapz([1,2,3], dx=2)
    8.0
    >>> a = np.arange(6).reshape(2, 3)
    >>> a
    array([[0, 1, 2],
           [3, 4, 5]])
    >>> np.trapz(a, axis=0)
    array([ 1.5,  2.5,  3.5])
    >>> np.trapz(a, axis=1)
    array([ 2.,  8.])

----------------------------------------------------------------------------
Help on function cumtrapz in module scipy.integrate.quadrature:

cumtrapz(y, x=None, dx=1.0, axis=-1, initial=None)
    Cumulatively integrate y(x) using the composite trapezoidal rule.

    Parameters
    ----------
    y : array_like
        Values to integrate.
    x : array_like, optional
        The coordinate to integrate along.  If None (default), use spacing `dx`
        between consecutive elements in `y`.
    dx : int, optional
        Spacing between elements of `y`.  Only used if `x` is None.
    axis : int, optional
        Specifies the axis to cumulate.  Default is -1 (last axis).
    initial : scalar, optional
        If given, uses this value as the first value in the returned result.
        Typically this value should be 0.  Default is None, which means no
        value at ``x[0]`` is returned and `res` has one element less than `y`
        along the axis of integration.

    Returns
```

```
    -------
    res : ndarray
        The result of cumulative integration of `y` along `axis`.
        If `initial` is None, the shape is such that the axis of integration
        has one less value than `y`.  If `initial` is given, the shape is equal
        to that of `y`.

    See Also
    --------
    numpy.cumsum, numpy.cumprod
    quad: adaptive quadrature using QUADPACK
    romberg: adaptive Romberg quadrature
    quadrature: adaptive Gaussian quadrature
    fixed_quad: fixed-order Gaussian quadrature
    dblquad: double integrals
    tplquad: triple integrals
    romb: integrators for sampled data
    ode: ODE integrators
    odeint: ODE integrators

    Examples
    --------
    >>> from scipy import integrate
    >>> import matplotlib.pyplot as plt

    >>> x = np.linspace(-2, 2, num=20)
    >>> y = x
    >>> y_int = integrate.cumtrapz(y, x, initial=0)
    >>> plt.plot(x, y_int, 'ro', x, y[0] + 0.5 * x**2, 'b-')
    >>> plt.show()

----------------------------------------------------------------------------
Help on function simps in module scipy.integrate.quadrature:

simps(y, x=None, dx=1, axis=-1, even='avg')
    Integrate y(x) using samples along the given axis and the composite
    Simpson's rule.  If x is None, spacing of dx is assumed.

    If there are an even number of samples, N, then there are an odd
    number of intervals (N-1), but Simpson's rule requires an even number
    of intervals.  The parameter 'even' controls how this is handled.

    Parameters
    ----------
    y : array_like
        Array to be integrated.
    x : array_like, optional
        If given, the points at which `y` is sampled.
    dx : int, optional
        Spacing of integration points along axis of `y`. Only used when
        `x` is None. Default is 1.
    axis : int, optional
        Axis along which to integrate. Default is the last axis.
    even : {'avg', 'first', 'str'}, optional
```

```
        'avg' : Average two results:1) use the first N-2 intervals with
                  a trapezoidal rule on the last interval and 2) use the last
                  N-2 intervals with a trapezoidal rule on the first interval.

        'first' : Use Simpson's rule for the first N-2 intervals with
                a trapezoidal rule on the last interval.

        'last' : Use Simpson's rule for the last N-2 intervals with a
               trapezoidal rule on the first interval.

    See Also
    --------
    quad: adaptive quadrature using QUADPACK
    romberg: adaptive Romberg quadrature
    quadrature: adaptive Gaussian quadrature
    fixed_quad: fixed-order Gaussian quadrature
    dblquad: double integrals
    tplquad: triple integrals
    romb: integrators for sampled data
    cumtrapz: cumulative integration for sampled data
    ode: ODE integrators
    odeint: ODE integrators

    Notes
    -----
    For an odd number of samples that are equally spaced the result is
    exact if the function is a polynomial of order 3 or less.  If
    the samples are not equally spaced, then the result is exact only
    if the function is a polynomial of order 2 or less.
```

```
In [10]: dir(scipy.integrate)
```

```
Out[10]: ['IntegrationWarning',
          'Tester',
          '__all__',
          '__builtins__',
          '__doc__',
          '__file__',
          '__name__',
          '__package__',
          '__path__',
          '_dop',
          '_ode',
          '_odepack',
          '_quadpack',
          'absolute_import',
          'complex_ode',
          'cumtrapz',
          'dblquad',
          'division',
          'fixed_quad',
          'lsoda',
          'newton_cotes',
          'nquad',
          'ode',
```

```
 'odeint',
 'odepack',
 'print_function',
 'quad',
 'quad_explain',
 'quadpack',
 'quadrature',
 'romb',
 'romberg',
 's',
 'simps',
 'test',
 'tplquad',
 'trapz',
 'vode']
```

In [11]:
```
# Defining x and y
x = np.linspace(0, 10, 100)
y = np.sin(x)
# Compare the integrales using two methods
print trapz(y, x)
print simps(y, x)
```

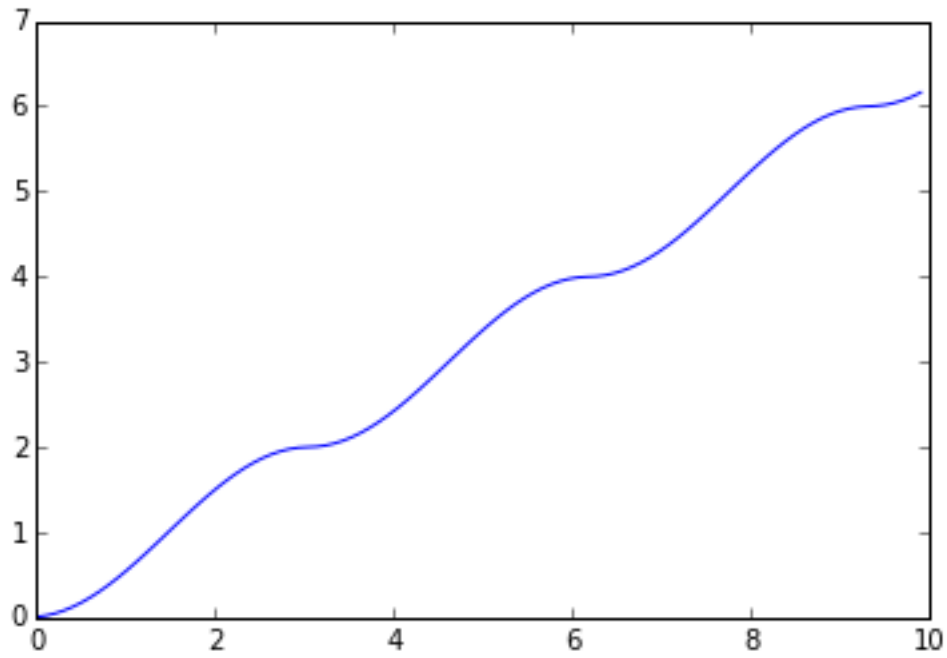
```
1.83750758633
1.83909194697
```

In [12]:
```
# Cumulative integrale
print cumtrapz(np.abs(y), x)
```

```
[  5.09284951e-03   2.03194796e-02   4.55246645e-02   8.04514533e-02
   1.24743789e-01   1.77950140e-01   2.39528101e-01   3.08849923e-01
   3.85208914e-01   4.67826642e-01   5.55860873e-01   6.48414152e-01
   7.44542958e-01   8.43267317e-01   9.43580798e-01   1.04446077e+00
   1.14487882e+00   1.24381125e+00   1.34024951e+00   1.43321048e+00
   1.52174647e+00   1.60495491e+00   1.68198755e+00   1.75205909e+00
   1.81445519e+00   1.86853977e+00   1.91376146e+00   1.94965927e+00
   1.97586722e+00   1.99211816e+00   1.99824642e+00   2.00334164e+00
   2.01754235e+00   2.04174210e+00   2.07569419e+00   2.11905248e+00
   2.17137499e+00   2.23212830e+00   2.30069308e+00   2.37637035e+00
   2.45838864e+00   2.54591181e+00   2.63804762e+00   2.73385681e+00
   2.83236266e+00   2.93256097e+00   3.03343028e+00   3.13394229e+00
   3.23307235e+00   3.32980988e+00   3.42316870e+00   3.51219709e+00
   3.59598746e+00   3.67368561e+00   3.74449946e+00   3.80770711e+00
   3.86266420e+00   3.90881048e+00   3.94567550e+00   3.97288346e+00
   3.99015699e+00   3.99732000e+00   4.00241706e+00   4.01559036e+00
   4.03878211e+00   4.07175590e+00   4.11417558e+00   4.16560871e+00
   4.22553095e+00   4.29333144e+00   4.36831900e+00   4.44972917e+00
   4.53673204e+00   4.62844065e+00   4.72392010e+00   4.82219704e+00
   4.92226959e+00   5.02311758e+00   5.12371293e+00   5.22303013e+00
   5.32005670e+00   5.41380353e+00   5.50331491e+00   5.58767835e+00
   5.66603380e+00   5.73758248e+00   5.80159500e+00   5.85741879e+00
   5.90448477e+00   5.94231312e+00   5.97051821e+00   5.98881251e+00
   5.99700951e+00   6.00210788e+00   6.01425236e+00   6.03643367e+00
   6.06842568e+00   6.10990226e+00   6.16044057e+00]
```

In [13]:
```
# Cumulative integral
print len(x), len(cumtrapz(np.abs(y), x))
f, ax = plt.subplots()
ax.plot(x[0:-1], cumtrapz(np.abs(y), x));
```

```
100 99
```



In [16]:
```
from scipy.integrate import quad # To compute a definite integral
from scipy.special import jv # Bessel function
#help(quad)
print quad(lambda x: jv(2.5, x), 0, 10) # Integrate the Bessel function of order 2.5 between 0
```

```
(0.8209075326034347, 1.1793289815399173e-08)
```

We now want to evaluate:

$$\int_0^1 1 + 2x + 3x^2 dx$$

In [18]:
```
# We want here integrate a user-defined function (here polynome) between 0 and 1
def f(x, a, b, c):
    """ Returning a 2nd order polynome """
    return a + b * x + c * x**2
%timeit I = quad(f, 0, 1, args=(1,2,3)) # args will send 1, 2, 3 to f
print I
Integ = I[0]
print Integ
```

```
10000 loops, best of 3: 22.6 µs per loop
(3.0, 3.3306690738754696e-14)
3.0
```

1.0.3 Interpolations

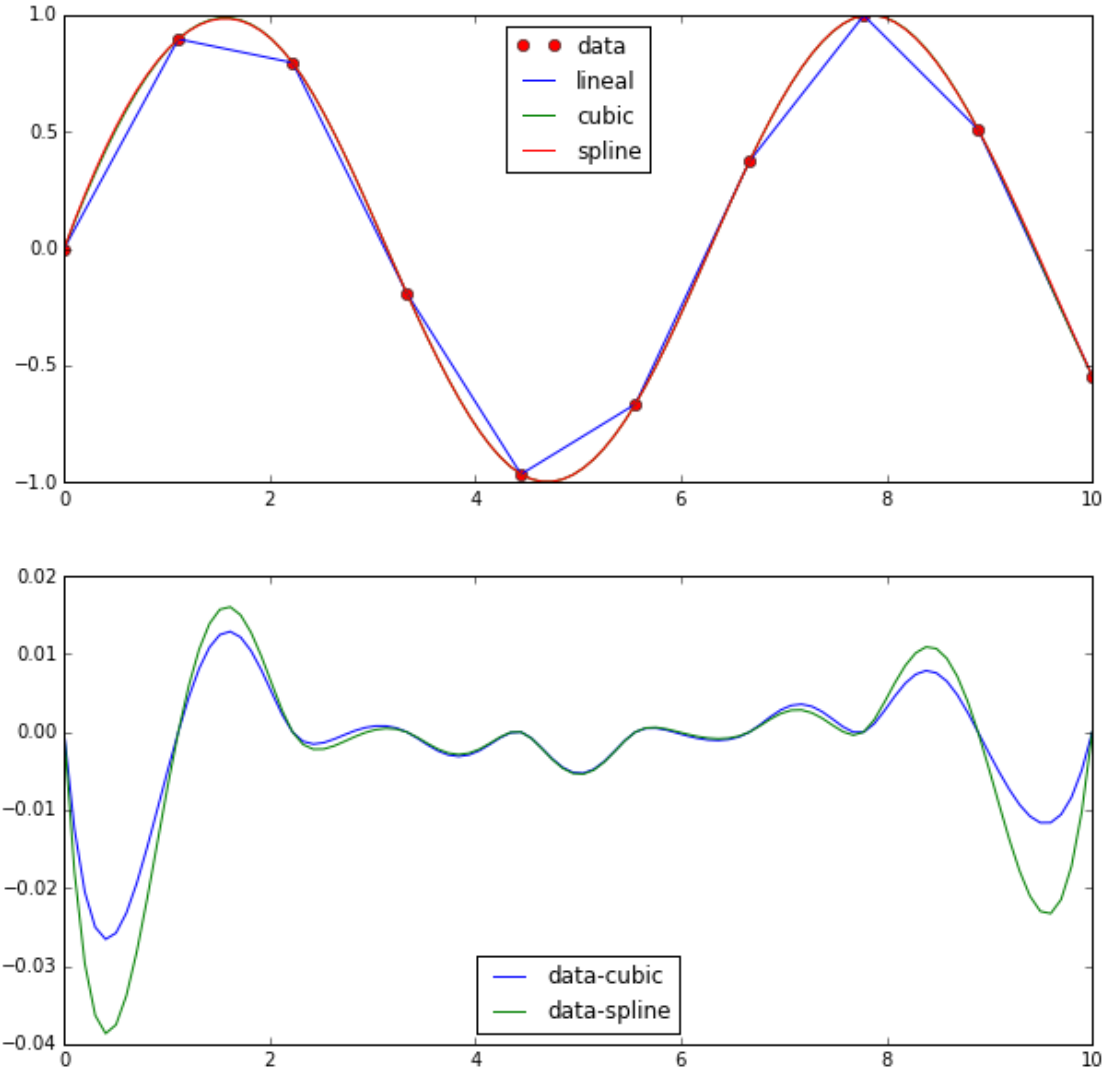
```
In [19]: from scipy.interpolate import interp1d, interp2d, splrep, splev, griddata

In [21]: #help(scipy.interpolate) # a huge one...
         help(interp1d)

Help on class interp1d in module scipy.interpolate.interpolate:

class interp1d(scipy.interpolate.polyint._Interpolator1D)
 |  interp1d(x, y, kind='linear', axis=-1, copy=True, bounds_error=True,
 |           fill_value=np.nan, assume_sorted=False)
 |
 |  Interpolate a 1-D function.
 |
 |  `x` and `y` are arrays of values used to approximate some function f:
 |  ``y = f(x)``.  This class returns a function whose call method uses
 |  interpolation to find the value of new points.
 |
 |  Parameters
 |  ----------
 |  x : (N,) array_like
 |      A 1-D array of real values.
 |  y : (...,N,...) array_like
 |      A N-D array of real values. The length of `y` along the interpolation
 |      axis must be equal to the length of `x`.
 |  kind : str or int, optional
 |      Specifies the kind of interpolation as a string
 |      ('linear', 'nearest', 'zero', 'slinear', 'quadratic, 'cubic'
 |      where 'slinear', 'quadratic' and 'cubic' refer to a spline
 |      interpolation of first, second or third order) or as an integer
 |      specifying the order of the spline interpolator to use.
 |      Default is 'linear'.
 |  axis : int, optional
 |      Specifies the axis of `y` along which to interpolate.
 |      Interpolation defaults to the last axis of `y`.
 |  copy : bool, optional
 |      If True, the class makes internal copies of x and y.
 |      If False, references to `x` and `y` are used. The default is to copy.
 |  bounds_error : bool, optional
 |      If True, a ValueError is raised any time interpolation is attempted on
 |      a value outside of the range of x (where extrapolation is
 |      necessary). If False, out of bounds values are assigned `fill_value`.
 |      By default, an error is raised.
 |  fill_value : float, optional
 |      If provided, then this value will be used to fill in for requested
 |      points outside of the data range. If not provided, then the default
 |      is NaN.
 |  assume_sorted : bool, optional
 |      If False, values of `x` can be in any order and they are sorted first.
 |      If True, `x` has to be an array of monotonically increasing values.
 |
 |  See Also
 |  --------
 |  UnivariateSpline : A more recent wrapper of the FITPACK routines.
```

```
|  splrep, splev
|      Spline interpolation based on FITPACK.
|  interp2d
|
|  Examples
|  --------
|  >>> from scipy import interpolate
|  >>> x = np.arange(0, 10)
|  >>> y = np.exp(-x/3.0)
|  >>> f = interpolate.interp1d(x, y)
|
|  >>> xnew = np.arange(0,9, 0.1)
|  >>> ynew = f(xnew)   # use interpolation function returned by `interp1d`
|  >>> plt.plot(x, y, 'o', xnew, ynew, '-')
|  >>> plt.show()
|
|  Method resolution order:
|      interp1d
|      scipy.interpolate.polyint._Interpolator1D
|      __builtin__.object
|
|  Methods defined here:
|
|  __init__(self, x, y, kind='linear', axis=-1, copy=True, bounds_error=True, fill_value=nan, assume_sor
|      Initialize a 1D linear interpolation class.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from scipy.interpolate.polyint._Interpolator1D:
|
|  __call__(self, x)
|      Evaluate the interpolant
|
|      Parameters
|      ----------
|      x : array-like
|          Points to evaluate the interpolant at.
|
|      Returns
|      -------
|      y : array-like
|          Interpolated values. Shape is determined by replacing
|          the interpolation axis in the original array with the shape of x.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from scipy.interpolate.polyint._Interpolator1D:
```

```
|
|  dtype
```

In [26]:
```
x = np.linspace(0, 10, 10)
y = np.sin(x)
f = interp1d(x, y) # this creates a function that can be call at any interpolate point
f2 = interp1d(x, y, kind='cubic') # The same but using cubic interpolation
tck = splrep(x, y, s=0) # This initiate the spline interpolating function, finding the B-splin
# tck is a sequence of length 3 returned by 'splrep' or 'splprep' containing the knots, coeffi
f3 = lambda x: splev(x, tck) # Evaluate the B-spline or its derivatives.
```

In [27]:
```
# Defining the high resolution mesh
xfine = np.linspace(0, 10, 100)
yfine = np.sin(xfine)
# Plot to compare the results
fig, (ax1, ax2) = plt.subplots(2, figsize=(10,10))

ax1.plot(x, y, 'or', label='data')
ax1.plot(xfine, f(xfine), label='lineal')
ax1.plot(xfine, f2(xfine), label='cubic')
ax1.plot(xfine, f3(xfine), label='spline')
ax1.legend(loc=9)

ax2.plot(xfine, (yfine-f2(xfine)), label='data-cubic')
ax2.plot(xfine, (yfine-f3(xfine)), label='data-spline')
ax2.legend(loc=8);
```




```
In [28]: x0 = 3.5
         print np.sin(x0), f(x0), f2(x0), f3(x0)
```

```
-0.35078322769 -0.306630335983 -0.349437256954 -0.349597252402
```

2D interpolation

```
In [29]: # Defining a 2D-function
         def func(x, y):
             return x * (1+x) * np.cos(4*np.pi*x) * np.sin(4*np.pi*y**2)**2
```

```
In [30]: # Initializing a 2D coordinate grid. Note the use of j to specify that the end point is includ
         grid_x, grid_y = np.mgrid[0:1:100j, 0:1:200j]
```

```
In [31]: print grid_x
         print grid_y
```

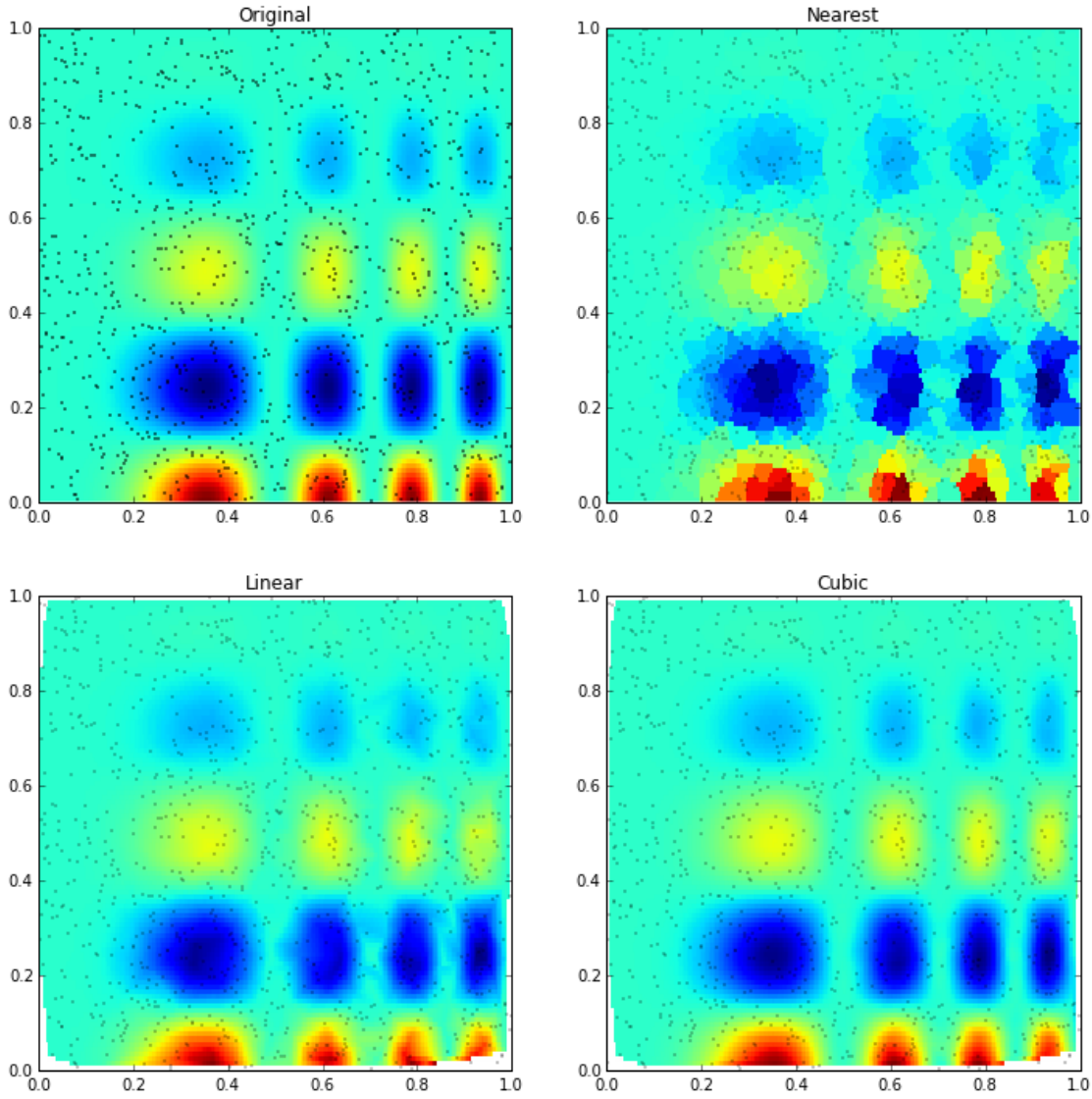
```
[[ 0.          0.          0.         ...,  0.          0.          0.        ]
 [ 0.01010101  0.01010101  0.01010101 ...,  0.01010101  0.01010101
   0.01010101]
 [ 0.02020202  0.02020202  0.02020202 ...,  0.02020202  0.02020202
   0.02020202]
 ...,
 [ 0.97979798  0.97979798  0.97979798 ...,  0.97979798  0.97979798
   0.97979798]
 [ 0.98989899  0.98989899  0.98989899 ...,  0.98989899  0.98989899
   0.98989899]
 [ 1.          1.          1.         ...,  1.          1.          1.        ]]
[[ 0.          0.00502513  0.01005025 ...,  0.98994975  0.99497487  1.        ]
 [ 0.          0.00502513  0.01005025 ...,  0.98994975  0.99497487  1.        ]
 [ 0.          0.00502513  0.01005025 ...,  0.98994975  0.99497487  1.        ]
 ...,
 [ 0.          0.00502513  0.01005025 ...,  0.98994975  0.99497487  1.        ]
 [ 0.          0.00502513  0.01005025 ...,  0.98994975  0.99497487  1.        ]
 [ 0.          0.00502513  0.01005025 ...,  0.98994975  0.99497487  1.        ]]
```

In [32]:
```
# Generating 1000 x 2 points randomly
points = np.random.rand(1000, 2)
values = func(points[:,0], points[:,1])
```

In [37]:
```
# griddata is the 2D-interpolating method. We want to obtain values on (grid_x, grid_y) points
# using "points" and "values".
%timeit grid_z0 = griddata(points, values, (grid_x, grid_y), method='nearest')
%timeit grid_z1 = griddata(points, values, (grid_x, grid_y), method='linear')
%timeit grid_z2 = griddata(points, values, (grid_x, grid_y), method='cubic')
```

```
10 loops, best of 3: 53.8 ms per loop
100 loops, best of 3: 13.2 ms per loop
10 loops, best of 3: 29.2 ms per loop
```

In [35]:
```
# 4 subplots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 12))

ax1.imshow(func(grid_x, grid_y), extent=(0,1,0,1), interpolation='none',
           origin='upper')
ax1.plot(points[:,0], points[:,1], 'ko', ms=1)
ax1.set_title('Original')

ax2.imshow(grid_z0, extent=(0,1,0,1), interpolation='none',
           origin='upper')
ax2.plot(points[:,0], points[:,1], 'k.', ms=1)
ax2.set_title('Nearest')

ax3.imshow(grid_z1, extent=(0,1,0,1), interpolation='none',
           origin='upper')
ax3.plot(points[:,0], points[:,1], 'k.', ms=1)
ax3.set_title('Linear')

ax4.imshow(grid_z2, extent=(0,1,0,1), interpolation='none',
           origin='upper')
ax4.plot(points[:,0], points[:,1], 'k.', ms=1)
ax4.set_title('Cubic');
```



1.0.4 Linear algebra

Scipy is able to deal with matrices, solving linear equations, solving linear least-squares problems and pseudo-inverses, finding eigenvalues and eigenvectors, and more, see here: http://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html

1.0.5 Data fit

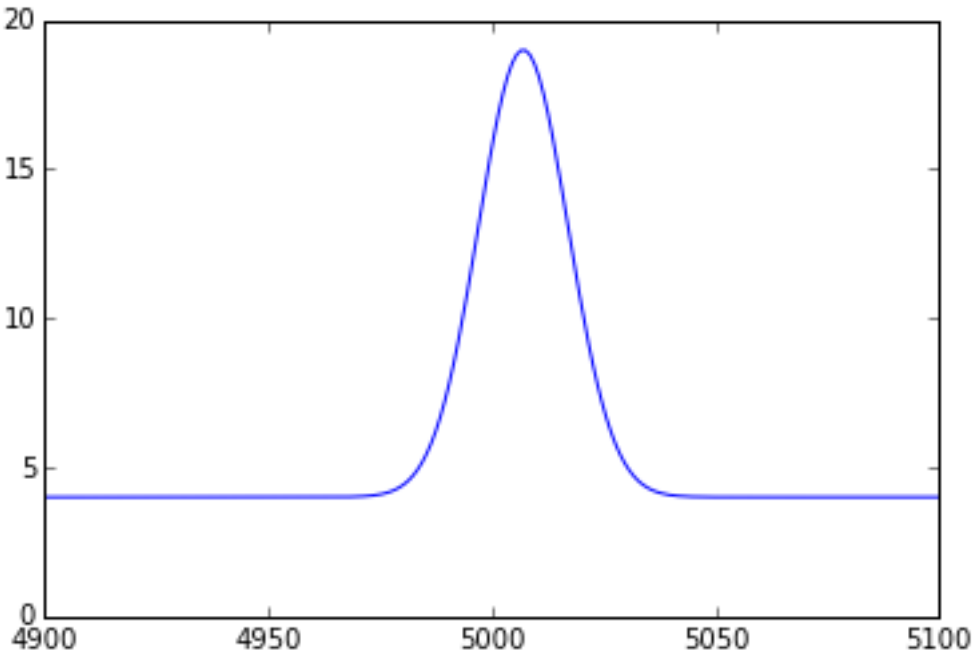
```
In [38]: from scipy.optimize import curve_fit # this is used to adjust a set of data

In [39]: help(curve_fit)

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

curve_fit(f, xdata, ydata, p0=None, sigma=None, absolute_sigma=False, check_finite=True, **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 M-length sequence or an (k,M)-shaped array
    for functions with k predictors.
    The independent variable where the data is measured.
ydata : M-length sequence
    The dependent data --- nominally f(xdata, ...)
p0 : None, scalar, or N-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 M-length sequence, optional
    If not None, the uncertainties in the ydata array. These are used as
    weights in the least-squares problem
    i.e. minimising ‘‘np.sum( ((f(xdata, *popt) - ydata) / sigma)**2 )‘‘
    If None, the uncertainties are assumed to be 1.
absolute_sigma : bool, optional
    If False, ‘sigma‘ denotes relative weights of the data points.
    The returned covariance matrix ‘pcov‘ is based on *estimated*
    errors in the data, and is not affected by the overall
    magnitude of the values in ‘sigma‘. Only the relative
    magnitudes of the ‘sigma‘ values matter.

    If True, ‘sigma‘ describes one standard deviation errors of
    the input data points. The estimated covariance in ‘pcov‘ is
    based on these values.
check_finite : bool, optional
    If True, check that the input arrays do not contain nans of infs,
    and raise a ValueError if they do. Setting this parameter to
    False may silently produce nonsensical results if the input arrays
    do contain nans.
    Default is True.

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. To compute one standard deviation errors
    on the parameters use ‘‘perr = np.sqrt(np.diag(pcov))‘‘.

    How the ‘sigma‘ parameter affects the estimated covariance
    depends on ‘absolute_sigma‘ argument, as described above.
```

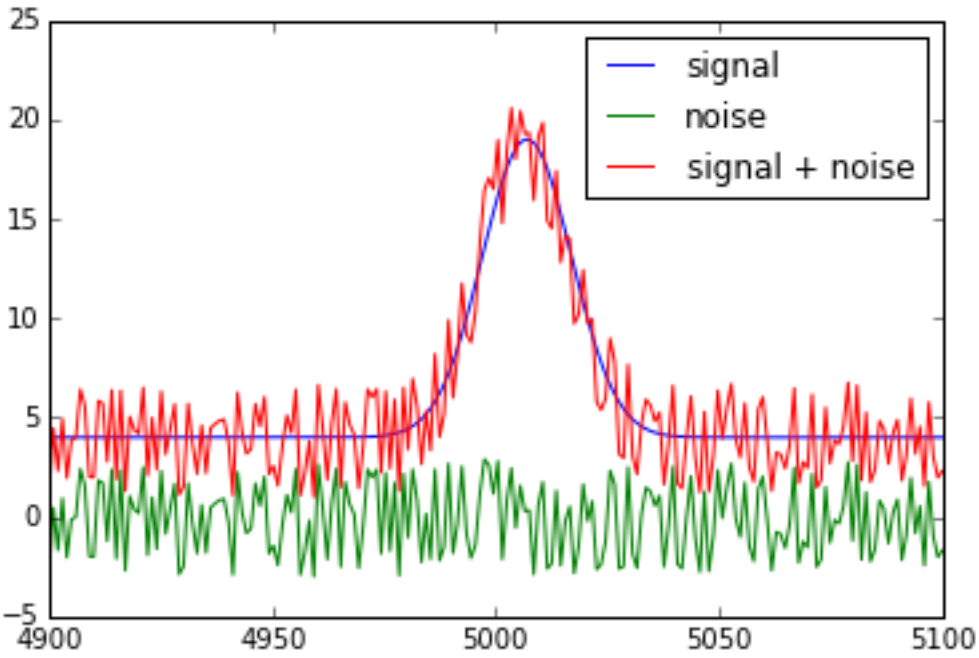
```
    Raises
    ------
    OptimizeWarning
        if covariance of the parameters can not be estimated.

    ValueError
        if ydata and xdata contain NaNs.

    See Also
    --------
    leastsq

    Notes
    -----
    The algorithm uses the Levenberg-Marquardt algorithm through `leastsq`.
    Additional keyword arguments are passed directly to that algorithm.

    Examples
    --------
    >>> import numpy as np
    >>> from scipy.optimize import curve_fit
    >>> def func(x, a, b, c):
    ...     return a * np.exp(-b * x) + c

    >>> xdata = np.linspace(0, 4, 50)
    >>> y = func(xdata, 2.5, 1.3, 0.5)
    >>> ydata = y + 0.2 * np.random.normal(size=len(xdata))

    >>> popt, pcov = curve_fit(func, xdata, ydata)
```

In [40]:
```
def gauss(x, A, B, C, S):
    # This is a gaussian function.
    return A + B*np.exp(-1 * (x - C)**2 / (2 * S**2))
```

In [41]:
```
# We define the parameters used to generate the signal (gaussian at lambda=5007)
N_lam = 200
A = 4.
B = 15.
Lam0 = 5007.
Sigma = 10.
# We define a wavelength range
lam = np.linspace(4900, 5100, N_lam)
# Computing the signal
fl = gauss(lam, A, B, Lam0, Sigma)
f, ax =plt.subplots()
ax.plot(lam, fl)
ax.set_ylim(0,20);
```



In [42]:
```
SN = 5. # Signal/Noise
noise = B / SN * (np.random.rand(N_lam)*2 - 1)
fl2 = fl + noise
f, ax =plt.subplots()
ax.plot(lam, fl, label='signal')
ax.plot(lam, noise, label='noise')
ax.plot(lam, fl2, label='signal + noise')
ax.legend(loc='best');
```



In [43]:
```
# Initial guess:
A_i = 0.
B_i = 1.
Lam0_i = 5000.
Sigma_i = 1.
fl_init = gauss(lam, A_i, B_i, Lam0_i, Sigma_i)
error = np.ones_like(lam) * np.mean(np.abs(noise)) # We define the error (the same on each pix
```

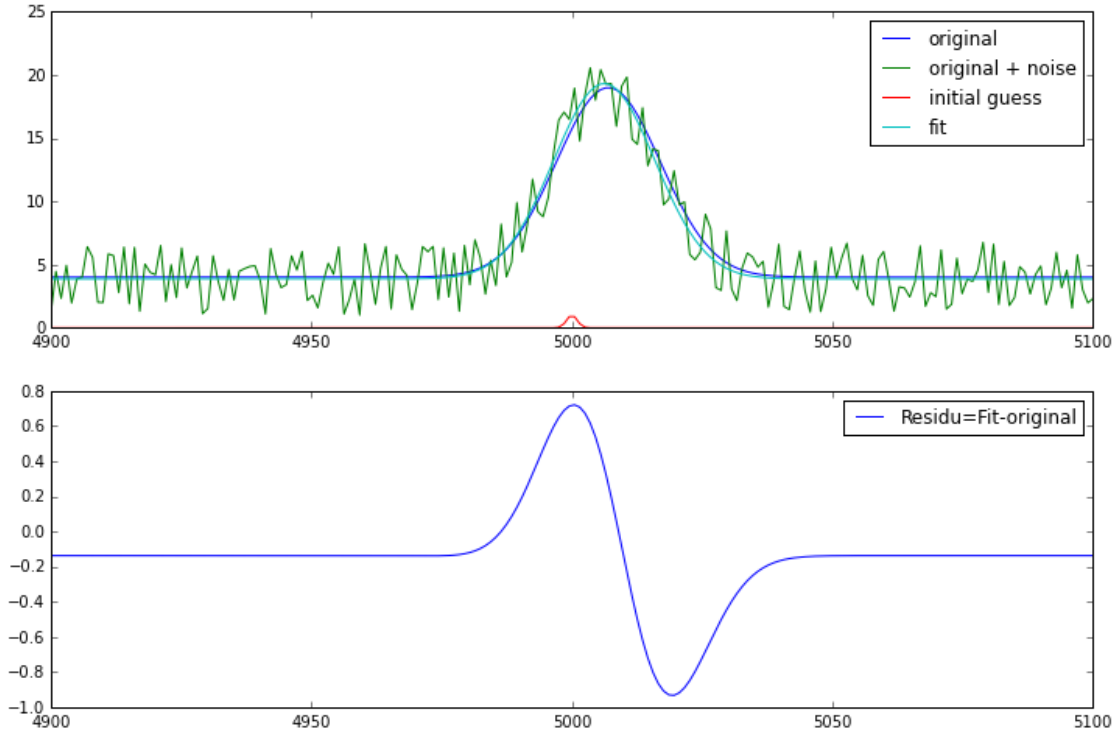
In [49]:
```
# fitting the noisy data with the gaussian function, using the initial guess and the errors
fit, covar = curve_fit(gauss, lam, fl2, [A_i, B_i, Lam0_i, Sigma_i], error)
print('   A     B    Lam0     S')
print('{0:.2f} {1:5.2f} {2:.2f} {3:5.2f}'.format(A, B, Lam0, Sigma))
print('{0:.2f} {1:5.2f} {2:.2f} {3:5.2f}'.format(A_i, B_i, Lam0_i, Sigma_i))
print('{0[0]:.2f} {0[1]:5.2f} {0[2]:5.2f} {0[3]:.2f}'.format(fit))
```

```
A     B    Lam0     S
4.00 15.00 5007.00 10.00
0.00  1.00 5000.00  1.00
3.86 15.46 5006.21 9.72
```

In [45]:
```
# Computing the fit on the lambdas
fl_fit = gauss(lam, fit[0], fit[1], fit[2], fit[3])
```

In [46]:
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

ax1.plot(lam, fl, label='original')
ax1.plot(lam, fl2, label='original + noise')
ax1.plot(lam, fl_init, label='initial guess')
ax1.plot(lam, fl_fit, label='fit')
ax1.legend()
```

```
ax2.plot(lam, fl_fit - fl, label='Residu=Fit-original')
ax2.legend();
```




```
In [47]: # Integrating using the Simpson method the gaussian (without the continuum)
         print simps(fl - A, lam)
         print simps(fl2 - fit[0], lam)
         print simps(fl_fit - fit[0], lam)
```

```
375.994241195
379.002413694
376.529447126
```

```
In [50]: khi_sq = (((fl2-fl_fit) / error)**2).sum() # The problem here is to determine the error...
         khi_sq_red = khi_sq / (len(lam) - 4 - 1) # reduced khi_sq = khi_sq / (N - free_params - 1)
         print khi_sq, khi_sq_red
```

```
252.647118662 1.29562624955
```

1.0.6 Multivariate estimation

```
In [51]: from scipy import stats
```

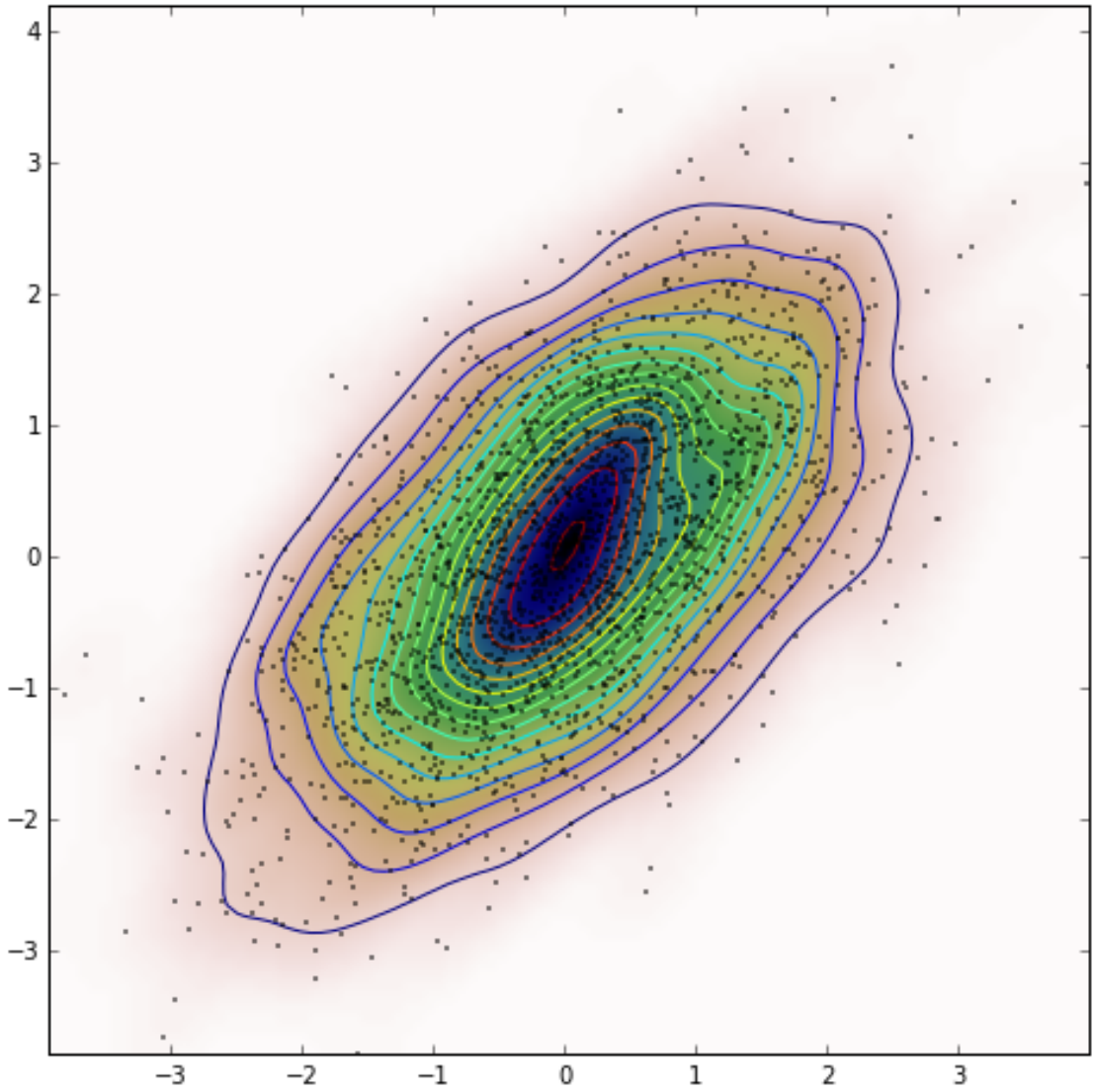
```
In [52]: def measure(n):
             """Measurement model, return two coupled measurements."""
             m1 = np.random.normal(size=n)
             m2 = np.random.normal(scale=0.5, size=n)
             return m1+m2, m1-m2
```

```
In [54]: # Define the
         m1, m2 = measure(2000)
         xmin = m1.min()
         xmax = m1.max()
         ymin = m2.min()
         ymax = m2.max()
         print xmin, xmax, ymin, ymax
```

```
-3.93164628862 3.98751403531 -3.7789894204 4.18726125807
```

```
In [55]: X, Y = np.mgrid[xmin:xmax:100j, ymin:ymax:100j]
         positions = np.vstack([X.ravel(), Y.ravel()])
         values = np.vstack([m1, m2])
         kernel = stats.gaussian_kde(values)
         Z = np.reshape(kernel.evaluate(positions).T, X.shape)
```

```
In [56]: fig, ax = plt.subplots(figsize=(12, 8))
         ax.imshow(np.rot90(Z), cmap=plt.cm.gist_earth_r, extent=[xmin, xmax, ymin, ymax], origin='uppe
         ax.plot(m1, m2, 'k.', markersize=2)
         ax.set_xlim([xmin, xmax])
         ax.set_ylim([ymin, ymax])
         levels = [0.15, 0.14, 0.13, 0.12, 0.11, 0.10, 0.09, 0.08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02,
         cs = ax.contour(X, Y, Z, levels=levels); # I dont't know what those levels mean... but it work
```



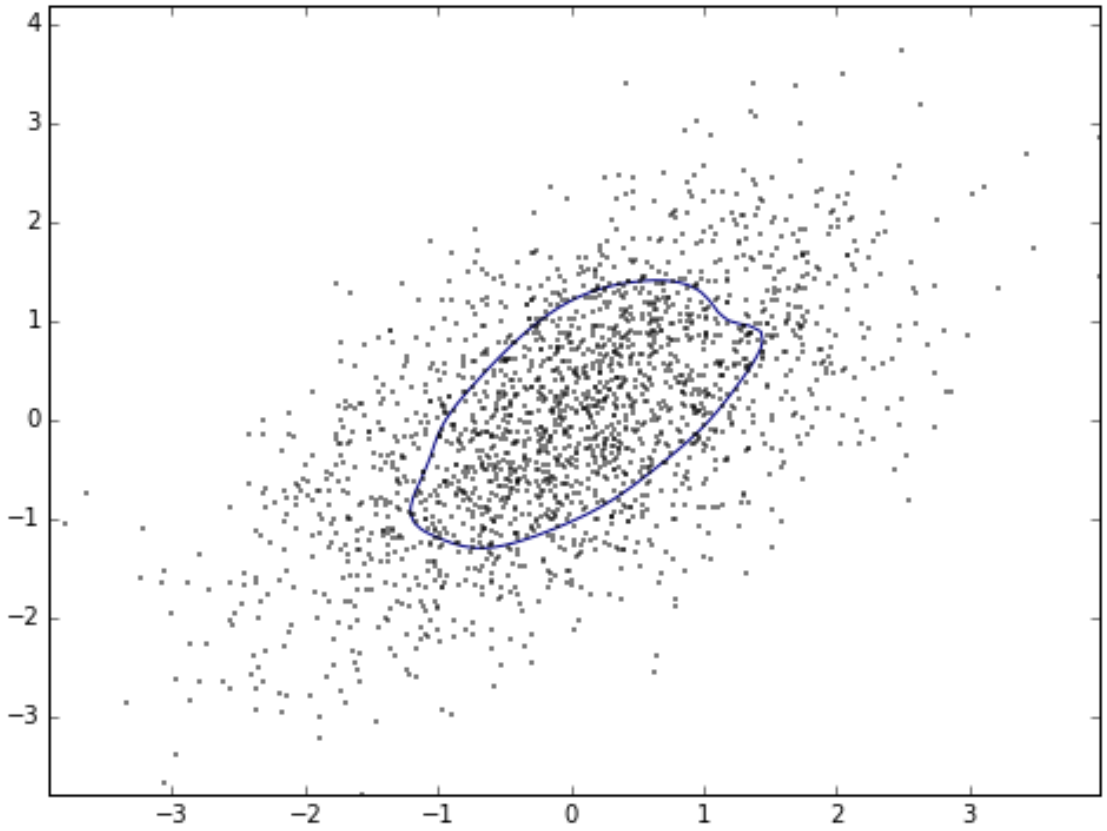
In [57]:
```
# We save the contour paths in a list
paths = []
for collec in cs.collections:
    try:
        paths.append(collec.get_paths()[0])
    except:
        pass
```

In [58]:
```
# Looking for the number of points inside each contour
print len(m1)
for level, path in zip(levels, paths):
    print('level {0:4.2f} contains {1:2.0f}% of the data'.format(level,
                                                                 path.contains_points(zip(m1, m2)
```

```
2000
level 0.15 contains  1% of the data
```

```
level 0.14 contains  8% of the data
level 0.13 contains 15% of the data
level 0.12 contains 22% of the data
level 0.11 contains 28% of the data
level 0.10 contains 35% of the data
level 0.09 contains 42% of the data
level 0.08 contains 49% of the data
level 0.07 contains 56% of the data
level 0.06 contains 62% of the data
level 0.05 contains 68% of the data
level 0.04 contains 75% of the data
level 0.03 contains 82% of the data
level 0.02 contains 89% of the data
level 0.01 contains 95% of the data
```

```
In [59]: fig, ax = plt.subplots(figsize=(8, 6))
         ax.plot(m1, m2, 'k.', markersize=2)
         ax.set_xlim([xmin, xmax])
         ax.set_ylim([ymin, ymax])
         cs = ax.contour(X, Y, Z, levels=[0.078]); #  seems to correspond to 50% of the points inside
```

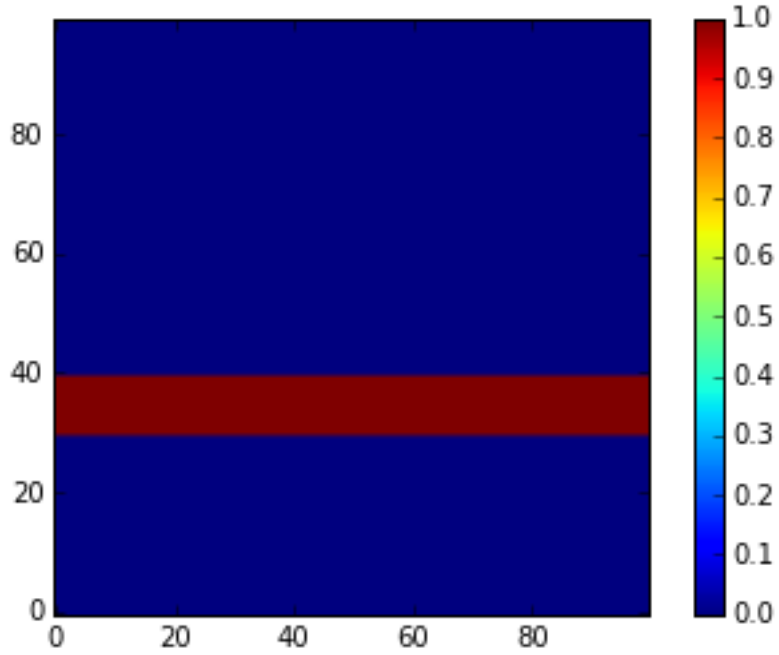


1.0.7 Convolution

More information there: http://docs.scipy.org/doc/scipy/reference/tutorial/ndimage.html

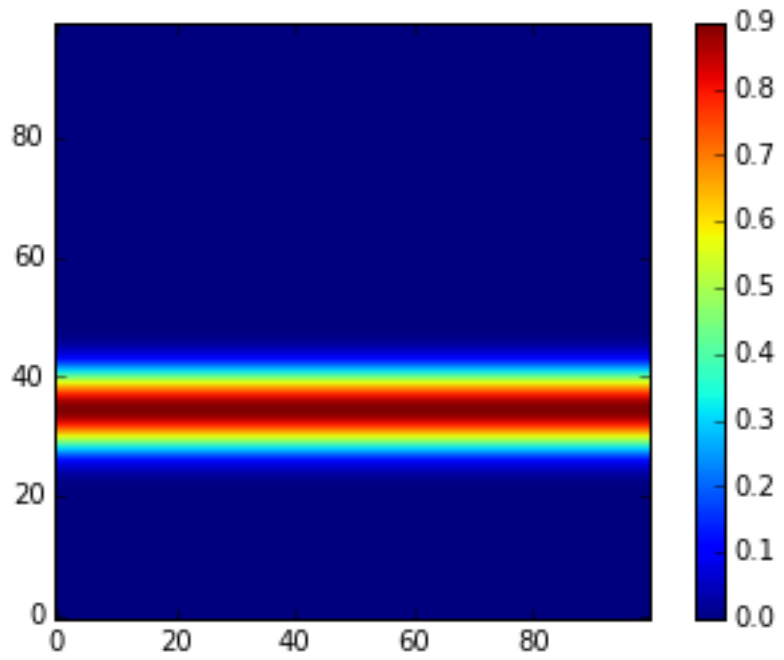
In [61]:
```
# Let's define an image representing a long slit of width 10 pixels
slit = np.zeros((100, 100))
slit[30:40, :] = 1
```

In [62]:
```
plt.imshow(slit)
plt.colorbar();
```

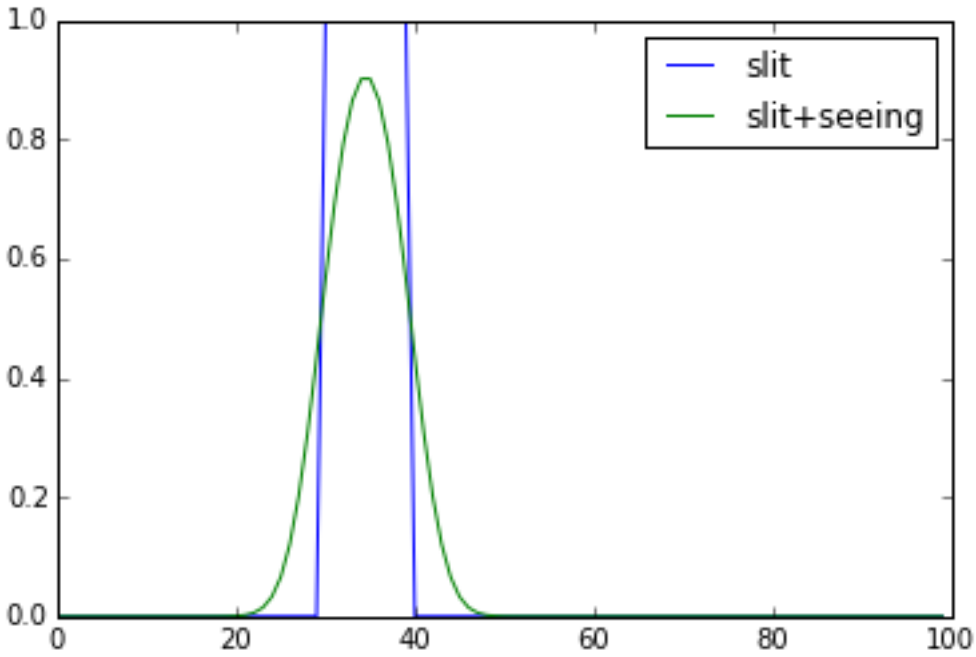


In [63]:
```
# This is the routine to apply a gaussian convolution
from scipy.ndimage.filters import gaussian_filter
```

In [72]:
```
slit_seeing = gaussian_filter(slit, 3) # Convolve with a gaussian, 3 is the standard deviation
plt.imshow(slit_seeing)
plt.colorbar();
```



In [73]:
```
f, ax =plt.subplots()
ax.plot(slit[:,50], label='slit') # original slit
ax.plot(slit_seeing[:,50], label='slit+seeing') # slit with seeing
ax.legend(loc='best');
```



```
In [67]: # Check that the slit transmission is conserved:
         print simps(slit[:,50]), simps(slit_seeing[:,50])
```

```
10.0 10.0
```

1.0.8 Quantiles

```
In [74]: from scipy.stats.mstats import mquantiles
```

```
In [76]: #help(mquantiles)
```

```
In [77]: data = np.random.randn(1000)
```

```
In [78]: mquantiles(data, [0.16, 0.84]) # should return something close to -1, 1 (the stv of the normal
```

```
Out[78]: array([-1.01830112,  0.96926378])
```

```
In [79]: data = np.array([[   6.,    7.,    1.],
                          [  47.,   15.,    2.],
                          [  49.,   36.,    3.],
                          [  15.,   39.,    4.],
                          [  42.,   40., -999.],
                          [  41.,   41., -999.],
                          [   7., -999., -999.],
                          [  39., -999., -999.],
                          [  43., -999., -999.],
                          [  40., -999., -999.],
                          [  36., -999., -999.]])
```

```
In [80]: mq = mquantiles(data, axis=0, limit=(0, 50))
         print mq
         print type(mq)
         mq?
         print mq.mask
```

```
[[ 19.2   14.6    1.45]
 [ 40.    37.5    2.5 ]
 [ 42.8   40.05   3.55]]
<class 'numpy.ma.core.MaskedArray'>
False
```

1.0.9 Input/Output

Scipy has many modules, classes, and functions available to read data from and write data to a variety of file formats.

Including MATLAB and IDL files. See http://docs.scipy.org/doc/scipy/reference/io.html