

Astropy1

January 30, 2019

```
In [4]: %matplotlib inline
```

```
In [3]: from astropy import constants as const
       from astropy import units as u
```

```
In [30]: print u.eV.to(u.erg)
```

```
      print (1*u.Jy).cgs.value
```

```
1.602176565e-12
1e-23
```

```
In [2]: print const.c
```

```
      print const.c.cgs
```

```
      print const.c.to('km/s')
      print const.c.to('pc/yr')
```

```
Name    = Speed of light in vacuum
Value   = 299792458.0
Uncertainty = 0.0
Unit    = m / s
Reference = CODATA 2010
29979245800.0 cm / s
299792.458 km / s
0.306601393788 pc / yr
```

```
In [23]: ?const
```

```
In [4]: F = (const.G.cgs * 3.*const.M_sun.cgs* 100* u.kg) / (2.2*u.au)**2
       print F
      print F.to(u.dyne)
      print F.to(u.N)
```

```
8.22826558512e+27 cm3 kg / (AU2 s2)
36766.9392028 dyn
0.367669392028 N
```

```
In [19]: a = 42.*u.cm
```

```
print a.value
print a.unit

x = 1.*u.pc
print x.to(u.km)
```

```
42.0
cm
3.08567758147e+13 km
```

```
In [6]: """ create custom units """
```

```
cms = u.cm / u.s
mph = u.imperial.mile / u.hour

q = 42.*cms

print q
print q.to(mph)
```

```
42.0 cm / s
0.939513242663 mi / h
```

```
In [7]: print 1100.1*u.m + 13.5*u.km
print 13.5*u.km + 1100.1*u.m
```

```
14600.1 m
14.6001 km
```

```
In [8]: import numpy as np
```

```
print np.sin(30.*u.deg)

print np.sin(np.pi/6.)
```

```
0.5
0.5
```

```
In [9]: """ dimensionless quantities """
nu = 3.*u.GHz
T = 30.*u.K

print np.exp(-const.h * nu / (const.k_B * T))

nu2 = 3.e9*u.Hz
print np.exp(-const.h * nu2 / (const.k_B * T))

0.995212254619
0.995212254619

In [10]: a = 1. + 1.*u.m/u.km
print a

b = 1.+(1.*u.m/u.km).value
print b

c = (1.*u.m/u.km)
print c
print c.unit
print c.unit.decompose()

1.001
2.0
1.0 m / km
m / km
0.001

In [11]: print u.dimensionless_unscaled == u.Unit('')

True

In [12]: print u.g.find_equivalent_units()

Primary name | Unit definition | Aliases
[

|             |                |                                    |
|-------------|----------------|------------------------------------|
| M_e         | 9.10938e-31 kg | ,                                  |
| M_p         | 1.67262e-27 kg | ,                                  |
| earthMass   | 5.9742e+24 kg  | M_earth, Mearth ,                  |
| g           | 0.001 kg       | gram ,                             |
| jupiterMass | 1.8987e+27 kg  | M_jup, Mjup, M_jupiter, Mjupiter , |
| kg          | irreducible    | kilogram ,                         |
| solMass     | 1.9891e+30 kg  | M_sun, Msun ,                      |
| t           | 1000 kg        | tonne ,                            |
| u           | 1.66054e-27 kg | Da, Dalton ,                       |

]
```

0.1 decomposing and composing units

```
In [13]: print (u.km / u.m).decompose()
          print (u.km / u.m).physical_type

1000
dimensionless

In [14]: print u.Ry

          print u.Ry.decompose(bases = u.cgs.bases)

Ry
2.17987e-11 cm2 g / s2
```

```
In [15]: x = u.Ry.decompose()

          print x.compose()

[Unit("Ry"), Unit("2.17987e-18 J"), Unit("2.17987e-11 erg"), Unit("13.6057 eV")]
```

```
In [16]: """ compound units """
          (u.s**-1).compose()

Out[16]: [Unit("Hz"), Unit("Bq"), Unit("2.7027e-11 Ci")]
```

```
In [17]: (u.s**-1).compose(equivalencies=u.spectral())
```

```
Out[17]: [Unit("Bq"),
          Unit("Hz"),
          Unit("6.62607e-34 J"),
          Unit("6.62607e-27 erg"),
          Unit("3.03966e-16 Ry"),
          Unit("4.13567e-15 eV"),
          Unit("2.7027e-11 Ci"),
          Unit("3.33564e-11 k"),
          Unit("9.71561e-09 pc"),
          Unit("3.16881e-08 lyr"),
          Unit("0.00200399 AU"),
          Unit("0.431041 solRad"),
          Unit("4.19337 jupiterRad"),
          Unit("47.0031 earthRad"),
          Unit("2.99792e+08 m"),
          Unit("2.99792e+10 cm"),
          Unit("2.99792e+14 micron"),
          Unit("2.99792e+18 Angstrom"),
          Unit("2.99792e+06 k m2"),
```

```

Unit("3.33564e-09 / m"),
Unit("3.50331e-42 pc2 / m3"),
Unit("3.72675e-41 lyr2 / m3"),
Unit("1.49049e-31 AU2 / m3"),
Unit("6.89565e-27 solRad2 / m3"),
Unit("6.52626e-25 jupiterRad2 / m3"),
Unit("8.19958e-23 earthRad2 / m3"),
Unit("3.33564e-05 cm2 / m3"),
Unit("3335.64 micron2 / m3"),
Unit("3.33564e+11 Angstrom2 / m3")]

```

In [18]: *""" Temperature Energy Equivalency """*

```

t_k = 1.e6*u.K

print t_k.to(u.eV, equivalencies = u.temperature_energy())

```

86.1733238496 eV

In [19]: `u.Hz.find_equivalent_units()`

```

Out[19]: Primary name | Unit definition | Aliases
[
    Bq           | 1 / s           | becquerel      ,
    Ci           | 3.7e+10 / s       | curie          ,
    Hz           | 1 / s           | Hertz, hertz  ,
]

```

In [20]: `u.Hz.find_equivalent_units(equivalencies=u.spectral())`

```

Out[20]: Primary name | Unit definition           | Aliases
[
    AU          | 1.49598e+11 m        | au, astronomical_unit   ,
    Angstrom    | 1e-10 m            | AA, angstrom          ,
    Bq           | 1 / s             | becquerel            ,
    Ci           | 3.7e+10 / s         | curie                ,
    Hz           | 1 / s             | Hertz, hertz          ,
    J            | kg m2 / s2         | Joule, joule          ,
    Ry           | 2.17987e-18 kg m2 / s2 | rydberg              ,
    cm           | 0.01 m            | centimeter           ,
    eV           | 1.60218e-19 kg m2 / s2 | electronvolt          ,
    earthRad    | 6.37814e+06 m       | R_earth, Rearth       ,
    erg          | 1e-07 kg m2 / s2     |                      ,
    jupiterRad | 7.1492e+07 m       | R_jup, Rjup, R_jupiter, Rjupiter ,
    k            | 100 / m            | Kayser, kayser        ,
    lyr          | 9.46073e+15 m       | lightyear             ,
    m            | irreducible         | meter                 ,
    micron       | 1e-06 m            |                      ,
]
```

```

pc           | 3.08568e+16 m           | parsec
solRad       | 6.95508e+08 m           | R_sun, Rsun
]

In [21]: print u.m.is_equivalent(u.pc)
          print (u.m**3).is_equivalent(u.l)

True
True

```

0.2 Models and Fitting

```

In [22]: from astropy.modeling import models
          import numpy as np
          import matplotlib.pyplot as plt

g = models.Gaussian1D(amplitude=1.2, mean=0.9, stddev=0.5)

print (g)

print g.amplitude
print g.mean
print g.stddev

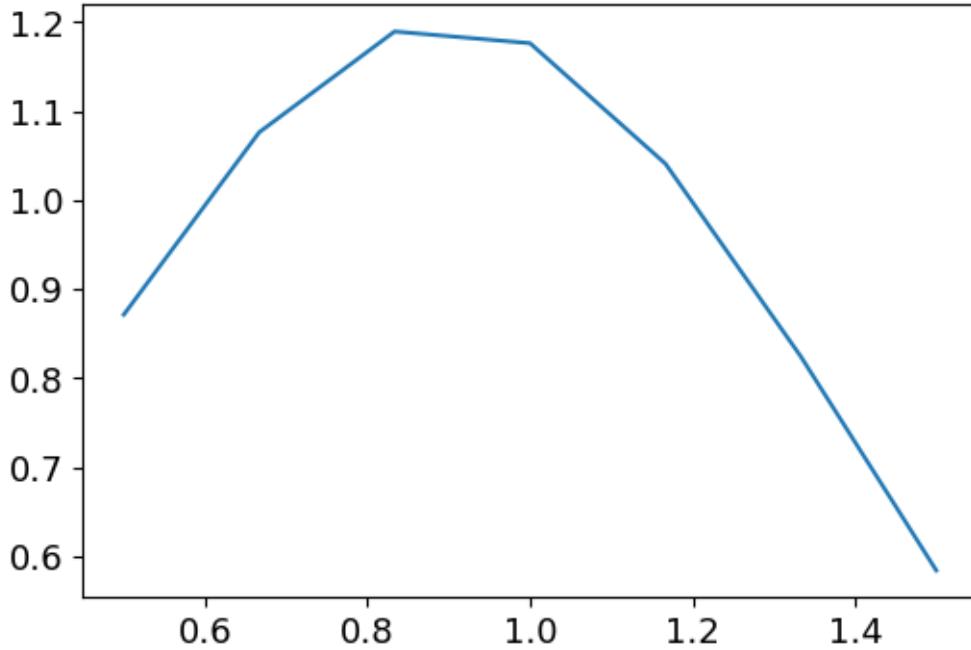
print g(0.1)

x = np.linspace(0.5,1.5,7)
y = g(x)
print y

plt.plot(x,y)
plt.show()

Model: Gaussian1D
Inputs: (u'x',)
Outputs: (u'y',)
Model set size: 1
Parameters:
  amplitude  mean  stddev
  -----  -----
    1.2    0.9    0.5
Parameter('amplitude', value=1.2)
Parameter('mean', value=0.9)
Parameter('stddev', value=0.5)
0.333644760544
[ 0.87137884  1.07619607  1.1893806   1.17623841  1.04091417  0.82428907
  0.58410271]

```



```
In [23]: import numpy as np
        from astropy.modeling import models, fitting
        import matplotlib.pyplot as plt

        # Generate face data
        np.random.seed(0)
        x = np.linspace(-5.,5.,200)
        y = 3*np.exp(-0.5*(x-1.3)**2 / 0.8**2)
        y += np.random.normal(0., 0.2, x.shape)

        # Fit the data using a box model
        t_init = models.Trapezoid1D(amplitude=1., x_0=0., width=1., slope=0.5)
        fit_t = fitting.LevMarLSQFitter()
        t = fit_t(t_init, x, y)

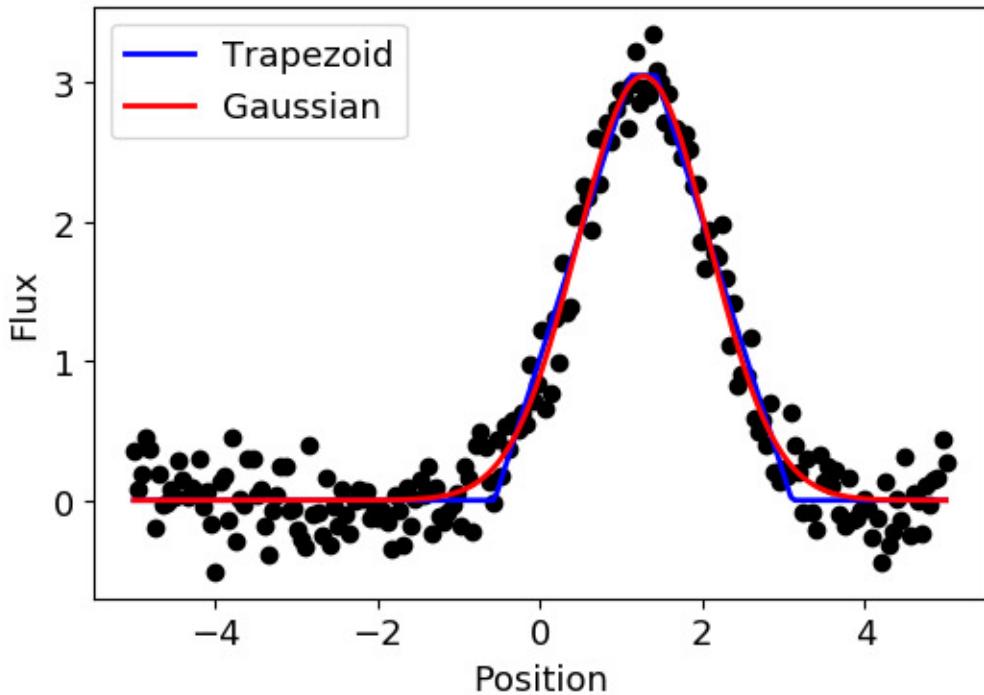
        # Fit the data using a Gaussian
        g_init = models.Gaussian1D(amplitude=1., mean=0., stddev=1.)
        fit_g = fitting.LevMarLSQFitter()
        g = fit_g(g_init, x, y)

        # Plot the data with the best-fit model
        # plt.figure(figsize=(8,5))
        plt.plot(x, y, 'ko')
        plt.plot(x, t(x), 'b-', lw=2, label='Trapezoid')
        plt.plot(x, g(x), 'r-', lw=2, label='Gaussian')
```

```

plt.xlabel('Position')
plt.ylabel('Flux')
plt.legend(loc=2, fontsize='medium')
plt.show()

```



```

In [24]: import numpy as np
         from astropy.modeling import models, fitting
         import matplotlib.pyplot as plt

         # Generate fake data
         np.random.seed(42)
         g1 = models.Gaussian1D(1, 0, 0.2)
         g2 = models.Gaussian1D(2.5, 0.5, 0.1)
         x = np.linspace(-1, 1, 200)
         y = g1(x) + g2(x) + np.random.normal(0., 0.2, x.shape)

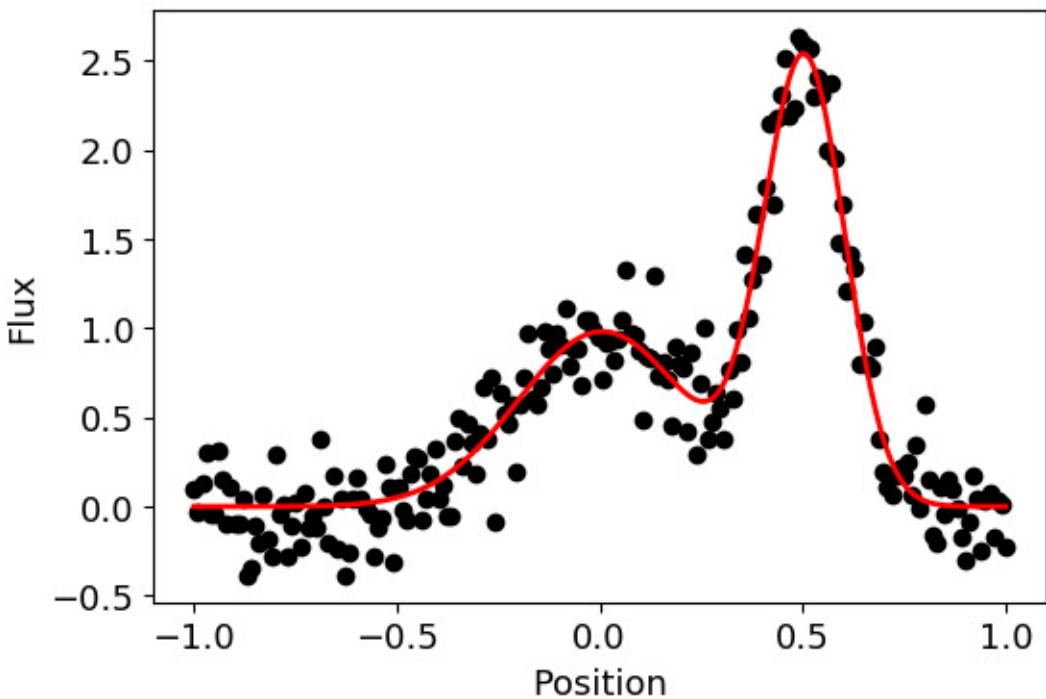
         # Now to fit the data create a new superposition with initial
         # guesses for the parameters:
         gg_init = models.Gaussian1D(1, 0, 0.1) + models.Gaussian1D(2, 0.5, 0.1)
         fitter = fitting.SLSQPLSQFitter()
         gg_fit = fitter(gg_init, x, y)

         # Plot the data with the best-fit model
         #plt.figure(figsize=(8,5))

```

```
plt.plot(x, y, 'ko')
plt.plot(x, gg_fit(x), 'r-', lw=2)
plt.xlabel('Position')
plt.ylabel('Flux')
plt.show()
```

```
Optimization terminated successfully.      (Exit mode 0)
Current function value: 6.83285936044
Iterations: 14
Function evaluations: 137
Gradient evaluations: 14
```



astrounit

January 30, 2019

1 pyds.astro

1.1 pyds.astro.astrounit

- For the purpose of quick reading the astronomical constants in cgs unit.
 - This module may be particularly useful if you compile it in the start-up batch file.
1. add PYTHONSTARTUP variable to .bashrc export PYTHONSTARTUP=\$PYTHONPATH/startup.py or .tcshrc setenv PYTHONSTARTUP \$PYTHONPATH/startup.py
 2. in startup.py, add the line below: from pyds.astro import astrounit as unit
 3. Then, whenever you open the python, you can use the units without hassle.

In [1]: `from pyds.astro import astrounit as unit`

```
unit.info()
```

name	name in astropy	value in cgs	unit
<hr/>			
me	m_e	9.109383e-28	g
kb	k_B	1.380649e-16	erg / K
c	c	2.997925e+10	cm / s
rsun	R_sun	6.955080e+10	cm
g	G	6.673840e-08	cm3 / (g s2)
pc	pc	3.085678e+18	cm
h	h	6.626070e-27	erg s
mn	m_n	1.674927e-24	g
mh	m_p	1.672622e-24	g
sigmaSB	sigma_sb	5.670373e-05	g / (K4 s3)
msun	M_sun	1.989100e+33	g
kpc	kpc	3.085678e+21	cm
au	au	1.495979e+13	cm
mp	m_p	1.672622e-24	g
sigmaT	sigma_T	6.652459e-25	cm2
lsun	L_sun	3.846000e+33	erg / s
amu	u	1.660539e-24	g
<hr/>			
not in astropy:			
year	not in astropy	3.153600e+07	s

lyr	not in astropy	9.454255e+17	cm
eV	not in astropy	1.602177e-12	erg
Jy	not in astropy	1.000000e-23	erg / (cm ² s Hz)
re	not in astropy	2.817941e-13	cm

```
In [2]: from pyds.astro import astrounit as unit
```

```
print unit.g, unit.pc, unit.lyr
```

```
6.67384e-08 3.08567758147e+18 9.45425495549e+17
```

In [3]: *""" In fact, the most values are referred to those of astropy package. This code is aimed to call the astrophysical costant in cgs unit somewhat easily. """*

```
from astropy import constants as cons
from pyds.astro import astrounit as unit
```

```
print cons.G.cgs.value, unit.g
```

```
6.67384e-08 6.67384e-08
```

```
In [4]: import pyds.astro as astro
```

```
print astro.info()
```

name	name in astropy	value in cgs	unit
<hr/>			
me	m_e	9.109383e-28	g
kb	k_B	1.380649e-16	erg / K
c	c	2.997925e+10	cm / s
rsun	R_sun	6.955080e+10	cm
g	G	6.673840e-08	cm ³ / (g s ²)
pc	pc	3.085678e+18	cm
h	h	6.626070e-27	erg s
mn	m_n	1.674927e-24	g
mh	m_p	1.672622e-24	g
sigmaSB	sigma_sb	5.670373e-05	g / (K ⁴ s ³)
msun	M_sun	1.989100e+33	g
kpc	kpc	3.085678e+21	cm
au	au	1.495979e+13	cm
mp	m_p	1.672622e-24	g
sigmaT	sigma_T	6.652459e-25	cm ²
lsun	L_sun	3.846000e+33	erg / s
amu	u	1.660539e-24	g
<hr/>			
not in astropy:			
year	not in astropy	3.153600e+07	s

lyr	not in astropy	9.454255e+17	cm
eV	not in astropy	1.602177e-12	erg
Jy	not in astropy	1.000000e-23	erg / (cm ² s Hz)
re	not in astropy	2.817941e-13	cm
None			

1.2 pyds.astro.astroeq

- Formulae in astronomy
- Most of equations fit to be in cgs unit.
- Some equations may need to be double-checked before their usages.

```
In [5]: from pyds.astro import astroeq as eq
        from pyds.astro import astrounit as unit

        print eq.Ledd(1e9*unit.msun)

        print eq.rsh(1.e9*unit.msun)

        print eq.cs(rho=1e-20, P=1e1) / 1e5

        print eq.tff(rho=1e10)

1.25741469797e+47
2.95407146642e+14
408248.290464
0.0210074181092
```

```
In [6]: help(eq)
```

Help on module pyds.astro.astroeq in pyds.astro:

NAME

pyds.astro.astroeq

FILE

/home/astrodoo/lib/py_lib/pyds/astro/astroeq.py

DESCRIPTION

filename:

astroeq.py

PURPOSE:

collection of formulae in astronomy

Written by:

Doosoo Yoon

Shanghai Astronomical Observatory

History:

Written, 22 November 2017

FUNCTIONS

Ledd(mbh)

Eddington Luminosity

mbh should be in cgs unit.

returned value would be in erg/s unit.

Ljeans(T=0.0, rho=1.0, mmw=1.3)

Jean's length (assume the uniform density at the spherical shape) in cm unit

Ljeans = 2 x Rjeans, where Rjeans = ($M_{\text{jeans}} / (4/3 \pi \rho)$)^(1/3)
(using eq.(5.27) for Rjeans in astropedia)

keywords:

T: temperature in K

rho: density in g/cm³

mmw: mean molecular weight (default=1.3 for neutral solar abundance)

Mdotbondi(mbh=1.0, rho=1.0, cs=1.0)

Bondi Accretion rate

mbh, rho, cs(sound speed) should be in cgs unit
returned value would be in cm unit.

keywords:

mbh: black hole mass in g unit

rho: density in g/cm³ unit

cs: sound speed in cm/s unit

Mdottededd(mbh, radeff=0.1)

Eddington BH mass accretion rate

mbh should be in cgs unit.

returned value would be in g/s unit

args:

mbh: black hole mass in g unit

keywords:

radeff: radiative efficiency (default=0.1)

Mjeans(T=0.0, rho=1.0, mmw=1.3)

Jean's mass (assume the uniform density at the spherical shape) in g unit
(eq.(5.26) in astropedia)

keywords:

T: temperature in K

rho: density in g/cm³

```

mmw: mean molecular weight (default=1.3 for neutral solar abundance)

cs(gamma=1.6666666666666667, **keywords)
    sound speed
    Either of T (temperature) or P (pressure) & rho (density) should be entered
    keywords:
        gamma: adiabatic index (default: 5./3.)
    **keywords:
        T: temperature in K
        P: pressure in cgs
        rho: density in cgs
        mmw: mean molecular weight (default: 0.62 for fully ionized)

rbondi(mbh=1.0, cs=1.0)
    Bondi radius
    mbh and cs(sound speed) should be in cgs unit
    returned value would be in cm unit.

    keywords:
        mbh: black hole mass in g unit
        cs: sound speed in cm/s unit

rsh(mbh)
    Schwarzschild radius
    mbh shoud be in cgs unit.
    returned value would be in cm unit.

tff(rho=1.0)
    free-fall time scale (assume the uniform density at the spherical shape) in s unit
    (eq.(5.28) in astropedia)

    keywords:
        rho: density in g/cm3

vkepl(pointmass, r=1.0)
    Keplerian velocity in the gravitational potential due to point mass.
    returned value would be in cm/s

    args:
        pointmass: mass of the central point in g
    keywords:
        r: radius

```

Basic1

January 30, 2019

```
In [1]: s='Hello world'  
       print s
```

```
Hello world
```

```
In [2]: %whos
```

Variable	Type	Data/Info
s	str	Hello world

```
In [3]: a = 3  
       b = 2*a  
       type(b)
```

```
Out[3]: int
```

```
In [4]: a = 1.5 + 0.5j  
       print 'real: ', a.real  
       print 'img: ', a.imag  
       print type(a)
```

```
real: 1.5  
img: 0.5  
<type 'complex'>
```

0.0.1 List

- mutable

mutable objects can be changed in place, while immutable objects cannot be modified once created

```
In [5]: # Long lines can be broken with \  
       L = ['red','blue','green', \  
             'black','white']
```

```

type(L)
print L[0],L[-1]
# important!! range from start to stop-1 [start:stop:step]
print L[2:4]

print 'red' in L
print 'yellow' in L

red white
['green', 'black']
True
False

In [6]: L=[3,-200,'hello']
        print L
        type(L[0])

[3, -200, 'hello']

Out[6]: int

In [7]: L = ['red','blue','green','black','white']
        L.append('pink')
        print L
        L.pop()
        print L
        L.pop(2)
        print L

        r=L[::-1]
        print type(r)
        print r
        r.reverse()
        print r
        print sorted(r)
        r.sort()
        print r

['red', 'blue', 'green', 'black', 'white', 'pink']
['red', 'blue', 'green', 'black', 'white']
['red', 'blue', 'black', 'white']
<type 'list'>
['white', 'black', 'blue', 'red']
['red', 'blue', 'black', 'white']
['black', 'blue', 'red', 'white']
['black', 'blue', 'red', 'white']

```

```
In [8]: a = [i**2 for i in range(4)]
      print a
      print type(a)
```

```
[0, 1, 4, 9]
<type 'list'>
```

```
In [9]: """mutable vs. immutable"""
      def try_to_modify(x,y,z):
```

```
          x = 23
          y.append(42)
          z = [99]    # new reference
```

```
      print x, y, z
```

```
      a = 77    # immutable variable
      b = [99]  # mutable variable
      c = [28]
```

```
      print a, b, c
```

```
try_to_modify(a,b,c)
```

```
      print a, b, c
```

```
77 [99] [28]
23 [99, 42] [99]
77 [99, 42] [28]
```

```
In [10]: nums = [1,2,3,4,5]
      a = [i for i in nums]
      print type(a)
      print a
```

```
      b = list(i for i in nums)
      print type(b)
      print b
```

```
<type 'list'>
[1, 2, 3, 4, 5]
<type 'list'>
[1, 2, 3, 4, 5]
```

0.0.2 Dictionary

- mutable

```
In [11]: tel = {'aaa': 123, 'ccc': 345}
           tel['bbb']=433
```

```
print type(tel)
print tel

print 'aaa = ', tel['aaa']

print tel.keys()
print tel.values()

'bbb' in tel

<type 'dict'>
{'aaa': 123, 'bbb': 433, 'ccc': 345}
aaa = 123
['aaa', 'bbb', 'ccc']
[123, 433, 345]
```

```
Out[11]: True
```

```
In [2]: import numpy as np
```

```
tel = {'a':1,'c':2,'b':3}

aaa = {i:np.zeros(10) for i in tel.keys()}

print aaa

{'a': array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]), 'c': array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])}
```

```
In [13]: d = {'a':1, 'b':2, 3:'hello'}
           print d
           print type(d[3])
```

```
{'a': 1, 3: 'hello', 'b': 2}
<type 'str'>
```

```
In [14]: a = {i for i in range(3)}
           print type(a)
           print a

b = {i:i**2 for i in range(3)}
           print type(b)
           print b
```

```
<type 'set'>
set([0, 1, 2])
<type 'dict'>
{0: 0, 1: 1, 2: 4}
```

0.0.3 Tuple

- immutable list

```
In [15]: t = 12334, 34522, 'hello!'
    print type(t)
    print t
    print t[0]
```

```
<type 'tuple'>
(12334, 34522, 'hello!')
12334
```

```
In [1]: def func(x,y):
    a = x+1
    b = y+1
    return (a,)

c, = func(1,2)
d = func(1,2)
print c,d
```

2 (2,)

0.0.4 Operator

```
In [3]: a = [1,2,3]
        print type(a)

        b = a
        print a is b

        b[1] = 'hi'

        print a
        print a[None:None:None]    # None can be used for empty
        print a[::-1]

<type 'list'>
True
[1, 'hi', 3]
```

```
[1, 'hi', 3]  
[1, 'hi', 3]
```

```
In [17]: a = ['a','b','c']  
        print a  
        print id(a)
```

```
a[:] = [1,2,3]  
print a  
print id(a)
```

```
['a', 'b', 'c']  
139807479654448  
[1, 2, 3]  
139807479654448
```

```
In [18]: a = 10
```

```
if a == 1:  
    print(1)  
elif a == 2:  
    print(2)  
else:  
    print('A lot')
```

```
A lot
```

```
In [19]: for i in range(4):  
        print i
```

```
0  
1  
2  
3
```

```
In [20]: for word in ('cool','powerful','readable'):  
        print 'Python is %s' %word
```

```
Python is cool  
Python is powerful  
Python is readable
```

```
In [21]: a = [1,0,2,4]  
        for element in a:  
            if element == 0:  
                continue  
            print 1./element
```

```
1.0  
0.5  
0.25
```

```
In [22]: words = ('cool','powerful','readable')  
        print type(word)  
  
        for i in range(0, len(words)):  
            print i, words[i]  
  
        for index,item in enumerate(words):  
            print index, item  
  
<type 'str'>  
0 cool  
1 powerful  
2 readable  
0 cool  
1 powerful  
2 readable
```

```
In [23]: print range(1,5)  
[1, 2, 3, 4]
```

```
In [24]: d = {'a':1,'b':1.2,'c':1j}  
  
        for key, val in d.iteritems():  
            print 'Key: %s has value: %s' %(key,val)  
  
Key: a has value: 1  
Key: c has value: 1j  
Key: b has value: 1.2
```

```
In [25]: z = 1 + 1j  
        while abs(z) < 100:  
            if z.imag == 0:  
                break  
            z = z**2 + 1  
  
        print z  
(-134+352j)
```

```
In [26]: print 1 == 1.  
        print 1 is 1.  
        print 1 is 1
```

```
True  
False  
True
```

```
In [27]: b = [1,2,3]  
        print 2 in b  
        print 5 in b
```

```
True  
False
```

```
In [28]: message = "Hellow how are you?"  
        mlist = message.split()  
        print mlist  
  
        for word in mlist:  
            print word
```

```
['Hellow', 'how', 'are', 'you?']  
Hellow  
how  
are  
you?
```

0.0.5 Functions

```
In [29]: def double_it(x=2):      # x=2 is the optional parameter in case of no input.  
        return x*2  
  
        print double_it()  
  
        print double_it(3)
```

```
4  
6
```

```
In [30]: """ global variables cannot be modified within the function,  
unless declared global in the funciton """
```

```
x = 5
```

```

# not declare
def setx(y):
    x = y
    print 'x is %d' %x

setx(10)
print x

# declare
def setx(y):
    global x
    x = y
    print 'x is %d' %x

setx(10)
print x

x is 10
5
x is 10
10

```

0.0.6 Special forms of parameters

- *args: any number of positional arguments packed into a tuple
- **kwargs: any number of keyword arguments packed into a dictionary

```

In [31]: def variable_args(*args, **kwargs):
          print 'args is', args
          print 'kwargs is', kwargs

variable_args('one', 'two', x=1, y=2, z=3)

args is ('one', 'two')
kwargs is {'y': 2, 'x': 1, 'z': 3}

```

0.0.7 Importing objects from modules

- Note: if you re-import an object in the old session, you have to reload it
--> reload(objname)

```

In [32]: import sys

print sys.argv
print sys.path

```

```
['/home/astrodoo/anaconda2/lib/python2.7/site-packages/ipykernel/__main__.py', '-f', '/run/user/1000/jupyter/ipykernel/c9a853a2c8e843a2b731f8a5a15a5d47.ipynb'],
[], '/home/astrodoo/heasoft-6.19/x86_64-unknown-linux-gnu-libc2.23-0/lib/python', '/home/astrodoo/heasoft-6.19/x86_64-unknown-linux-gnu-libc2.23-0/lib/python2.7/os.pyc',
[]]
```

In [33]: `import os`

```
pypath = os.environ['PYTHONPATH']      # environment check
print pypath

print os.getcwd()  # current directory

print os.listdir(os.getcwd())
print os.listdir('.')    # '.' represent the current directory
```

```
/home/astrodoo/heasoft-6.19/x86_64-unknown-linux-gnu-libc2.23-0/lib/python:/home/astrodoo/heasoft-6.19/x86_64-unknown-linux-gnu-libc2.23-0/lib/python2.7/:/home/astrodoo/CodeExr/Python/ipython
['astrounit.ipynb', 'tmp.eps', 'matplotlib4.ipynb', 'Astropy1.ipynb', 'img2data.ipynb', 'matplotlib4.ipynb', 'astrounit.ipynb', 'tmp.eps', 'matplotlib4.ipynb', 'Astropy1.ipynb', 'img2data.ipynb', 'matplotlib4.ipynb']
```

In [34]: `who`

```
L      a      aaa      b      c      d      double_it      element
index      item      key      message      mlist      nums      os      p
s      setx      sys      t      tel      try_to_modify      val      var
words      x      z
```

In [35]: `whos`

Variable	Type	Data/Info
L	list	n=4
a	list	n=4
aaa	dict	n=3
b	list	n=3
c	list	n=1
d	dict	n=3
double_it	function	<function double_it at 0x7f2777266a28>
element	int	4
i	int	2
index	int	2
item	str	readable
key	str	b
message	str	Hellow how are you?
mlist	list	n=4
nums	list	n=5
os	module	<module 'os' from '/home/...>a2/lib/python2.7/os.pyc'>
pypath	str	/home/astrodoo/heasoft-6.<...>home/astrodoo/lib/py_lib/

```

r           list      n=4
s           str       Hello world
setx        function   <function setx at 0x7f277723b410>
sys         module    <module 'sys' (built-in)>
t           tuple     n=3
tel          dict     n=3
try_to_modify function  <function try_to_modify at 0x7f2777266c80>
val          float    1.2
variable_args function  <function variable_args at 0x7f2777266c08>
word         str      you?
words        tuple    n=3
x            int      10
z            complex  (-134+352j)

```

```
In [36]: import pyds
        dir(pyds.astro)
```

```
Out[36]: ['__builtins__',
          '__doc__',
          '__file__',
          '__name__',
          '__package__',
          '__path__',
          'astrounit',
          'info']
```

0.1 Input & Output

- Read-only: r
- Write-only: w
- Append a file: a
- Read/Write: r+
- Binary mode: b

```
In [37]: f = open('../data/workfile','w')
        print type(f)

        f.write('This is a test \nand another test')
        f.close()

<type 'file'>
```

```
In [38]: """ Making a data table with formmated data """
        import numpy as np

        Te = np.linspace(0.,100.,1000)
        cool = Te**2.
```

```

coolX = Te**3.

f = open('../data/CoolingX.dat','w')
f.write('Temperature (K)    Cooling (erg cm^-3 s^-1)    Cooling (0.3<Ebin<8 keV) \n')
for i in range(len(Te)):
    f.write('%10.6e \t %10.6e \t %10.6e \n' %(Te[i],cool[i],coolX[i]))

f.close()

```

In [39]: *""" Read the data with column """*

```

coolf = open('../data/CoolingX.dat','r')
text = coolf.readlines(); text = text[1:len(text)]; coolf.close()
cool_all = np.array([[float(num) for num in line.split()] for line in text])

col = {"T":0, "Cool":1, "Coolx":2}

Temp = cool_all[:,col["T"]]
Cool = cool_all[:,col["Cool"]]      # Cooling for all energy bins (erg cm^-3 s^-1)
Coolx = cool_all[:,col["Coolx"]]    # Cooling at the energy bins between 0.3 and 8 keV

```

In [40]: *""" set the zero values to the minimum of non-zero value """*

```
Coolx[Coolx==0.] = np.min(Coolx[np.nonzero(Coolx)])      # set the zero values to be minimum
```

In [41]: `f = open('../data/workfile','r')`

```

s = f.read()

print s

f.close()

```

This is a test
and another test

In [42]: `f = open('../data/workfile','r')`

```

for line in f:
    print line

f.close()

```

This is a test
and another test

```
In [43]: import glob  
  
fnames = glob.glob('*.ipynb')  
  
print fnames
```

```
['astrounit.ipynb', 'matplotlib4.ipynb', 'Astropy1.ipynb', 'img2data.ipynb', 'matplotlib2.ipynb']
```

```
In [44]: import pickle  
  
l = [1,2,3,4,5.,6]  
pickle.dump(l,file('../data/test.pkl','w'))  
  
pickle.load(file('../data/test.pkl'))  
print l  
  
[1, 2, 3, 4, 5.0, 6]
```

1 Using IDL script in python

- requires to install pIDLy package

```
In [45]: import pidly # for using idl script in python  
  
idl=pidly.IDL('/home/astrodoo/idl/idl/bin/bin.linux.x86_64/idl')  
idl('read_inp','../data/zmp_inp',/python,ggen1=ggen1,ggen2=ggen2,eqos=eqos,unit=unit')  
rmin, rmax, x1rat, Nr = idl.ggen1["x1min"], idl.ggen1["x1max"], idl.ggen1["x1rat"], idl.ggen1["Nr"]  
thmin, thmax, Nth = idl.ggen2["x2min"], idl.ggen2["x2max"], idl.ggen2["nbl"]  
mmw = idl.eqos["mmw"]  
gamma = idl.eqos["gamma"]  
idl.close()  
  
print mmw,gamma,rmin,rmax  
  
% Compiled module: READ_INP.  
% Compiled module: STRSPLIT.  
1.21 1.66666674614 0.0025 250.0
```

2 Object-oriented Programming (CLASS)

- to organize the code
- to re-use code in similar contexts

```
In [46]: class Student(object):  
    def __init__(self, name):
```

```

        self.name = name
    def set_age(self, age):
        self.age = age
    def set_major(self, major):
        self.major = major

anna = Student('anna')
anna.set_age(21)
anna.set_major('astronomy')

print anna.name, anna.age, anna.major

anna 21 astronomy

```

In [47]: *"[”] inherit the methods & attributes from previous class and add an additional attribute "[”]"*

```

class MasterStudent(Student):
    internship = 'mandatory, from March to June'

james = MasterStudent('james')
print james.internship
james.set_age(23)
print james.age

mandatory, from March to June
23

```

In [48]: *"[”] Callable Class "[”]"*

```

class Factorial:
    def __init__(self):
        self.cache = {}
    def __call__(self, n):
        if n not in self.cache:
            if n == 0:
                self.cache[n] = 1
            else:
                self.cache[n] = n * self.__call__(n-1)
        return self.cache[n]

fact = Factorial()

for i in xrange(10):
    print("{}! = {}".format(i, fact(i)))

0! = 1
1! = 1

```

```
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
```

2.0.1 exec (execute) or eval

```
In [49]: def func():
    exec('print "hi from test2"',{})
def subfunction():
    return None

func()
```

```
hi from test2
```

```
In [50]: def func(var):
    a = ['aa','bb']
    aa = [1,2,3]
    bb = [3,5,6]

    print eval(a[var])

    return None

print func(1)
```

```
[3, 5, 6]
None
```

Basic2

January 30, 2019

0.1 Named Tuple

- create structure & substructure

```
In [1]: """ basic usage """
from collections import namedtuple

Point = namedtuple('Point', 'x,y')

a = Point(11,22)
b = Point(x=12,y=34)

print a[0],a[1]
print b[0],b[1]
print a.x, b.y
print a._fields

11 22
12 34
11 34
('x', 'y')
```

```
In [2]: """ replace """
from collections import namedtuple

Point = namedtuple('Point', 'x,y')

a = Point(x=11,y=22)
b = a._replace(x=33)

print b.x,b.y

33 22
```

```
In [3]: """ transform to dictionary """
from collections import namedtuple
```

```

Point = namedtuple('Point','x,y')
a = Point(x=11,y=22)
b = a._asdict()
print b['x'],b['y']

11 22

In [4]: """ Substructures """
from collections import namedtuple

Struc1 = namedtuple('Struc1','a,b,c')
Struc2 = namedtuple('Struc2','d,e')

aa = Struc1(a=1,b='hey',c=Struc2(d=23.,e=1))

print aa.a, aa.b, aa.c.d, aa.c.e

1 hey 23.0 1

In [5]: """ alternative way: substructures by class """
class mysubstruc:
    a = 2
    b = 'hey'
    c = 3.
    class kk:
        d = 5.
        e = 12.

    xx = mysubstruc()
    print xx.a, xx.b, xx.c, xx.kk, xx.kk.d, xx.kk.e

2 hey 3.0 __main__.kk 5.0 12.0

```

0.2 Ordered Dictionary

```

In [6]: from collections import OrderedDict

# regular unsorted dictionary
a = {'banana': 3, 'apple': 4, 'pear': 1, 'orange': 2}

# dictionary sorted by key
b = OrderedDict(sorted(a.items(), key=lambda t: t[0]))

```

```

# dictionary sorted by value
c = OrderedDict(sorted(a.items(), key=lambda t: t[1]))

# dictionary sorted by length of the key string
d = OrderedDict(sorted(a.items(), key=lambda t: len(t[0])))

print a
print b
print c
print d

print a.keys()[0],a.values()[0]
print b.keys()[0],b.values()[0]
print c.keys()[0],c.values()[0]
print d.keys()[0],d.values()[0]

{'orange': 2, 'pear': 1, 'banana': 3, 'apple': 4}
OrderedDict([('apple', 4), ('banana', 3), ('orange', 2), ('pear', 1)])
OrderedDict([('pear', 1), ('orange', 2), ('banana', 3), ('apple', 4)])
OrderedDict([('pear', 1), ('apple', 4), ('orange', 2), ('banana', 3)])
orange 2
apple 4
pear 1
pear 1

```

0.3 Mutable & Immutable Class

<http://www.blog.pythonlibrary.org/2014/01/17/how-to-create-immutable-classes-in-python/>

In [1]: *""" Mutable example (default of class) """*

```

class Mutable(object):
    """
    A mutable class
    """

    #-----
    def __init__(self):
        """Constructor"""
        pass

```

In [2]: mut_obj = Mutable()
mut_obj.monkey = "tamarin"
print mut_obj.monkey

tamarin

```
In [138]: class Immutable(object):
```

```
    """
    An immutable class
    """
    __slots__ = ["one", "two", "three"]

    #-----
    def __init__(self, one, two, three):
        """Constructor"""
        super(Immutable, self).__setattr__("one", one)
        super(Immutable, self).__setattr__("two", two)
        super(Immutable, self).__setattr__("three", three)

    #-----
    def __setattr__(self, name, value):
        """
        msg = "'%s' has no attribute %s" % (self.__class__,
                                             name)
        raise AttributeError(msg)
```

```
In [71]: i = Immutable(1,2,3)
```

```
#i.one = 4
#i.four = 3
#print i.one
```

```
In [74]: # 'object' should be written (meaning of no inheritance.)
```

```
# Following example shows mutable class but not allow to have additional attributes o
class Immutable(object):
```

```
    """
    mutable class but only in slots
    """
    __slots__ = ("one", "two", "three")
```

```
#     aa = 3
```

```
    #-----
    def __init__(self, one, two, three):
        """Constructor"""
        self.one = one
        self.two = two
        self.three = three
    #     self.bb = 3
    #self.four = 21
```

```
In [75]: i = Immutable(1,2,3)
```

```
#print i.cc
#i.one = 43
```

```
print i.one
#print i.four
```

1

```
In [129]: class hey(object):
    __slots__ = ('x',)
    #__statics__ = {'x', 'y'}

    def __init__(self):
        #self.__class__.y = 1
        self.x = 2
        #self.z = 3
        #self.x = 1
        #pass

a = hey()

hey().x = 3
#a.x = 2

print a.x
#a.x = 2
#hey.y = 2
#print a.__dict__
#print a.__class__.__dict__
```

2

In [185]: *"Class attribute and instance attribute"*

```
class test(object):
    c=2
    __slots__ = ('a',)

    def __init__(self):
        self.a = 20
        #@classmethod
        @staticmethod
    def aa():
        print 'hey'
        pass

test.b = 3
test.aa()
print test.b, test.c
inst = test()
```

```
print inst.a, inst.c
#inst.b = 10
#print inst.b
```

```
hey
3 2
20 2
```

0.4 Class: classmethod vs. staticmethod

```
class Dummy(object):

    @classmethod
    def some_function(cls,*args,**kwargs):
        print cls
```

1 both of these will have exactly the same effect

```
Dummy.some_function()
Dummy().some_function()
=====
class MyClass(object):

    def some_instance_method(self, *args, **kwds):
        pass

    @classmethod
    def some_class_method(cls, *args, **kwds):
        pass

    @staticmethod
    def some_static_method(*args, **kwds):
        pass
```

- classmethod must have a reference to a class object as the first parameter, whereas staticmethod can have no parameters at all.

```
In [175]: class Date(object):
```

```
    def __init__(self, day=0, month=0, year=0):
        self.day = day
        self.month = month
        self.year = year

    @classmethod
    def from_string(cls, date_as_string):
        day, month, year = map(int, date_as_string.split('-'))
```

```

        date1 = cls(day, month, year)
        return date1

    date2 = Date.from_string('11-09-2012')
    print date2.day, date2.month, date2.year

```

11 9 2012

In [176]: `class Date(object):`

```

    def __init__(self, day=0, month=0, year=0):
        self.day = day
        self.month = month
        self.year = year

    @staticmethod
    def is_date_valid(date_as_string):
        day, month, year = map(int, date_as_string.split('-'))
        return day <= 31 and month <= 12 and year <= 3999

    # usage:
    is_date = Date.is_date_valid('11-09-2012')
    print is_date

```

True

1.1 Intrinsic Functions

1.2 zip()

- combine two (same size; n-elements) list to produce n-elements list, at which each element has tuple of two data points

In [14]: *Basic example of zip function*

```

x = [1, 2, 3]
y = [4, 5, 6]
zipped = zip(x, y)
print zipped
print len(zipped)
print zipped[0]

```

```

[(1, 4), (2, 5), (3, 6)]
3
(1, 4)

```

In [8]: *'*' can be used for unzip*

```
print zip(*zipped)
```

```
x2, y2 = zip(*zipped)
print ((x==list(x2)) & (y==list(y2)))

[(1, 2, 3), (4, 5, 6)]
True
```

In [9]: *"zip function with for-loop"*

```
alist = ['a1', 'a2', 'a3']
blist = ['b1', 'b2', 'b3']

for a, b in zip(alist, blist):
    print a, b

a1 b1
a2 b2
a3 b3
```

In [10]: *"zip function with enumerated for-loop"*

```
alist = ['a1', 'a2', 'a3']
blist = ['b1', 'b2', 'b3']

for i, (a, b) in enumerate(zip(alist, blist)):
    print i, a, b

0 a1 b1
1 a2 b2
2 a3 b3
```

1.3 map()

- map(some_function, some_iterable) --> return the list

```
In [3]: def func(x):
        return x+10

xx = range(5)

yy = map(func,xx)
print xx, yy

# such map function is equivalent to list-comprehensive

yy2 = [func(x) for x in xx]
print xx, yy2
```

```
[0, 1, 2, 3, 4] [10, 11, 12, 13, 14]  
[0, 1, 2, 3, 4] [10, 11, 12, 13, 14]
```

```
In [6]: def func(a,b):  
    return a+b  
  
    aa = range(5)  
    bb = range(5)  
  
    yy = map(func,aa,bb)  
    print aa,bb,yy  
  
    # such map function is equivalent to list-comprehensive  
  
    yy2 = [func(a,b) for a,b in zip(aa,bb)]  
    print aa,bb,yy2
```

```
[0, 1, 2, 3, 4] [0, 1, 2, 3, 4] [0, 2, 4, 6, 8]  
[0, 1, 2, 3, 4] [0, 1, 2, 3, 4] [0, 2, 4, 6, 8]
```

```
In [8]: """ useful implication with lambda function"""  
a = [1, 2, 3, 4, 5]  
b = [2, 2, 9, 0, 9]  
  
c = map(lambda pair: max(pair), zip(a, b))  
print c
```

```
[2, 2, 9, 4, 9]
```

1.4 lambda()

- one-line mini-function

```
In [11]: g = lambda x: x*2  
  
        print g(3)  
  
        # it is equivalent to  
        def g2(x):  
            return x*2  
  
        print g2(3)
```

6
6

In [12]: *"multiple inputs"*

```
f = lambda x,y: x+y
```

```
print f(1,2)
```

3

dummy

January 30, 2019

```
In [40]: import matplotlib.colors as cls
         import numpy as np

def mimicAlpha_to_rgb(color,alpha=0.5,bg='w'):

    if type(color).__name__ == 'str':
        clr_rgb = cls.to_rgb(color)
    elif ((type(color).__name__ != 'tuple') | (len(color) != 3)):
        raise ValueError('color should be in string or (r,g,b) tuple')
    else:
        clr_rgb = color

    if type(bg).__name__ == 'str':
        clr_bg = cls.to_rgb(bg)
    elif ((type(bg).__name__ != 'tuple') | (len(bg) != 3)):
        raise ValueError('bg should be in string or (r,g,b) tuple')
    else:
        clr_bg = bg

    clr_rgb = np.asarray(clr_rgb,dtype='float')
    clr_bg = np.asarray(clr_bg,dtype='float')

    clrA_rgb = (1.-alpha)*clr_bg + alpha*clr_rgb
    return tuple(clrA_rgb)
```

```
In [8]: import matplotlib.colors as cls

wh = np.array(cls.to_rgb('white'),dtype='float')
bl = np.array(cls.to_rgb('blue'),dtype='float')
print wh,bl
```

```
[ 1.  1.  1.] [ 0.  0.  1.]
```

```
In [9]: print wh

[ 1.  1.  1.]
```

```
In [10]: print tuple(wh)
```

```
(1.0, 1.0, 1.0)
```

```
In [45]: print mimicAlpha_to_rgb((1,2,3),alpha=0.5,bg=(1,1,1,1))
```

```
ValueError
```

```
Traceback (most recent call last)
```

```
<ipython-input-45-2cbddd918b8a> in <module>()
```

```
----> 1 print mimicAlpha_to_rgb((1,2,3),alpha=0.5,bg=(1,1,1,1))
```

```
<ipython-input-40-8e20163eec61> in mimicAlpha_to_rgb(color, alpha, bg)
```

```
14     clr_bg = cls.to_rgb(bg)
```

```
15     elif ((type(bg).__name__ != 'tuple') | (len(bg) != 3)):
```

```
---> 16         raise ValueError('bg should be in string or (r,g,b) tuple')
```

```
17     else:
```

```
18         clr_bg = bg
```

```
ValueError: bg should be in string or (r,g,b) tuple
```

```
In [25]: aa = cls.to_rgb('blue')
```

```
In [22]: print type(aa).__name__
```

```
tuple
```

```
In [23]: aa = 'k'
```

```
print type(aa).__name__
```

```
str
```

```
In [26]: print len(aa)
```

```
3
```

```
In [28]: raise ValueError('Hey')
```

```
ValueError                                     Traceback (most recent call last)

<ipython-input-28-8d17a827f5ae> in <module>()
----> 1 raise ValueError('Hey')

ValueError: Hey

In [1]: from scipy import ndimage
       import numpy as np

a = np.arange(12.).reshape((4, 3))

print a

[[ 0.   1.   2.]
 [ 3.   4.   5.]
 [ 6.   7.   8.]
 [ 9.  10.  11.]]

In [21]: inds = np.array([[1.5, 2], [0, 4]])
         print ndimage.map_coordinates(a, inds, order=1, mode='nearest')

[ 4.5  8. ]
```

img2data

January 30, 2019

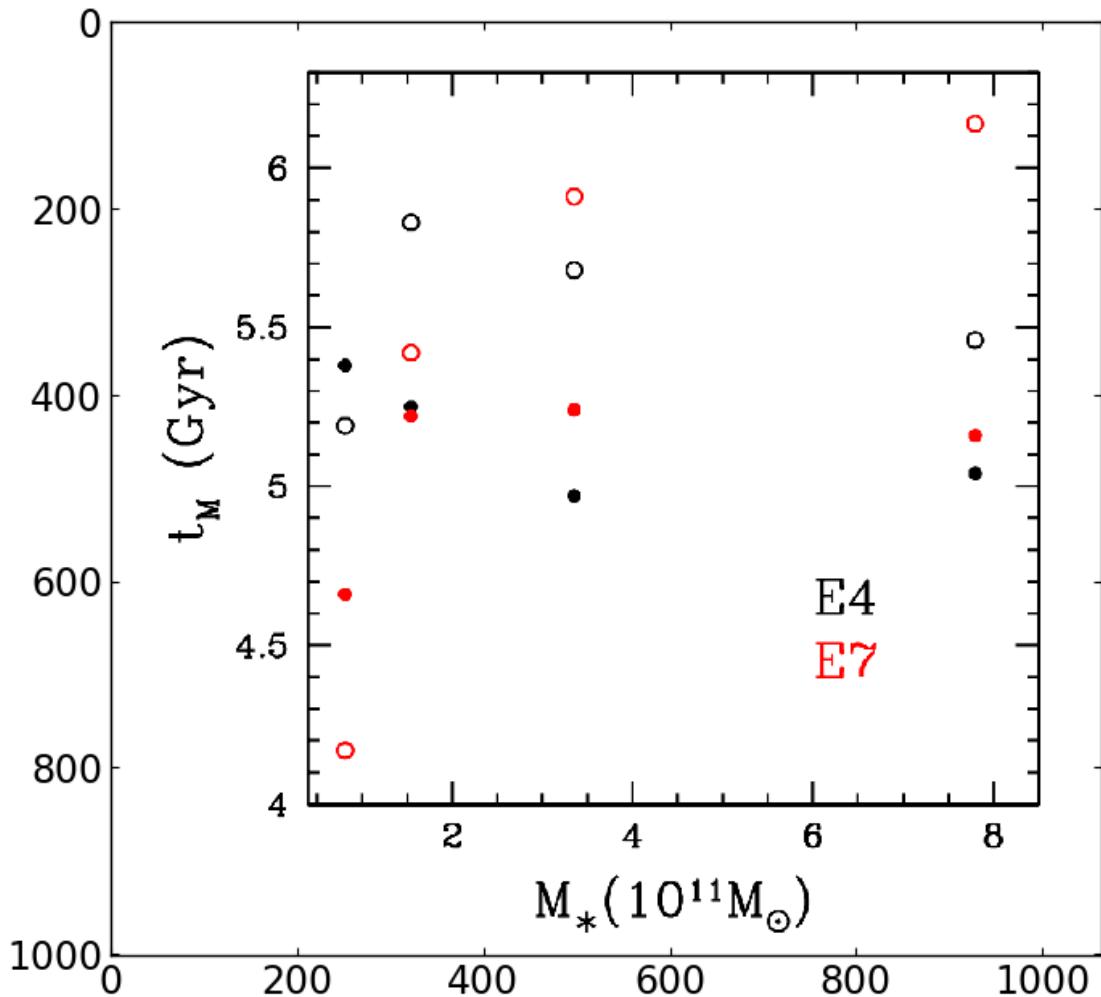
1 pyds.img2data

NOTE that magic function should be set to the one that allows interactive plotting such as '%matplotlib notebook'

- First, you should click four boundary points. --> click first two points for setting minimum x & maximum x and next two points for setting minimum y & maximum y.
- Second, now you can click data points repeatedly.
- Third, when you finished clicking all data point, then you should finalize it with clicking 'Stop Interaction' Mark.
- All data points will be saved in self.xdata & self.ydata in the type of 'list'.

1.1 Linear scale plot

```
In [1]: """ Let's prepare the figure in any readable format (e.g. png, jpg ...),  
        and check the figure  
        This is not interactive mode yet.  
"""  
  
import matplotlib.image as mpimg  
import matplotlib.pyplot as plt  
#import os  
%matplotlib inline  
  
#pyfig = os.environ['PYTHONDATA']  
pyfig = './example_data/'  
img = mpimg.imread(pyfig+'testlinear.png')  
  
## note that img.shape[0] represents y-axis size  
fig, ax = plt.subplots(figsize=(8,8.*img.shape[0]/img.shape[1]))  
  
imgplot = ax.imshow(img)
```



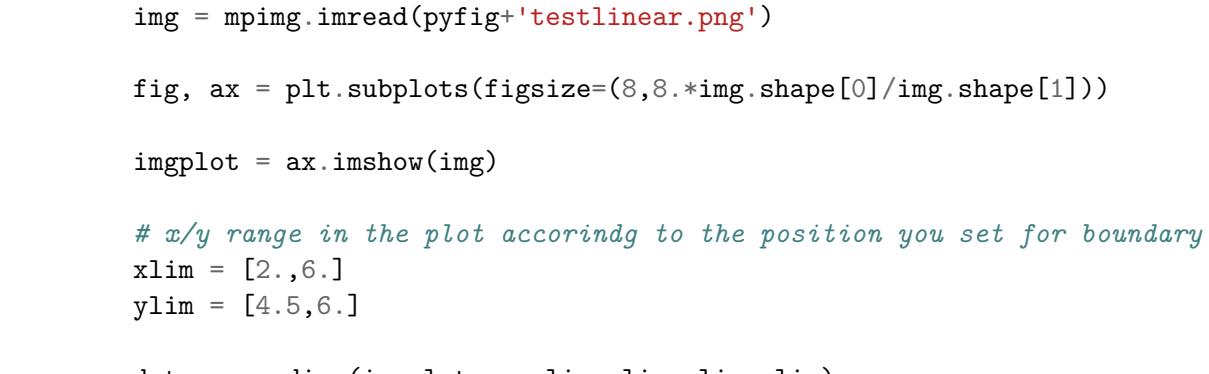
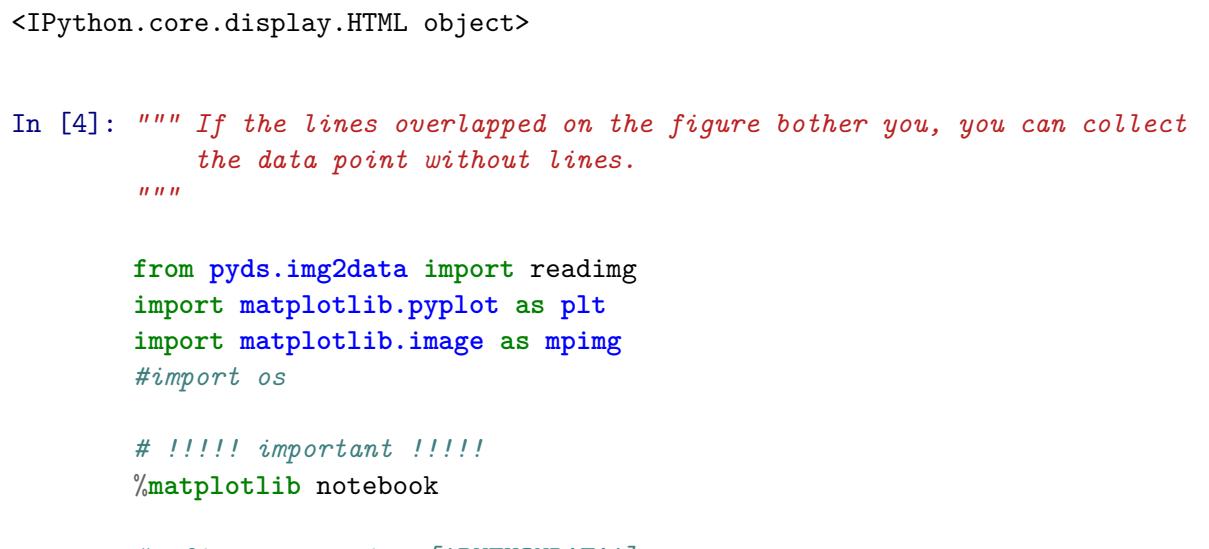
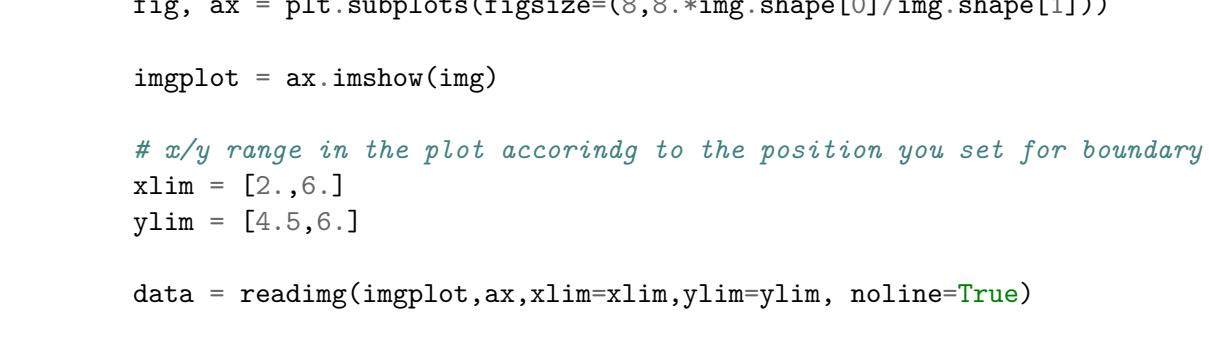
In [3]: """ Now, it is interactive mode and you can collect all data point by clicking mouse. Sometimes, transitioning from '%matplotlib inline' to '%matplotlib notebook' seems not work smoothly.
In this case, please just re-run this one (or two) more time, then it would work.
"""

```
from pyds.img2data import readimg
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
#import os

# !!!!! important !!!!!
%matplotlib notebook

#pyfig = os.environ['PYTHONDATA']
pyfig = './example_data/'
```

```

```

img = mpimg.imread(pyfig+'testlinear.png')

fig, ax = plt.subplots(figsize=(8,8.*img.shape[0]/img.shape[1]))

imgplot = ax.imshow(img)

x/y range in the plot according to the position you set for boundary

xlim = [2.,6.]

ylim = [4.5,6.]

data = readimg(imgplot,ax,xlim=xlim,ylim=ylim)

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

In [4]: *""" If the lines overlapped on the figure bother you, you can collect the data point without lines.*

"""

```

from pyds.img2data import readimg
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
#import os

# !!!!! important !!!!!
%matplotlib notebook

#pyfig = os.environ['PYTHONDATA']
pyfig = './example_data/'
img = mpimg.imread(pyfig+'testlinear.png')

fig, ax = plt.subplots(figsize=(8,8.*img.shape[0]/img.shape[1]))

imgplot = ax.imshow(img)

# x/y range in the plot according to the position you set for boundary
xlim = [2.,6.]
ylim = [4.5,6.]

data = readimg(imgplot,ax,xlim=xlim,ylim=ylim, noline=True)

```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

In [5]: *"'"' printing the data points you collected "'"''*

```
print data.xdata,data.ydata
```

```
[0.82142857142857206, 0.82142857142857206, 0.82142857142857206, 0.83928571428571463, 1.553571428571463]
```

In [6]: *"'"' check the data points with plot "'"''*

```
import matplotlib.pyplot as plt  
%matplotlib inline
```

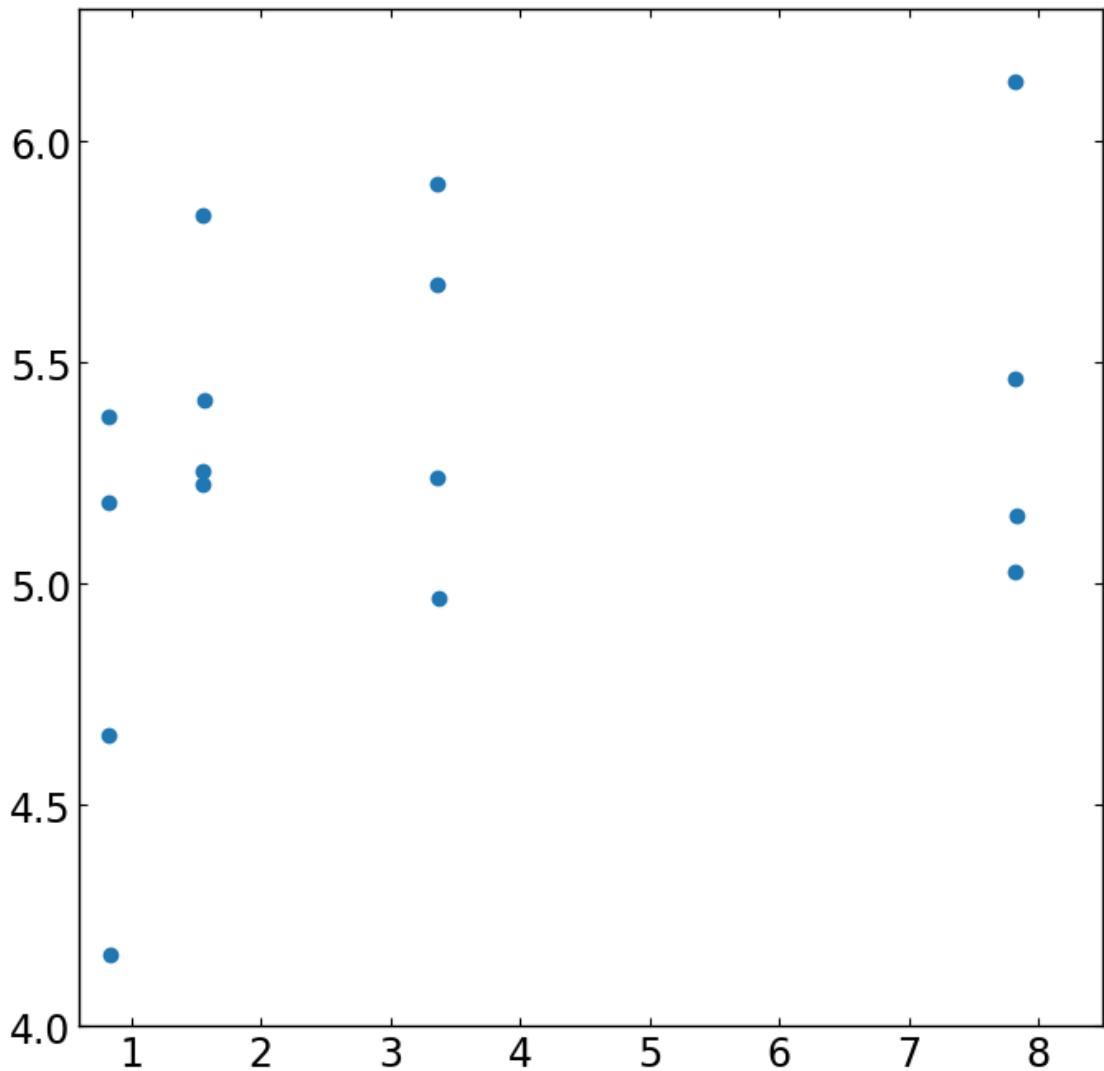
```
fig,ax = plt.subplots(figsize=(8,8))
```

```
ax.plot(data.xdata,data.ydata,'o')
```

```
ax.set_xlim([0.6,8.5])
```

```
ax.set_ylim([4.,6.3])
```

Out[6]: (4.0, 6.3)



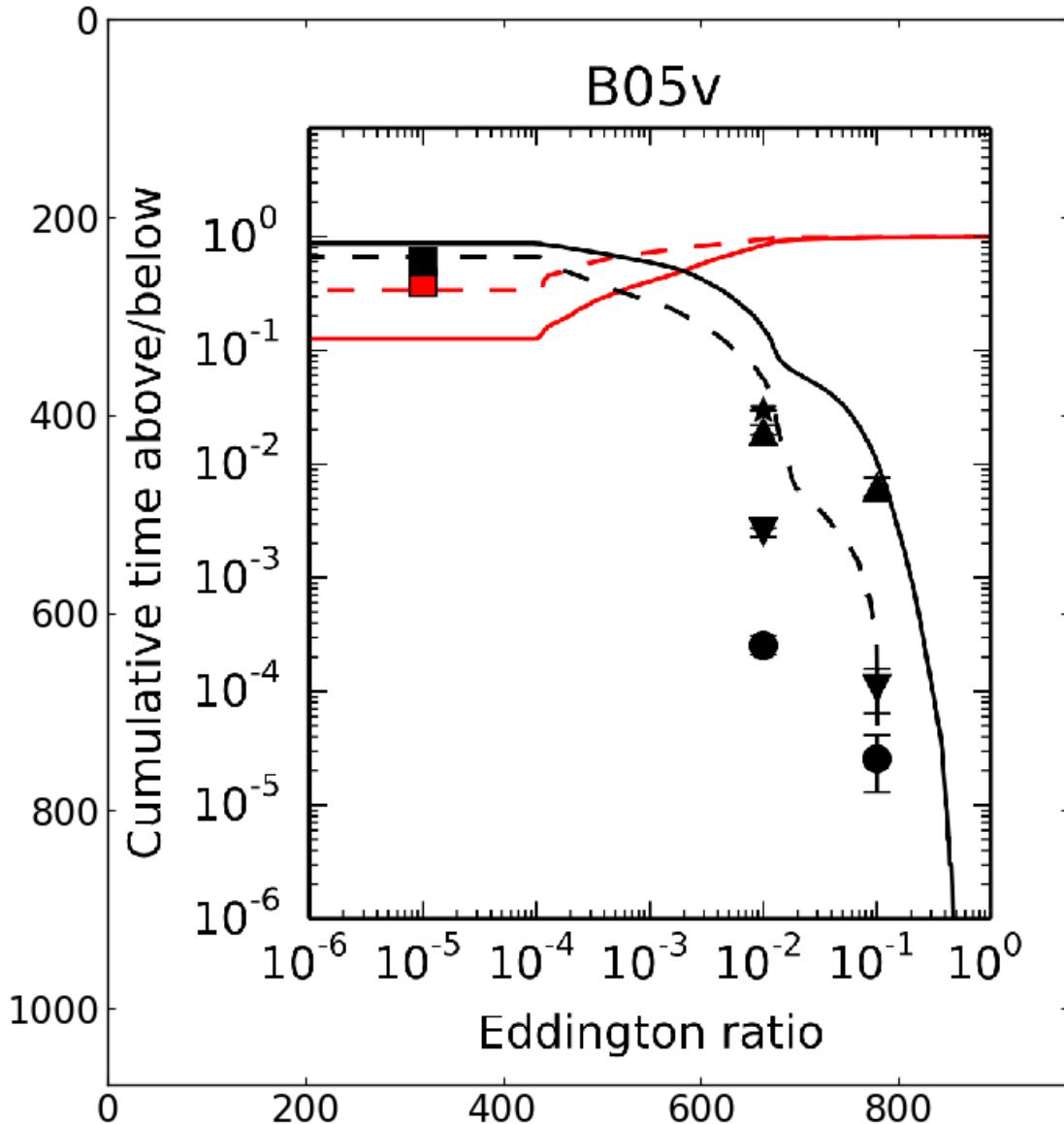
1.2 Log scale plot

In [7]: *""" Let's prepare the figure in any readable format (e.g. png, jpg ..), and check the figure. This is not interactive mode yet. This example is for the figure in logarithmic scale. """*

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
# import os
%matplotlib inline

#pyfig = os.environ['PYTHONDATA']
pyfig = './example_data/'
img = mpimg.imread(pyfig+'testlog.png')
```

```
fig, ax = plt.subplots(figsize=(8,8.*img.shape[0]/img.shape[1]))
imgplot = ax.imshow(img)
```



In [9]: """ Now, it is interactive mode and you can collect all data point by clicking mouse. Sometimes, transitioning from '%matplotlib inline' to '%matplotlib notebook' seems not work smoothly. In this case, please just run this one (or two) more time then it would work.

```
"""
```

```
from pyds.img2data import reading
```

```

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
#import os
# !!!!! important !!!!!
%matplotlib notebook

#pyfig = os.environ['PYTHONDATA']
pyfig = './example_data/'
img = mpimg.imread(pyfig+'testlog.png')

fig, ax = plt.subplots(figsize=(8,8.*img.shape[0]/img.shape[1]))
imgplot = ax.imshow(img)

# x/y range in the plot according to the position you set for boundary
xlim = [1e-6,1.]
ylim = [1e-6,1.]

data = readimg(imgplot,ax,xlim=xlim,ylim=ylim,xlog=True,ylog=True)

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

```

In [10]: *"Printing the data points you collected"*

```

print data.xdata,data.ydata

[1.0000000000000021e-05, 1.0000000000000021e-05, 0.0096895000626137026, 0.0096895000626137026,

```

In [11]: *"Check the data points with plot"*

```

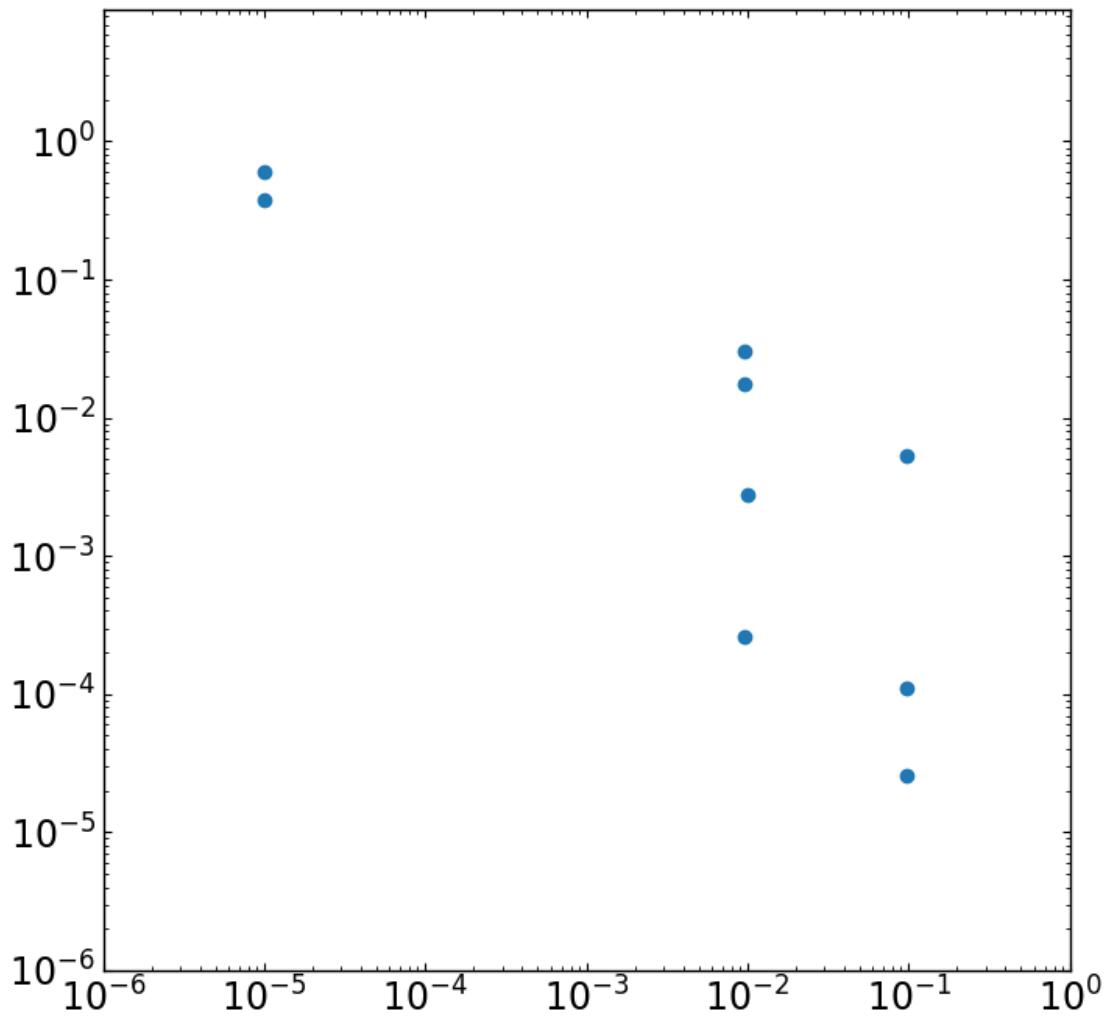
import matplotlib.pyplot as plt
%matplotlib inline

fig,ax = plt.subplots(figsize=(8,8))

ax.loglog(data.xdata,data.ydata,'o')
ax.set_xlim([1e-6,1])
ax.set_ylim([1e-6,9.])

```

Out[11]: (1e-06, 9.0)



Interactive Plot

January 30, 2019

```
In [ ]: import ipywidgets as widgets
         from IPython.display import display
         import matplotlib.pyplot as plt
         import matplotlib.image as mpimg
         import numpy as np
         %matplotlib notebook

         xx=[]; yy=[]
         x0=[]; y0=[]

         img = mpimg.imread('/home/astrodoo/Work/Sketch/test.png')
         fig, ax = plt.subplots()
         imgplot = ax.imshow(img)

         w = widgets.HTML()

         i=0
         def onclick(event):
             global ax,i,xx,yy
             w.value = 'xdata=%f, ydata=%f'%(event.xdata, event.ydata)

             if i<2:
                 ax.plot(event.xdata,event.ydata, 'o')
                 x0.append(event.xdata); y0.append(event.ydata)
             else:
                 xx.append(event.xdata); yy.append(event.ydata)
                 ax.axhline(y=event.ydata)
                 ax.axvline(x=event.xdata)

             i=i+1

         cid = fig.canvas.mpl_connect('button_press_event', onclick)
         display(w)

In [ ]: print xx

In [ ]: import ipywidgets as widgets
         from IPython.display import display
```

```

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
%matplotlib notebook

class readpos:
    def __init__(self, imgplot, ax, w):
        self.imgplot = imgplot
        self.ax = ax
        self.w = w
        self.x0 = []; self.y0 = []
        self.xx = []; self.yy = []
        self.ind = 0
        self.cid = imgplot.figure.canvas.mpl_connect('button_press_event', self)
        display(self.w)

    def __call__(self, event):
        self.w.value = 'xdata=%f, ydata=%f'%(event.xdata, event.ydata)
        if self.ind < 2:
            self.ax.plot(event.xdata, event.ydata, 'o')
            self.x0.append(event.xdata); self.y0.append(event.ydata)
        else:
            self.ax.axhline(y=event.ydata)
            self.ax.axvline(x=event.xdata)
            self.xx.append(event.xdata); self.yy.append(event.ydata)

        self.ind = self.ind + 1

In [ ]: img = mpimg.imread('/home/astrodoo/Work/Sketch/test.png')
fig, ax = plt.subplots()
imgplot = ax.imshow(img)
w = widgets.HTML()

pos = readpos(imgplot, ax, w)

In [ ]: import ipywidgets as widgets
from IPython.display import display
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
%matplotlib notebook

wflag = True
class read_pos:
    def __init__(self, imgplot, ax, *args):
        global wflag

```

```

        self.imgplot = imgplot
        self.ax = ax
        if len(args)>=1:
            self.w = args[0]
        else:
            wflag = False
        self.x0 = []; self.y0 = []
        self.xx = []; self.yy = []
        self.xdata = []; self.ydata = []
        self.ind = 0
        self.cid = imgplot.figure.canvas.mpl_connect('button_press_event', self)
        if wflag: display(self.w)

    def __call__(self, event):
        if wflag:
            self.w.value = 'xdata=%f, ydata=%f'%(event.xdata, event.ydata)
        if self.ind < 2:
            self.ax.plot(event.xdata,event.ydata,'o')
            self.x0.append(event.xdata); self.y0.append(event.ydata)
            if wflag:
                self.w.value = 'xdata=%f, ydata=%f'%(event.xdata, event.ydata)
        else:
            self.ax.axhline(y=event.ydata)
            self.ax.axvline(x=event.xdata)
            self.xx.append(event.xdata); self.yy.append(event.ydata)

            xinplt = self.calcdatay(event.xdata)
            self.xdata.append(xinplt)

            yinplt = self.calcdatay(event.ydata)
            self.ydata.append(yinplt)

            if wflag:
                self.w.value = 'xpixel=%f, ypixel=%f ,xdata=%f, ydata=%f'%(event.xdata,
                    event.ydata)

        self.ind = self.ind + 1

    def calcdatay(self, ypos):
        xdat = (ypos-self.y0[0])/ (self.y0[1]-self.y0[0])*(xlim[1]-xlim[0])+xlim[0]
        return xdat
    def calcdatay(self, ypos):
        ydat = (ypos-self.y0[0])/ (self.y0[1]-self.y0[0])*(ylim[1]-ylim[0])+ylim[0]
        return ydat

#def img2data(imgplot, ax, w, xlim=[], ylim=[]):

```

```

def readpos(*args, **keywords):
    global xlim, ylim

    imgplot = args[0]; ax = args[1]
    if len(args) == 3:
        w = args[2]
        pos=read_pos(imgplot,ax,w)
    elif len(args) == 2:
        pos=read_pos(imgplot,ax)
    else:
        print 'Number of Args should be 2 or 3.'
        return None

    if 'xlim' in keywords.keys():
        xlim = keywords['xlim']
    else:
        print 'xlim should be entered.'
        return None
    if 'ylim' in keywords.keys():
        ylim = keywords['ylim']
    else:
        print 'ylim should be entered.'
        return None

    x0pos = pos.x0; y0pos = pos.y0
    xxpos = pos.xx; yypos = pos.yy

    a = [1.,2.,3.]
    b = np.array(xxpos,dtype=float)

    xx = np.asarray(xxpos,dtype=float)
    #x0 = np.asarray(x0pos, dtype=float); y0 = np.asarray(y0pos, dtype=float)
    #xx = np.asarray(xxpos, dtype=float); yy = np.asarray(yypos, dtype=float)

    return pos.xdata

#def img2data(*args, **keywords):
#    print len(args)
#    imgplot = args[0]; ax = args[1]
#    if len(args) == 3:
#        w = args[2]
#        pos = readpos(imgplot,ax,w)
#    elif len(args) == 2:
#        pos = readpos(imgplot,ax)
#    else:
#        print 'Number of Args should be 2 or 3.'

```

```

#           return None

#     if 'xlim' in keywords.keys():
#         xlim = keywords['xlim']
#     else:
#         print 'xlim should be entered.'
#         return None
#     if 'ylim' in keywords.keys():
#         ylim = keywords['ylim']
#     else:
#         print 'ylim should be entered.'
#         return None

#     x0 = np.array(pos.xx, dtype=float); y0 = np.array(pos.yy, dtype=float)
#     xx = np.array(pos.xx, dtype=float); yy = np.array(pos.yy, dtype=float)

#     xdat = (xx-x0[0])/(x0[1]-x0[0]) * (xlim[1]-xlim[0]) + xlim[0]
#     ydat = (yy-y0[0])/(y0[1]-y0[0]) * (ylim[1]-ylim[0]) + ylim[0]

#     return (xdat,ydat)

```

```

In [ ]: import ipywidgets as widgets
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg

%matplotlib notebook

img = mpimg.imread('/home/astrodoo/Work/Sketch/test.png')
fig, ax = plt.subplots()
imgplot = ax.imshow(img)
w = widgets.HTML()

#pos = readpos(imgplot,ax,w)
#pos = readpos(imgplot,ax)

xxx = readpos(imgplot,ax,w,xlim=[1.,100.],ylim=[1.,100.])

```

```
In [ ]: print np.asarray(xxx, dtype=float)
```

```
In [ ]: xx = np.asarray(pos.xx, dtype=float)
```

```
In [ ]: print xx
```

```

In [1]: import ipywidgets as widgets
        from IPython.display import display
        import numpy as np

wflag = True
class readimg:

```

```

def __init__(self, *args, **keywords):
    global wflag

    if len(args) == 2:
        self.imgplot = args[0]; self.ax = args[1]
        wflag = False
    elif len(args) == 3:
        self.imgplot = args[0]; self.ax = args[1]
        self.w = args[2]
    else:
        print 'Number of Args should be 2 or 3.'
        return None

    if 'xlim' in keywords.keys():
        self.xlim = keywords['xlim']
    else:
        print 'xlim should be entered.'
        return None
    if 'ylim' in keywords.keys():
        self.ylim = keywords['ylim']
    else:
        print 'ylim should be entered.'
        return None

    if 'xlog' in keywords.keys():
        self.xlog = keywords['xlog']
    else:
        self.xlog = False
    if 'ylog' in keywords.keys():
        self.ylog = keywords['ylog']
    else:
        self.ylog = False

    self.x0 = []; self.y0 = []
    self.xx = []; self.yy = []
    self.xdata = []; self.ydata = []
    self.ind = 0
    self.cid = imgplot.figure.canvas.mpl_connect('button_press_event', self)
    if wflag: display(self.w)

def __call__(self, event):
    if wflag:
        self.w.value = 'xdata=%f, ydata=%f'%(event.xdata, event.ydata)
    if self.ind < 2:
        self.ax.plot(event.xdata, event.ydata, 'o')
        self.x0.append(event.xdata); self.y0.append(event.ydata)
        if wflag:

```

```

        self.w.value = 'xdata=%f, ydata=%f'%(event.xdata, event.ydata)
    else:
        self.ax.axhline(y=event.ydata)
        self.ax.axvline(x=event.xdata)
        self.xx.append(event.xdata); self.yy.append(event.ydata)

    if self.xlog:
        xinplt = self.calcdatadxlg(event.xdata)
    else:
        xinplt = self.calcdatadx(event.xdata)
    self.xdata.append(xinplt)

    if self.ylog:
        yinplt = self.calcdataylg(event.ydata)
    else:
        yinplt = self.calcdatay(event.ydata)
    self.ydata.append(yinplt)

    if wflag:
        self.w.value = 'xpixel=%f, ypixel=%f ,xdata=%f, ydata=%f'%(event.xdata,
                                                                     event.ydata)

    self.ind = self.ind + 1

def calcdatadx(self, xpos):
    xdat = (xpos-self.x0[0])/ (self.x0[1]-self.x0[0])*(self.xlim[1]-self.xlim[0])+self.xlim[0]
    return xdat

def calcdatay(self, ypos):
    ydat = (ypos-self.y0[0])/ (self.y0[1]-self.y0[0])*(self.ylim[1]-self.ylim[0])+self.ylim[0]
    return ydat

def calcdatadxlg(self, xpos):
    xlimlg = np.log10(self.xlim)
    xdat = np.power(10.,(xpos-self.x0[0])/ (self.x0[1]-self.x0[0])*(xlimlg[1]-xlimlg[0])+xlimlg[0])
    return xdat

def calcdataylg(self, ypos):
    ylimlg = np.log10(self.ylim)
    ydat = np.power(10.,(ypos-self.y0[0])/ (self.y0[1]-self.y0[0])*(ylimlg[1]-ylimlg[0])+ylimlg[0])
    return ydat

In [2]: import ipywidgets as widgets
       import matplotlib.pyplot as plt
       import matplotlib.image as mpimg
       %matplotlib notebook

img = mpimg.imread('/home/astrodoo/Work/Sketch/test.png')

```

```

fig, ax = plt.subplots()
imgplot = ax.imshow(img)
w = widgets.HTML()

#pos = readpos(imgplot,ax,w)
#pos = readpos(imgplot,ax)

xxx = readimg(imgplot,ax,w,xlim=[1e-6,1.],ylim=[1e-6,9.],xlog=True,ylog=True)

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

In [1]: -
In [1]: -
In [4]: print xxx.ydata
[0.72806284921665443, 0.017987307436770412, 0.00012942828798413126]

In [ ]: import matplotlib.pyplot as plt
        import numpy as np
        %matplotlib notebook

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(np.random.rand(10))

        def onclick(event):
            print('button=%d, x=%d, y=%d, xdata=%f, ydata=%f' %
                  (event.button, event.x, event.y, event.xdata, event.ydata))

        cid = fig.canvas.mpl_connect('button_press_event', onclick)

In [ ]: print np.shape(img)

In [ ]: from matplotlib import pyplot as plt

%pylab notebook

class LineBuilder:
    def __init__(self, line):
        self.line = line
        self.xs = list(line.get_xdata())
        self.ys = list(line.get_ydata())

```

```

        self.cid = line.figure.canvas.mpl_connect('button_press_event', self)

    def __call__(self, event):
        print('click', event)

        if event.inaxes!=self.line.axes: return
        self.xs.append(event.xdata)
        self.ys.append(event.ydata)

        self.line.set_data(self.xs, self.ys)
        self.line.figure.canvas.draw()

    fig = plt.figure()
    ax = fig.add_subplot(111)

    ax.set_title('click to build line segments')
    line, = ax.plot([0], [0]) # empty line
    linebuilder = LineBuilder(line)

In [ ]: print linebuilder.xs

In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib notebook

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.set_title('click on points')

        line, = ax.plot(np.random.rand(100), 'o', picker=5) # 5 points tolerance

    def onpick(event):
        thisline = event.artist
        xdata = thisline.get_xdata()
        ydata = thisline.get_ydata()
        ind = event.ind
        points = tuple(zip(xdata[ind], ydata[ind]))
#        print('onpick points:', points)

        fig.canvas.mpl_connect('pick_event', onpick)

        plt.show()

In [ ]: %matplotlib notebook

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(np.random.rand(10))

```

```
def onclick(event):
    print('button=%d, x=%d, y=%d, xdata=%f, ydata=%f' %
          (event.button, event.x, event.y, event.xdata, event.ydata))

cid = fig.canvas.mpl_connect('button_press_event', onclick)

In [ ]: fig.canvas.mpl_disconnect(cid)

In [ ]: import numpy as np

def test(a=[1,2]):
    b = np.asarray(a)
    return b

c = test()

In [ ]: print c

In [ ]: class test:
    def __init__(self, *args):
        self.a=args[0]

    b = test('a')

In [ ]: print b.a

In [ ]: import numpy as np

a = [10.,100.]

b = np.log10(a)

print b[0]
```

matplotlib1

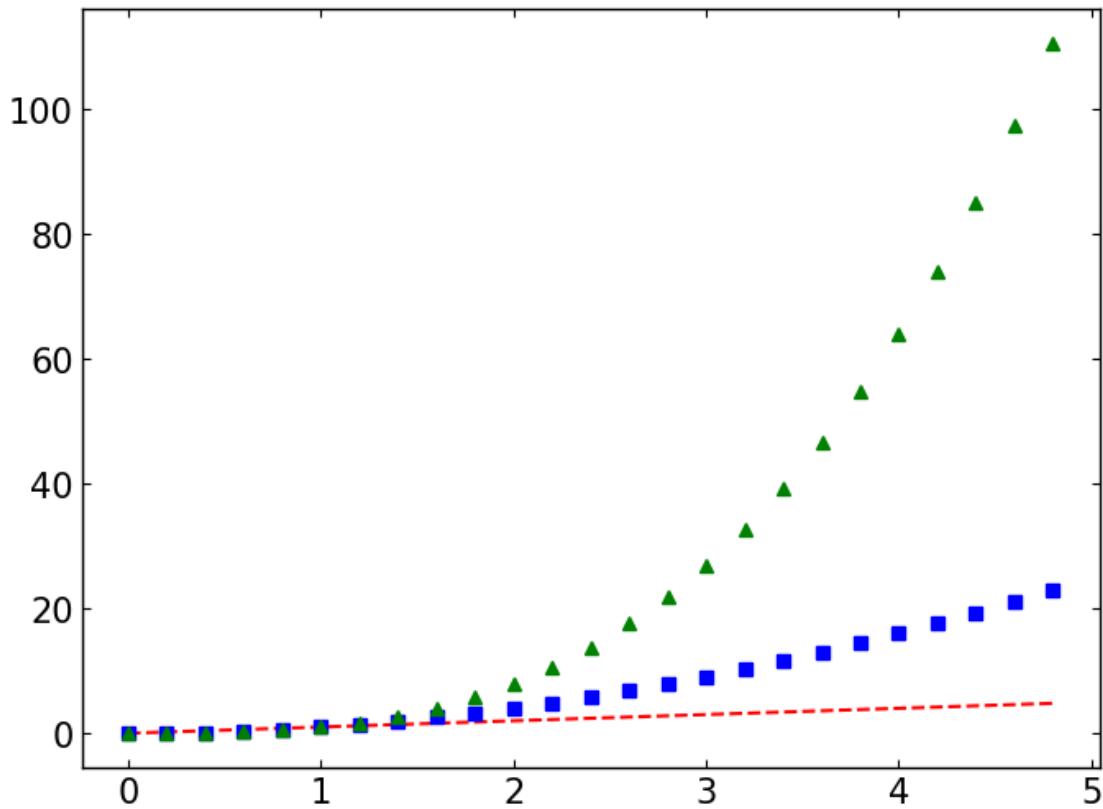
January 30, 2019

1 Basic Plotting

1.0.1 recover matplotlib defaults

```
import matplotlib as mpl  
mpl.rcParams.update(mpl.rcParamsDefault)  
%pylab inline
```

```
In [2]: # recover matplotlib defaults  
#import matplotlib as mpl  
#mpl.rcParams.update(mpl.rcParamsDefault)  
#%pylab inline  
  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
t = np.arange(0.,5.,0.2)  
plt.plot(t,t,'r--',t,t**2,'bs',t,t**3,'g^')  
#plt.show()  
#plt.savefig('tmp.eps')
```



In [12]: """ fontsize """

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

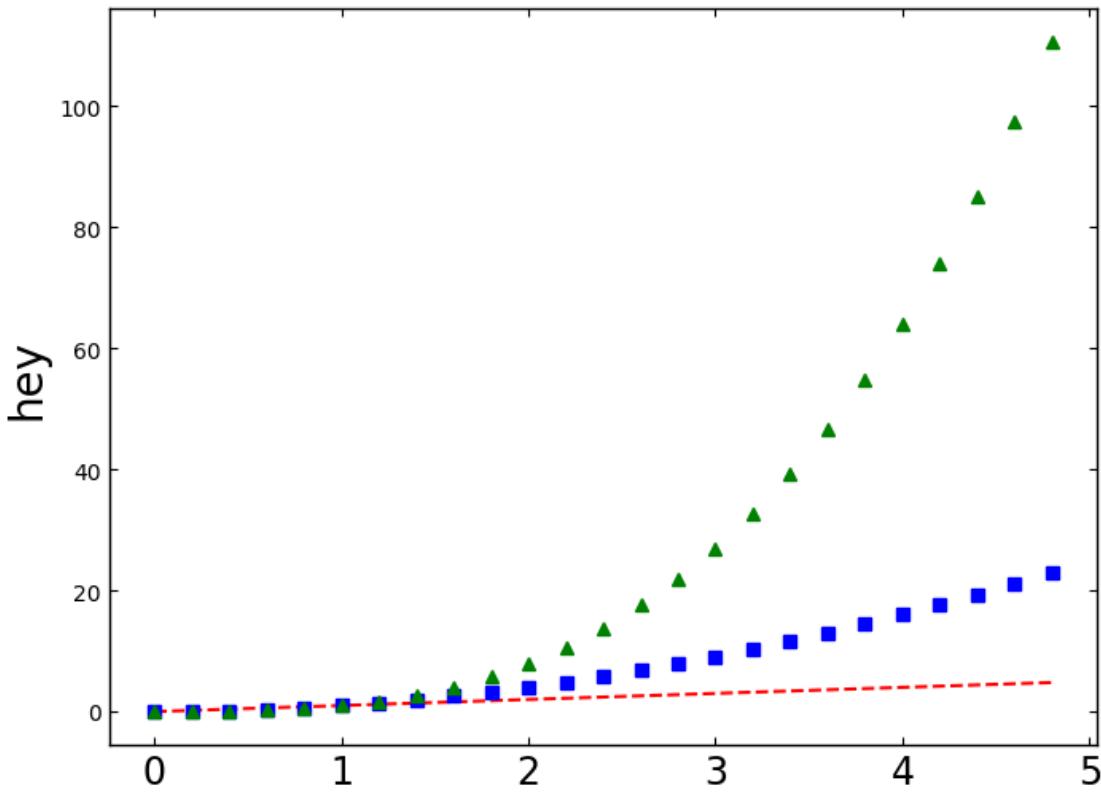
t = np.arange(0.,5.,0.2)

fig, ax = plt.subplots()

ax.plot(t,t,'r--',t,t**2,'bs',t,t**3,'g^')

plt.yticks(fontsize=10)
ax.set_ylabel('hey',fontsize=20)
```

Out[12]: <matplotlib.text.Text at 0x7ff6582b6ad0>



1.0.2 dark background

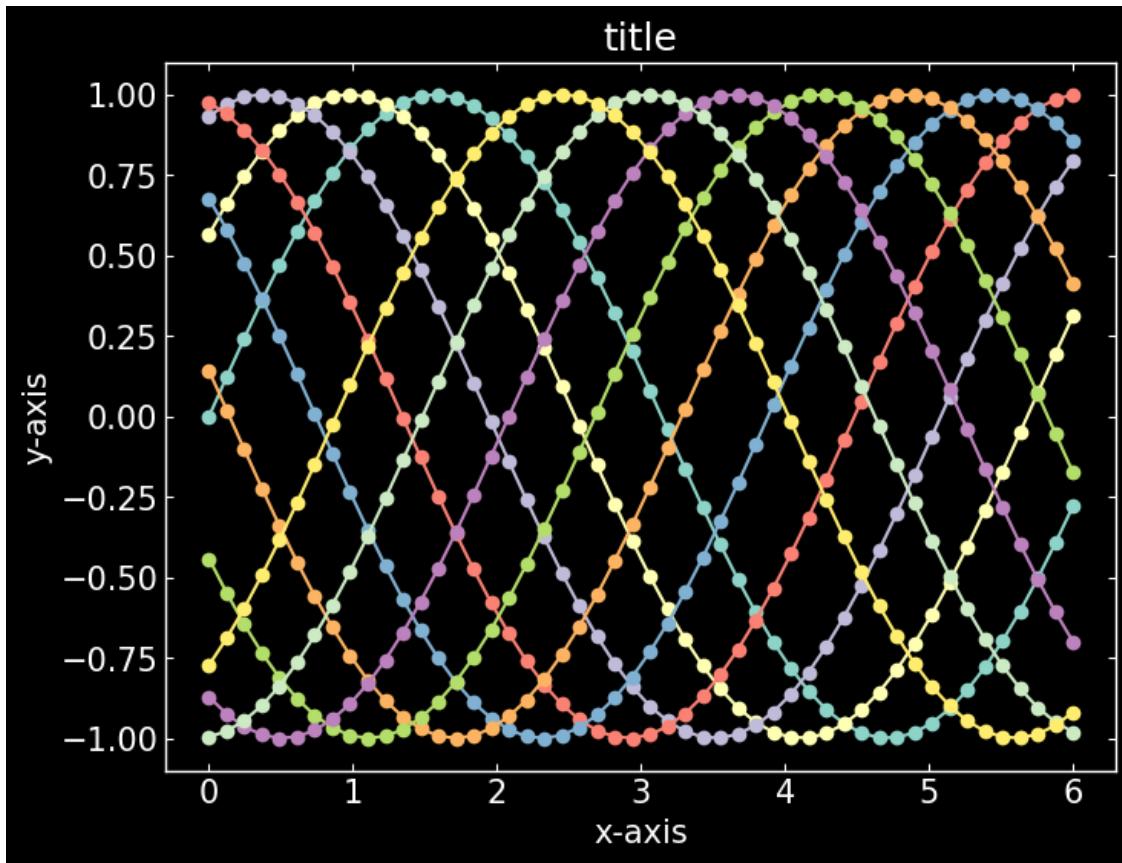
```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        plt.style.use('dark_background')

        L = 6
        x = np.linspace(0, L)
        ncolors = len(plt.rcParams['axes.color_cycle'])
        shift = np.linspace(0, L, ncolors, endpoint=False)
        for s in shift:
            plt.plot(x, np.sin(x + s), 'o-')
        plt.xlabel('x-axis')
        plt.ylabel('y-axis')
        plt.title('title')

        plt.show()

/home/astrodoo/anaconda2/lib/python2.7/site-packages/matplotlib/__init__.py:938: UserWarning: a
  warnings.warn(self.msg_depr % (key, alt_key))
```



1.0.3 save figures

Note that `matplotlib.use('Agg')` is not compatible with "%pylab inline". But without the option, we can save the figure as png, eps, pdf.

```
In [4]: """ recover matplotlib defaults """
import matplotlib as mpl
#mpl.rcParams.update(mpl.rcParamsDefault)
mpl.rcParams.update(mpl.rcParamsOrig)
%matplotlib inline

print mpl.matplotlib_fname()
```

Populating the interactive namespace from numpy and matplotlib
`/home/astrodoo/anaconda2/lib/python2.7/site-packages/matplotlib/mpl-data/matplotlibrc`

```
In [5]: # recover matplotlib defaults
#import matplotlib as mpl
```

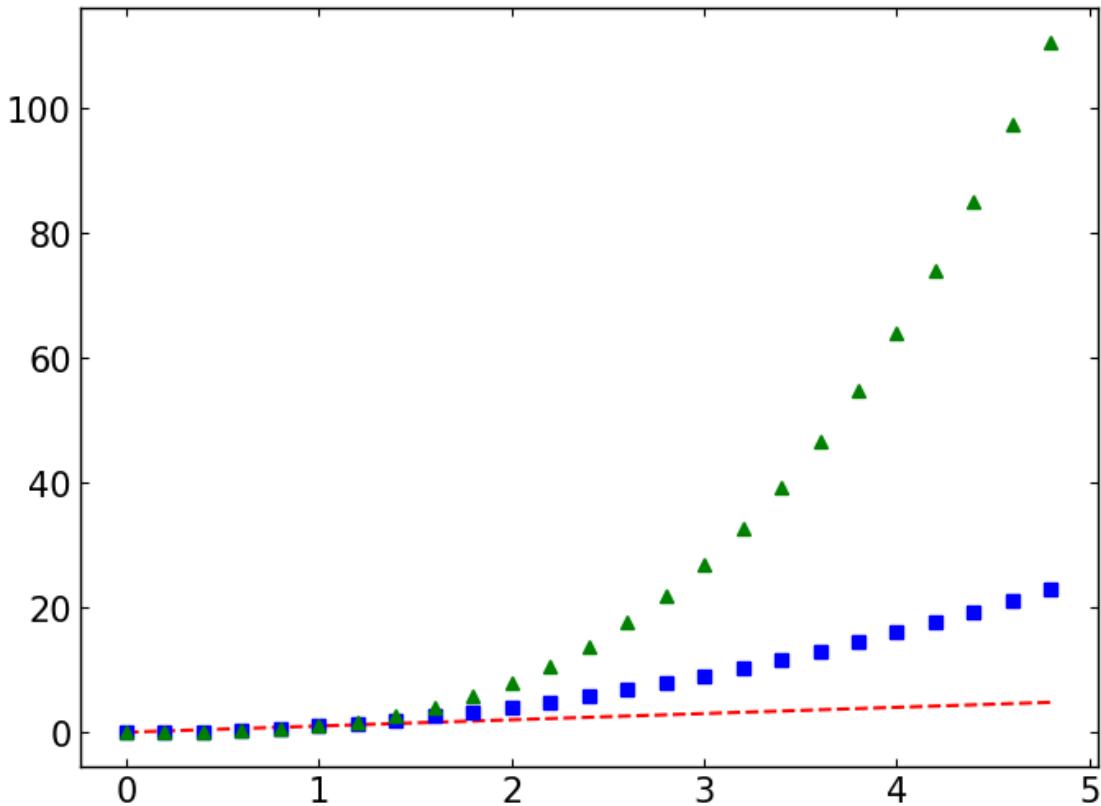
```

#mpl.rcParams.update(mpl.rcParamsDefault)
#%pylab inline

import matplotlib
#matplotlib.use('Agg')
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0.,5.,0.2)
plt.plot(t,t,'r--',t,t**2,'bs',t,t**3,'g^')
#plt.show()
#plt.savefig('../figures/test.png')
#plt.savefig('../figures/test.eps')

```



1.1 Multiplots

```

In [6]: import numpy as np
        import matplotlib.pyplot as plt

        def f(t):
            return np.exp(-t)*np.cos(2.*np.pi*t)

```

```

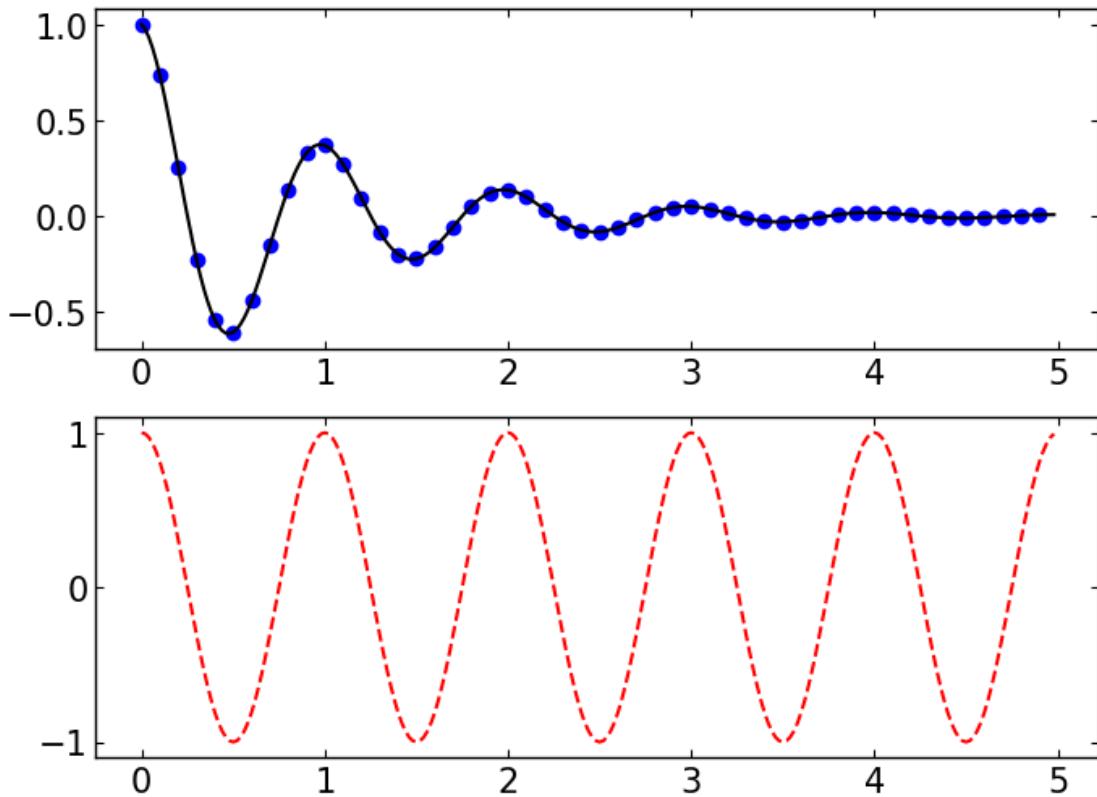
t1 = np.arange(0.,5.,0.1)
t2 = np.arange(0.,5.,0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1,f(t1),'bo',t2,f(t2),'k')

plt.subplot(212)
plt.plot(t2,np.cos(2.*np.pi*t2),'r--')

# set the number of ticks
# if the plot is log scale:
# > plt.locator_params(axis='y',numticks=3)
# for the ticks in colorbar, you should check matplotlib3.ipynb
plt.locator_params(axis='y',nbins=3)
plt.show()

```



1.2 Two scales

Demonstrate how to do two plots on the same axes with different left right scales.

The trick is to use 2 *different axes*. Turn the axes rectangular frame off on the 2nd axes to keep it from obscuring the first. Manually set the tick locs and labels as desired. You can use separate matplotlib.ticker formatters and locators as desired since the two axes are independent.

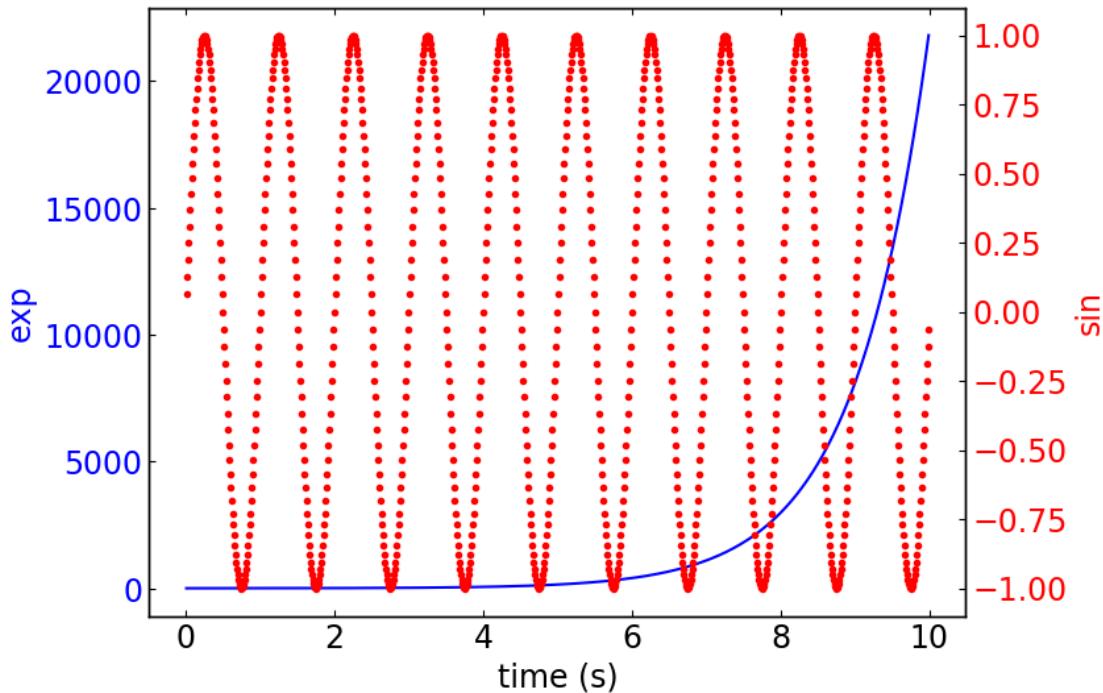
This is achieved in the following example by calling the Axes.twinx() method, which performs this work. See the source of twinx() in axes.py for an example of how to do it for different x scales. (Hint: use the xaxis instance and call tick_bottom and tick_top in place of tick_left and tick_right.)

The twinx and twiny methods are also exposed as pyplot functions.

```
In [4]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        fig, ax1 = plt.subplots()
        t = np.arange(0.01, 10.0, 0.01)
        s1 = np.exp(t)
        ax1.plot(t, s1, 'b-')
        ax1.set_xlabel('time (s)')
        # Make the y-axis label and tick labels match the line color.
        ax1.set_ylabel('exp', color='b')
        for tl in ax1.get_yticklabels():
            tl.set_color('b')

        ax2 = ax1.twinx()
        s2 = np.sin(2*np.pi*t)
        ax2.plot(t, s2, 'r.')
        ax2.set_ylabel('sin', color='r')
        for tl in ax2.get_yticklabels():
            tl.set_color('r')
        plt.show()
```



1.3 Log scale

```
In [9]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y>0) & (y<1)]
y.sort()
x = np.arange(len(y))

plt.figure(1)

#linear plot
plt.subplot(311)
plt.plot(x,y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)

#log
plt.subplot(312)
plt.plot(x,y)
plt.yscale('log')
```

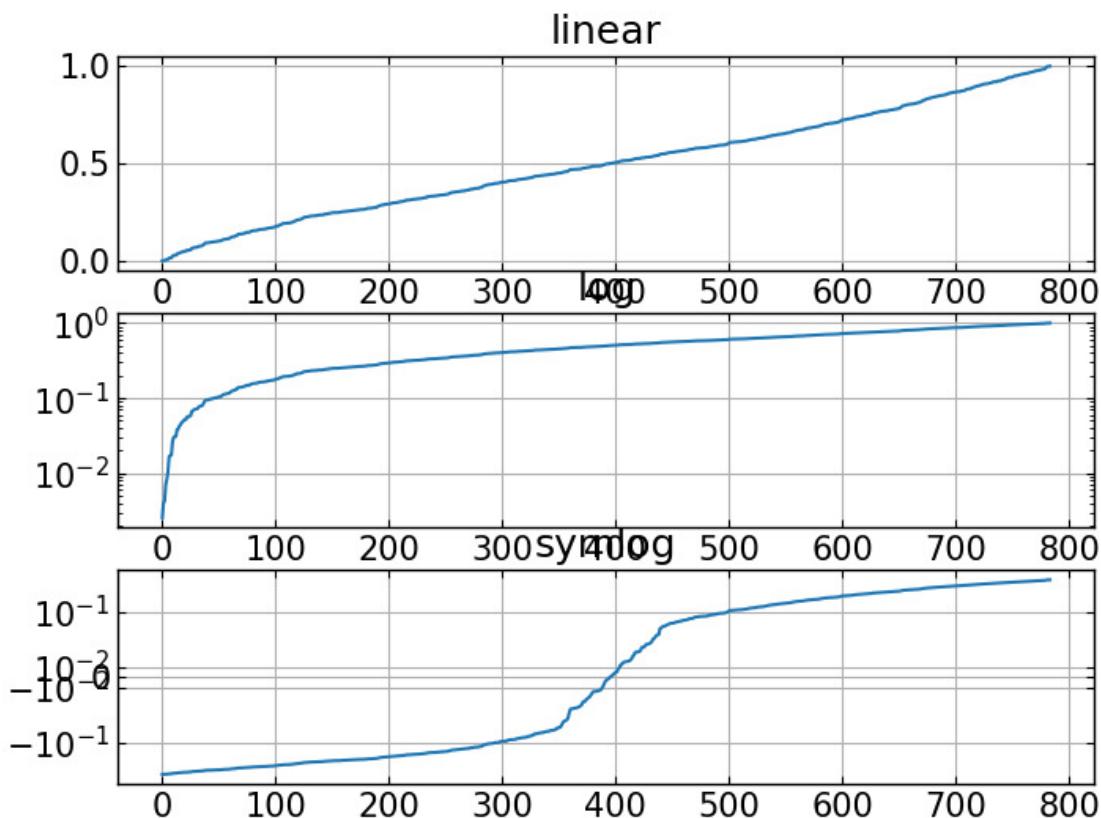
```

plt.title('log')
plt.grid(True)
plt.locator_params(axis='y', numticks=4)      # Note that in log scale, the number of ticks is not necessarily 4

#symmetric log
plt.subplot(313)
plt.plot(x,y-y.mean())
plt.yscale('symlog', linthreshy=0.05)
plt.title('symlog')
plt.grid(True)
#plt.locator_params(axis='y', numticks=4)

plt.show()

```



```

In [10]: import numpy as np
         import matplotlib.pyplot as plt

         plt.subplots_adjust(hspace=0.4)
         t = np.arange(0.01, 20.0, 0.01)

# log y axis

```

```

plt.subplot(221)
plt.semilogy(t, np.exp(-t/5.0))
plt.title('semilogy')
plt.grid(True)

# log x axis
plt.subplot(222)
plt.semilogx(t, np.sin(2*np.pi*t))
plt.title('semilogx')
plt.grid(True)

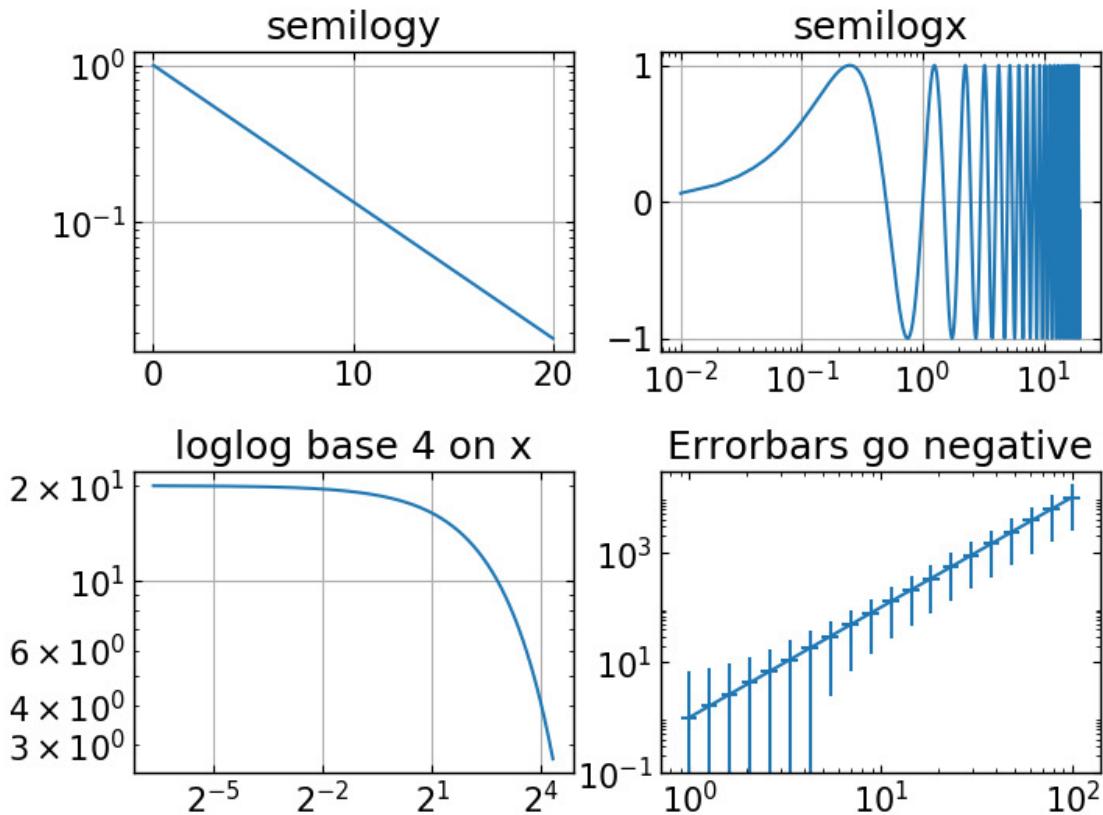
# log x and y axis
plt.subplot(223)
plt.loglog(t, 20*np.exp(-t/10.0), basex=2)
plt.grid(True)
plt.title('loglog base 4 on x')

# with errorbars: clip non-positive values
ax = plt.subplot(224)
ax.set_xscale("log", nonposx='clip')
ax.set_yscale("log", nonposy='clip')

x = 10.0**np.linspace(0.0, 2.0, 20)
y = x**2.0
plt.errorbar(x, y, xerr=0.1*x, yerr=5.0 + 0.75*y)
ax.set_ylim(ymin=0.1)
ax.set_title('Errorbars go negative')

plt.show()

```



1.4 Text with LaTex

```
In [11]: import matplotlib.pyplot as plt

fig = plt.figure()
fig.suptitle('bold figure suptitle', fontsize=14, fontweight='bold')

ax = fig.add_subplot(111)
fig.subplots_adjust(top=0.85)
ax.set_title('axes title')

ax.set_xlabel('xlabel')
ax.set_ylabel('ylavel')

ax.text(1,8, 'boxed italics text in data coords', style='italic',
       bbox={'facecolor':'red','alpha':0.5,'pad':10})
ax.text(2,6, r'an equation: $E=mc^2$', fontsize=15)
ax.text(3,2,unicode('unicode: Institut für Festkörperphysik', 'latin-1'))
ax.text(0.95,0.01,'colored text in axes coords',
       verticalalignment='bottom', horizontalalignment='right',
       transform=ax.transAxes,
```

```

        color='green', fontsize=15)

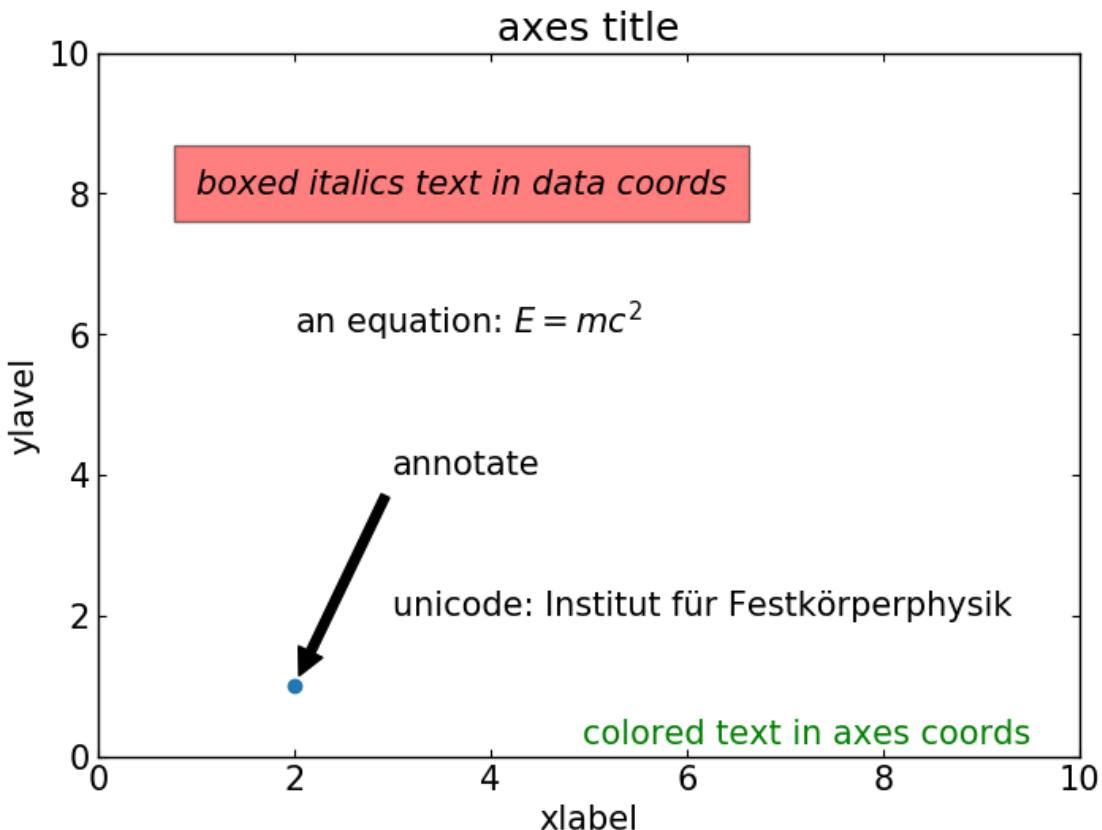
ax.plot([2],[1], 'o')
ax.annotate('annotate', xy=(2,1), xytext=(3,4),
            arrowprops=dict(facecolor='black', shrink=0.05))

ax.axis([0,10,0,10])

plt.show()

```

bold figure suptitle



```

In [12]: import numpy as np
          import matplotlib.pyplot as plt

t = np.arange(0.,2.,0.01)
s = np.sin(2.*np.pi*t)

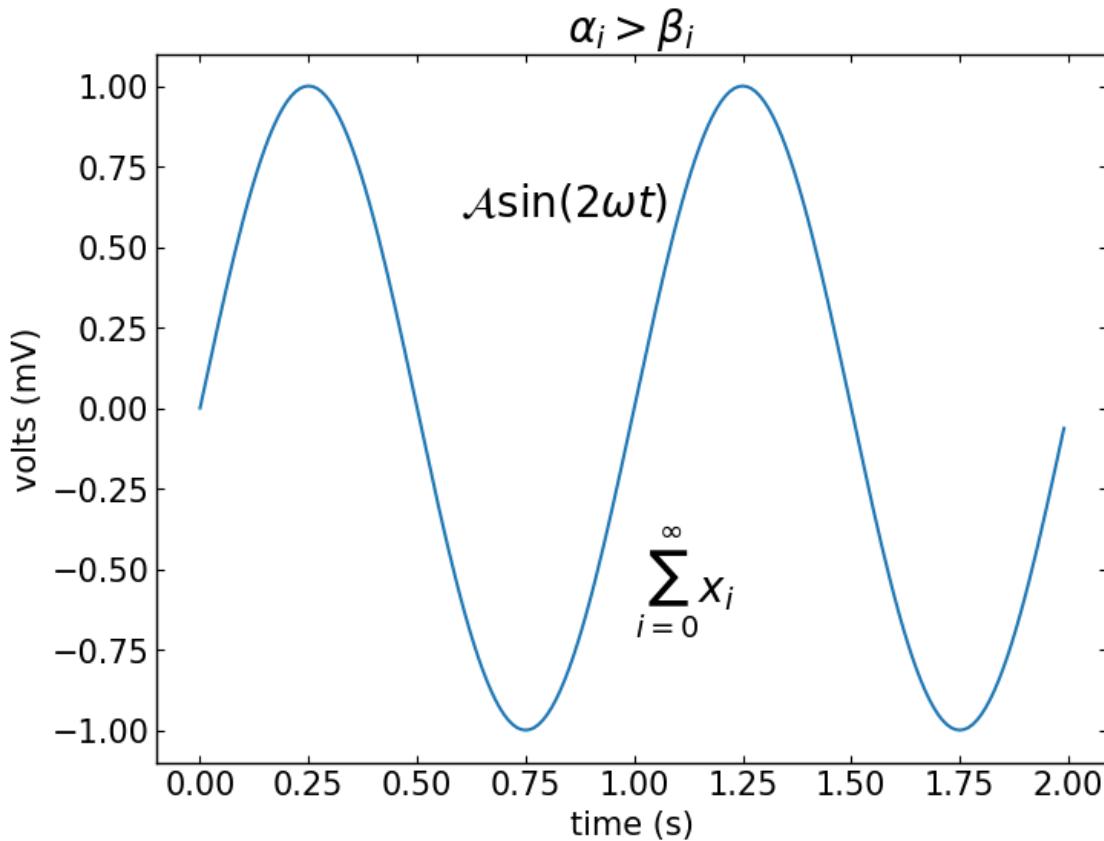
plt.plot(t,s)
plt.title(r'$\alpha_i > \beta_i$', fontsize=20)

```

```

plt.text(1,-0.6, r'$\sum_{i=0}^{\infty} x_i$', fontsize=20)
plt.text(0.6,0.6, r'$\mathcal{A}\sin(2 \omega t)$',
         fontsize=20)
plt.xlabel('time (s)')
plt.ylabel('volts (mV)')
plt.show()

```



```

In [13]: """
Selected features of Matplotlib's math rendering engine.
"""

from __future__ import print_function
import matplotlib.pyplot as plt
import os
import sys
import re
import gc

# Selection of features following "Writing mathematical expressions" tutorial
mathtext_titles = {
    0: "Header demo",

```

```

1: "Subscripts and superscripts",
2: "Fractions, binomials and stacked numbers",
3: "Radicals",
4: "Fonts",
5: "Accents",
6: "Greek, Hebrew",
7: "Delimiters, functions and Symbols"}
n_lines = len(mathtext_titles)

# Randomly picked examples
mathext_demos = {
    0: r"$W^{\beta} \delta_1 \rho_1 \sigma_2 = "
    r"U^{\beta} \delta_1 \rho_1 + \frac{1}{8 \pi^2} "
    r"\int^{\alpha_2}_{\alpha_1} d \alpha^{\prime 2} \left[ \frac{ "
    r"U^{2\beta} \delta_1 \rho_1 - \alpha^{\prime 2} U^{\beta} \rho_1 \sigma_2 }{U^0 \rho_1 \sigma_2} \right] $",
    1: r"\alpha_i > \beta_i, "
    r"\alpha_{i+1}^j = \sin(2\pi f_j t_i) e^{-5 t_i/\tau}, "
    r"\dots",
    2: r"\frac{3}{4}, \binom{3}{4}, \stackrel{3}{4}, "
    r"\left(\frac{5}{4} - \frac{1}{x}\right), \dots",
    3: r"\sqrt{2}, \sqrt[3]{x}, \dots",
    4: r"\mathrm{Roman}, \mathit{Italic}, \mathtt{Typewriter} \ "
    r"\mathrm{or} \ \mathcal{CALLIGRAPHY}",
    5: r"\acute{a}, \bar{a}, \breve{a}, \dot{a}, \ddot{a}, \grave{a}, \ "
    r"\hat{a}, \tilde{a}, \vec{a}, \widehat{xyz}, \widetilde{xyz}, \ "
    r"\dots",
    6: r"\alpha, \beta, \chi, \delta, \lambda, \mu, \ "
    r"\Delta, \Gamma, \Omega, \Phi, \Pi, \Upsilon, \nabla, \ "
    r"\aleph, \beth, \daleth, \gimel, \dots",
    7: r"\coprod, \int, \oint, \prod, \sum, \ "
    r"\log, \sin, \approx, \oplus, \star, \varpropto, \ "
    r"\infty, \partial, \Re, \leftrightsquigarrow, \ \dots"
}

def doall():
    # Colors used in mpl online documentation.
    mpl_blue_rvb = (191./255., 209./256., 212./255.)
    mpl_orange_rvb = (202/255., 121/256., 0./255.)
    mpl_grey_rvb = (51./255., 51./255., 51./255.)

```

```

# Creating figure and axis.
plt.figure(figsize=(6, 7))
# plt.axes([0.01, 0.01, 0.98, 0.90], axisbg="white", frameon=True) # axisbg is no
plt.axes([0.01, 0.01, 0.98, 0.90], frameon=True)
plt.gca().set_xlim(0., 1.)
plt.gca().set_ylim(0., 1.)
plt.gca().set_title("Matplotlib's math rendering engine",
                     color=mpl_grey_rvb, fontsize=14, weight='bold')
plt.gca().set_xticklabels("", visible=False)
plt.gca().set_yticklabels("", visible=False)

# Gap between lines in axes coords
line_axesfrac = (1. / (n_lines))

# Plotting header demonstration formula
full_demo = mathext_demos[0]
plt.annotate(full_demo,
            xy=(0.5, 1. - 0.59*line_axesfrac),
            xycoords='data', color=mpl_orange_rvb, ha='center',
            fontsize=20)

# Plotting features demonstration formulae
for i_line in range(1, n_lines):
    baseline = 1. - (i_line)*line_axesfrac
    baseline_next = baseline - line_axesfrac*1.
    title = mathtext_titles[i_line] + ":" + "\n"
    fill_color = ['white', mpl_blue_rvb][i_line % 2]
    plt.fill_between([0., 1.], [baseline, baseline],
                    [baseline_next, baseline_next],
                    color=fill_color, alpha=0.5)
    plt.annotate(title,
                xy=(0.07, baseline - 0.3*line_axesfrac),
                xycoords='data', color=mpl_grey_rvb, weight='bold')
    demo = mathext_demos[i_line]
    plt.annotate(demo,
                xy=(0.05, baseline - 0.75*line_axesfrac),
                xycoords='data', color=mpl_grey_rvb,
                fontsize=16)

for i in range(n_lines):
    s = mathext_demos[i]
    print(i, s)
plt.show()

if '--latex' in sys.argv:
    # Run: python mathtext_examples.py --latex
    # Need amsmath and amssymb packages.
    fd = open("mathtext_examples.ltx", "w")

```

```

fd.write("\\"documentclass{article}\n")
fd.write("\\"usepackage{amsmath, amssymb}\n")
fd.write("\\"begin{document}\n")
fd.write("\\"begin{enumerate}\n")

for i in range(n_lines):
    s = mathext_demos[i]
    s = re.sub(r"(?<!\\)\$+", " $$", s)
    fd.write("\\"item %s\n" % s)

fd.write("\\"end{enumerate}\n")
fd.write("\\"end{document}\n")
fd.close()

os.system("pdflatex mathtext_examples.ltx")
else:
    doall()

0 $W^{3\beta}_{\delta_1 \rho_1 \sigma_2} = U^{3\beta}_{\delta_1 \rho_1} + \frac{1}{8 \pi^2} \int
1 \$\alpha_i > \beta_i, \ \alpha_{i+1}^j = \{\rm sin\}(2\pi f_j t_i) e^{-5 t_i/\tau}, \ \ldots
2 $\frac{3}{4}, \ \binom{3}{4}, \ \stackrel{3}{\left( \frac{5 - \frac{1}{x}}{4} \right)}, \ \ldots
3 $\sqrt{2}, \ \sqrt[3]{x}, \ \ldots
4 $\mathrm{Roman}\ , \ \mathit{Italic}\ , \ \mathtt{Typewriter}\ \mathrm{or}\ \mathcal{CALLIGR}
5 $\acute{a}, \ \bar{a}, \ \breve{a}, \ \dot{a}, \ \ddot{a}, \ \grave{a}, \ \hat{a}, \ \tilde{a}, \ \vec{a}, \ \ddot{a}
6 $\alpha, \ \beta, \ \chi, \ \delta, \ \lambda, \ \mu, \ \Delta, \ \Gamma, \ \Omega, \ \Phi, \ \Pi, \ \Upsilon
7 $\coprod, \ \int, \ \oint, \ \prod, \ \sum, \ \log, \ \sin, \ \approx, \ \oplus, \ \star, \ \varpropto

```

Matplotlib's math rendering engine

$$W_{\delta_1\rho_1\sigma_2}^{3\beta} = U_{\delta_1\rho_1}^{3\beta} + \frac{1}{8\pi 2} \int_{\alpha_2}^{\alpha_2} d\alpha'_2 \left[\frac{U_{\delta_1\rho_1}^{2\beta} - \alpha'_2 U_{\rho_1\sigma_2}^{1\beta}}{U_{\rho_1\sigma_2}^{0\beta}} \right]$$

Subscripts and superscripts:

$\alpha_i > \beta_i$, $\alpha_{i+1}^j = \sin(2\pi f_j t_i) e^{-5t_i/\tau}$, ...

Fractions, binomials and stacked numbers:

$\frac{3}{4}$, $(\frac{3}{4})$, $\frac{3}{4}$, $(\frac{5-\frac{1}{x}}{4})$, ...

Radicals:

$\sqrt{2}$, $\sqrt[3]{x}$, ...

Fonts:

Roman, Italic, Typewriter or *CALLIGRAPHY*

Accents:

\acute{a} , \bar{a} , \check{a} , \grave{a} , \ddot{a} , \hat{a} , \tilde{a} , \vec{a} , \widehat{xyz} , \widetilde{xyz} , ...

Greek, Hebrew:

α , β , χ , δ , λ , μ , Δ , Γ , Ω , Φ , Π , Υ , ∇ , \aleph , \beth , \beth_1 , \beth_2 , ...

Delimiters, functions and Symbols:

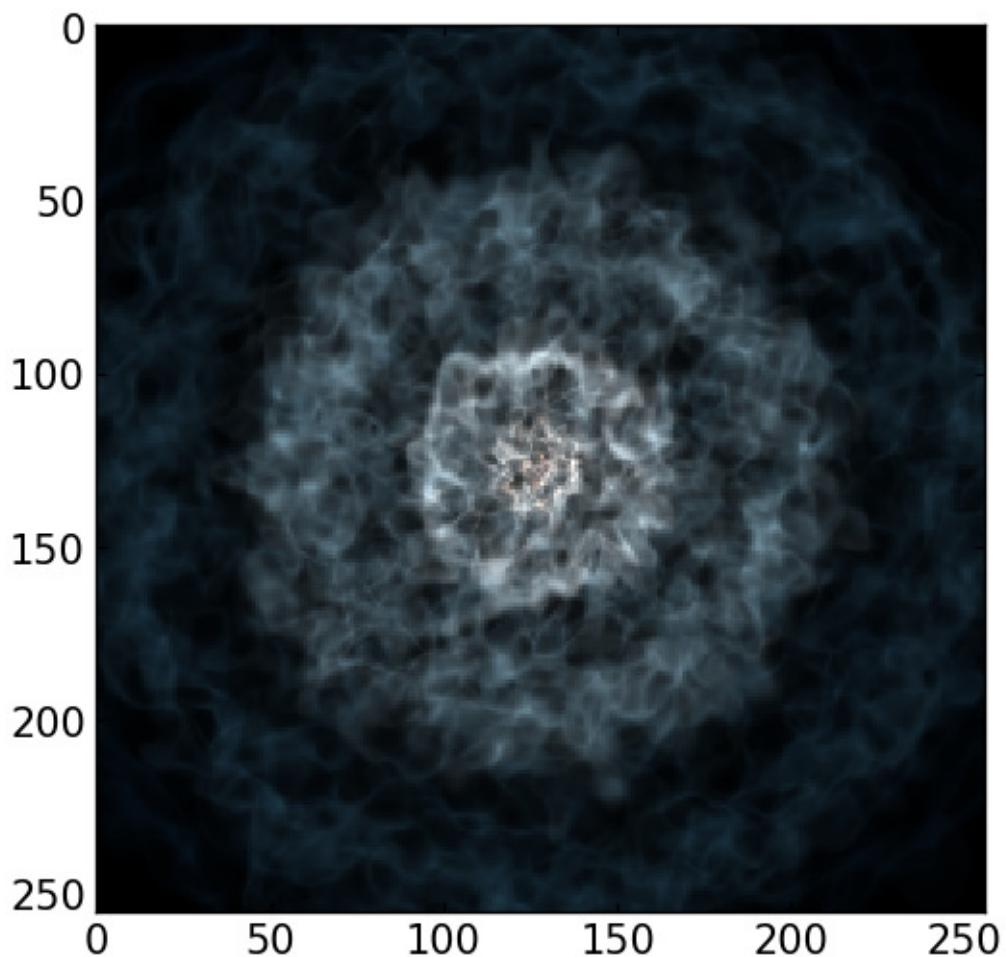
\lfloor , \rfloor , \oint , \prod , \sum , \log , \sin , \approx , \oplus , $*$, \propto , ∞ , ∂ , \Re , \Leftrightarrow , ...

1.5 2D Image & colorbar

```
In [14]: import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        import numpy as np

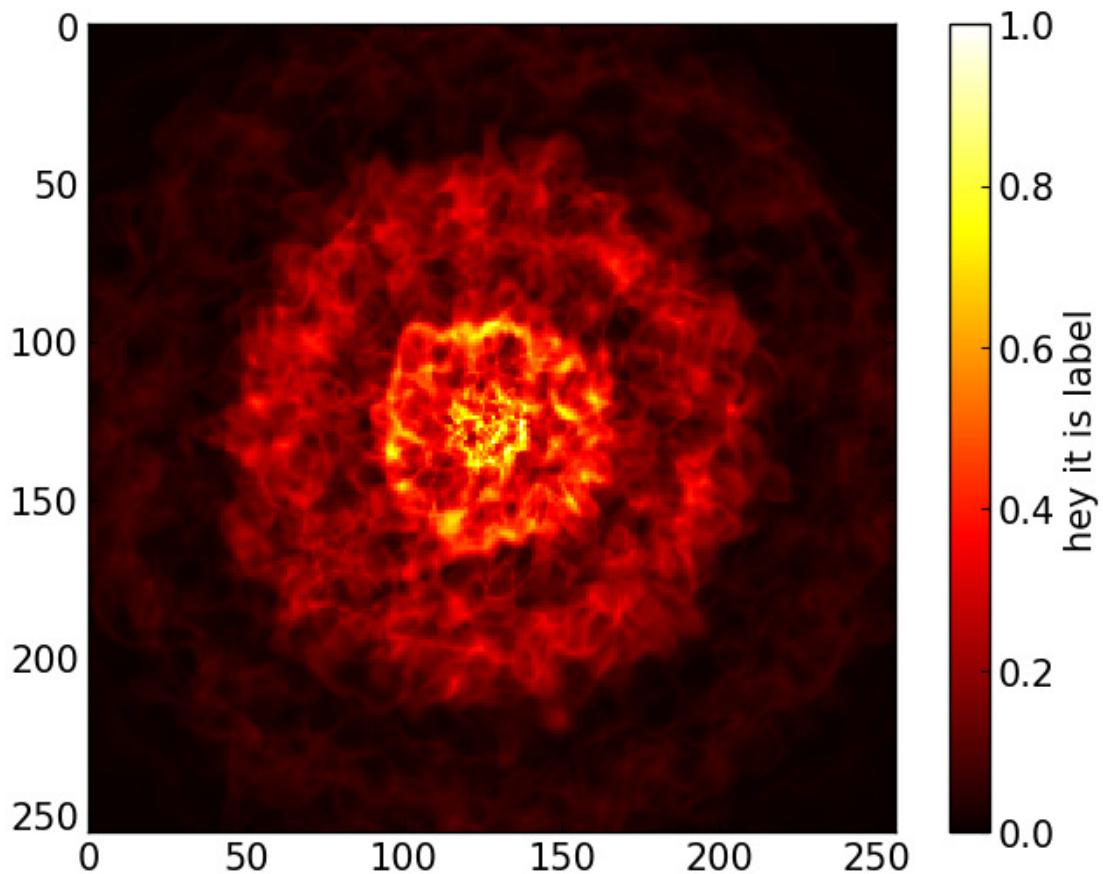
%matplotlib inline

img = mpimg.imread('../figures/test_rendering.png')
imgplot = plt.imshow(img)
```

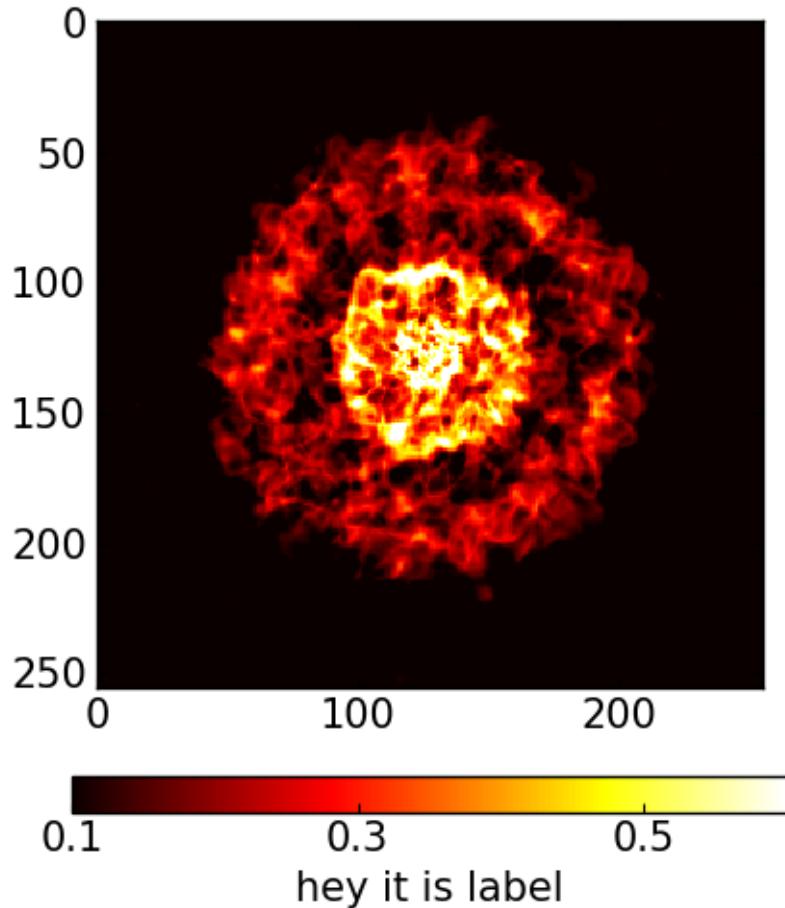


```
In [15]: lum_img = img[:, :, 0]
imgplot2 = plt.imshow(lum_img)
imgplot2.set_cmap('hot')
plt.colorbar(label='hey it is label')
```

```
Out[15]: <matplotlib.colorbar.Colorbar at 0x7f41ae799450>
```



```
In [16]: imgplot2 = plt.imshow(lum_img)
imgplot2.set_cmap('hot')
imgplot2.set_clim(0.1,0.6)
clb = plt.colorbar(imgplot2,orientation='horizontal', label='hey it is label' \
                   ,aspect=20,shrink=0.6,pad=0.1 \
                   ,ticks=[i*0.2+0.1 for i in range(3)])
```



```
In [17]: """ draw colorbar in manual way - position, width """
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt

%matplotlib inline

fig = plt.figure(figsize=(5,7))

cmap1 = mpl.cm.cool
minv, maxv = 0.1, 0.6

img = mpimg.imread('../figures/test_rendering.png')
imgplot = plt.imshow(img[:, :, 0])
imgplot.set_cmap(cmap1)
imgplot.set_clim(minv, maxv)

# axis for color bar
```

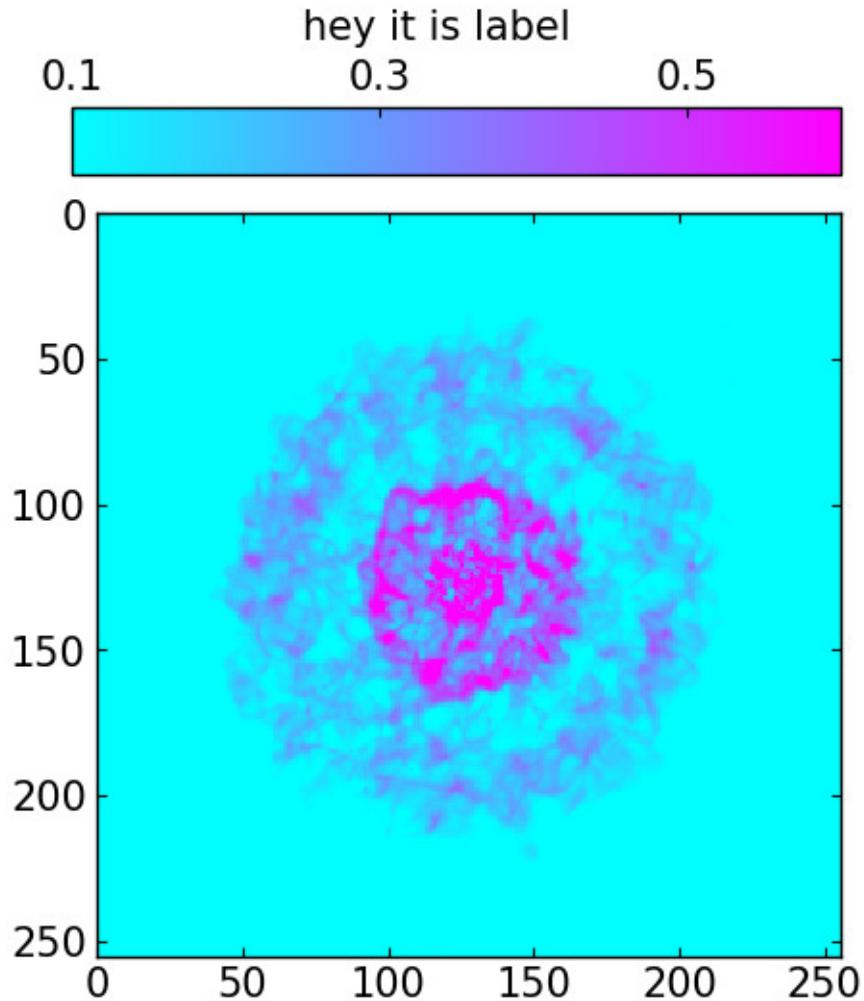
```

ax1 = fig.add_axes([0.1,0.8,0.8,0.05])    #[x,y, x-width, y-width]

norm = mpl.colors.Normalize(vmin=minv, vmax=maxv)
clb = mpl.colorbar.ColorbarBase(ax1,orientation='horizontal' \
                                ,norm=norm,cmap=cmap1 \
                                ,ticks=[i*0.2+0.1 for i in range(3)])
ax1.xaxis.set_ticks_position("top")
ax1.annotate('hey it is label',(0.3,2.),xycoords='axes fraction')

```

Out[17]: <matplotlib.text.Annotation at 0x7f41aea08150>



```

In [18]: import matplotlib.pyplot as plt
        import numpy as np
%matplotlib inline

arr = np.arange(100).reshape((10,10))

```

```

plt.close('all')
fig = plt.figure(figsize=(5,4))

ax = plt.subplot(111)
im = ax.imshow(arr, interpolation='none')

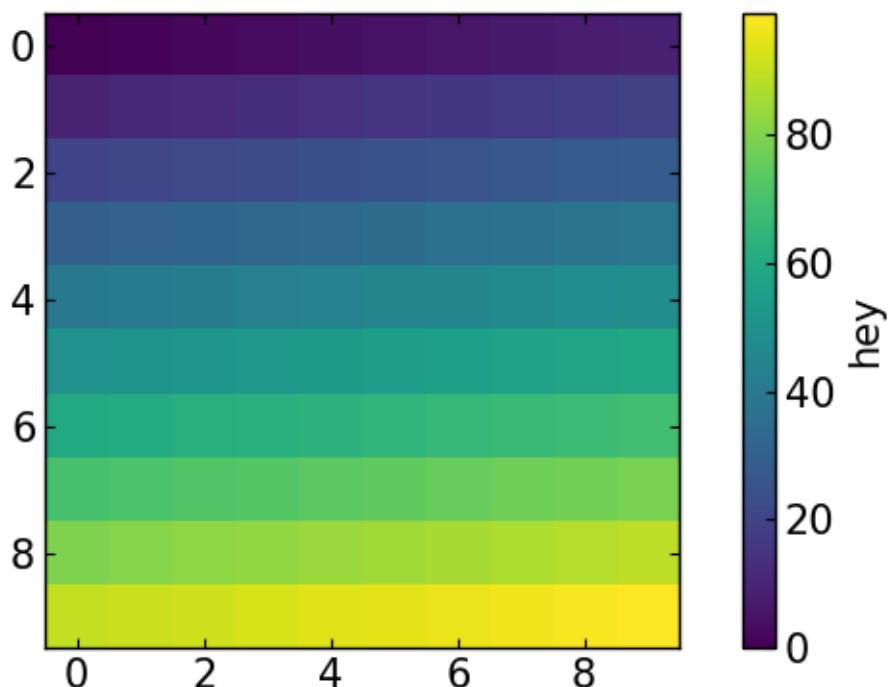
from mpl_toolkits.axes_grid1 import make_axes_locatable

divider = make_axes_locatable(plt.gca())
cax = divider.append_axes("right", "5%", pad="10%")
cab = plt.colorbar(im, cax=cax)
cab.set_label('hey')

plt.tight_layout()

plt.show()

```



```

In [19]: import matplotlib.pyplot as plt
        import numpy as np
        %matplotlib inline

arr = np.arange(100).reshape((10,10))

```

```

plt.close('all')
fig = plt.figure(figsize=(5,4))

ax = plt.subplot(111)
im = ax.imshow(arr, interpolation='none')

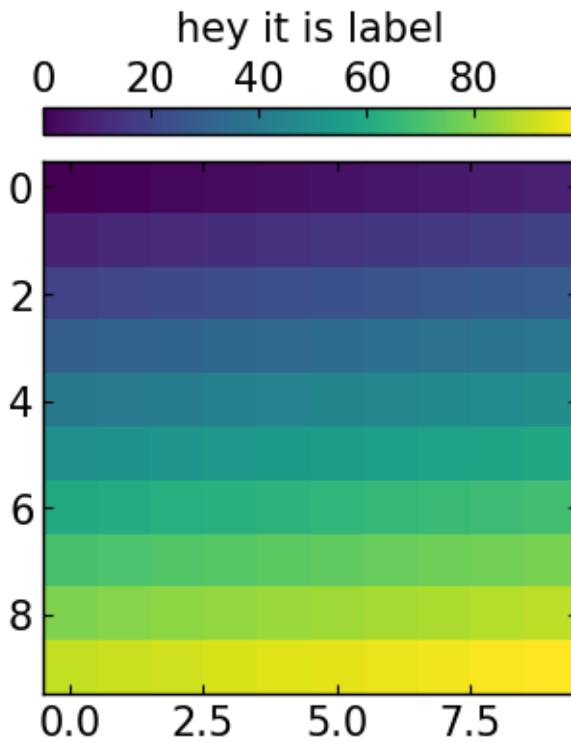
""" This allows to fit the color bar with the size of figure """
from mpl_toolkits.axes_grid1 import make_axes_locatable

divider = make_axes_locatable(plt.gca())
cax = divider.append_axes("top", "5%", pad="5%")
cab = plt.colorbar(im, cax=cax, orientation='horizontal')
cax.xaxis.set_ticks_position('top')
#cab.set_label('hey')
cab.annotate('hey it is label',(0.5,3.5),xycoords='axes fraction',ha='center')

plt.tight_layout()

plt.show()

```



```
In [20]: import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
```

```

#discrete color scheme
cMap = ListedColormap(['white', 'green', 'blue','red'])

#data
np.random.seed(42)
data = np.random.rand(4, 4)
fig, ax = plt.subplots()
heatmap = ax.pcolor(data, cmap=cMap)

#legend
cbar = plt.colorbar(heatmap)

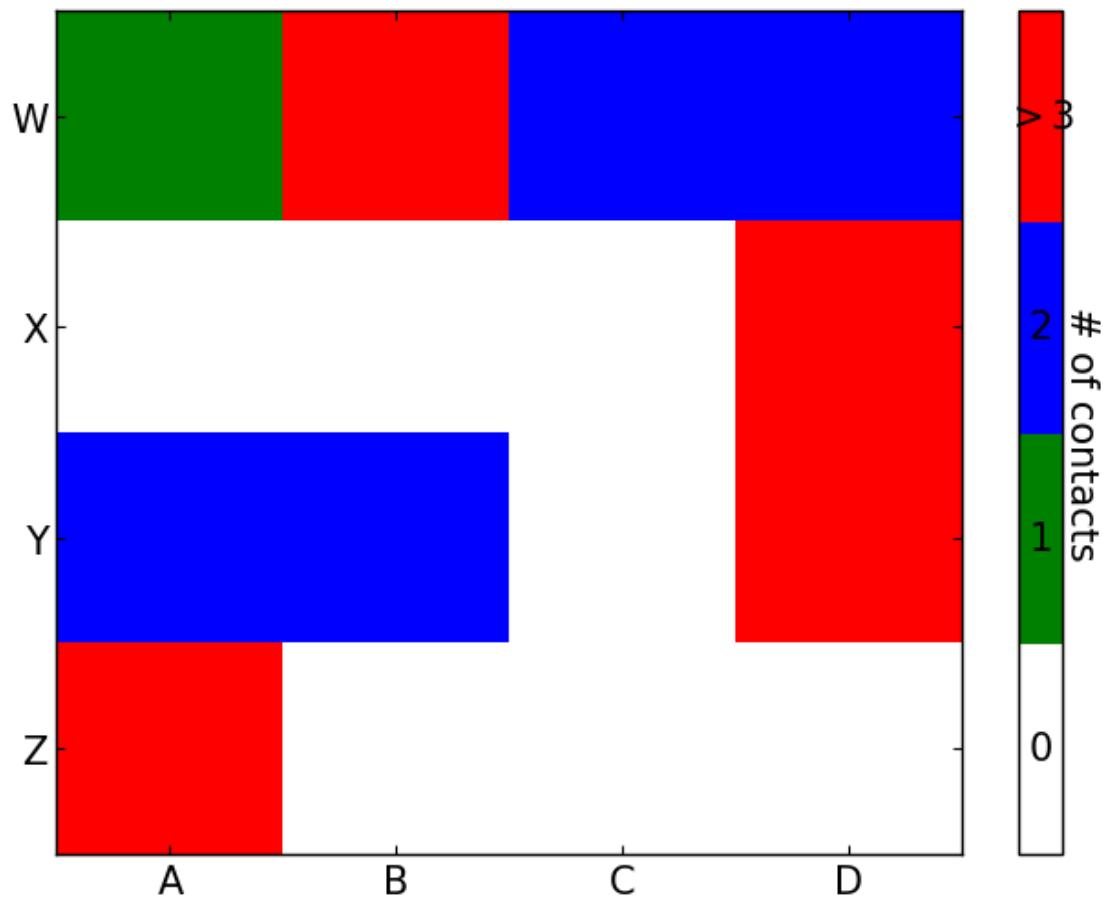
cbar.ax.get_yaxis().set_ticks([])
for j, lab in enumerate(['$0$', '$1$', '$2$', '$>3$']):
    cbar.ax.text(.5, (2 * j + 1) / 8.0, lab, ha='center', va='center')
cbar.ax.get_yaxis().labelpad = 15
cbar.ax.set_ylabel('# of contacts', rotation=270)

# put the major ticks at the middle of each cell
ax.set_xticks(np.arange(data.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(data.shape[0]) + 0.5, minor=False)
ax.invert_yaxis()

#labels
column_labels = list('ABCD')
row_labels = list('WXYZ')
ax.set_xticklabels(column_labels, minor=False)
ax.set_yticklabels(row_labels, minor=False)

plt.show()

```



1.6 Contour

In [21]: `"""`

Illustrate simple contour plotting, contours on an image with a colorbar for the contours, and labelled contours.

See also `contour_image.py`.

`"""`

```
import matplotlib
import numpy as np
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

#matplotlib.rcParams['xtick.direction'] = 'out'
#matplotlib.rcParams['ytick.direction'] = 'out'
```

```

delta = 0.025
x = np.arange(-3.0, 3.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = mlab.bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)
Z2 = mlab.bivariate_normal(X, Y, 1.5, 0.5, 1, 1)
# difference of Gaussians
Z = 10.0 * (Z2 - Z1)

# Create a simple contour plot with labels using default colors. The
# inline argument to clabel will control whether the labels are draw
# over the line segments of the contour, removing the lines beneath
# the label
plt.figure()
CS = plt.contour(X, Y, Z)
plt.clabel(CS, inline=1, fontsize=10)
plt.title('Simplest default with labels')

# contour labels can be placed manually by providing list of positions
# (in data coordinate). See ginput_manual_clabel.py for interactive
# placement.
plt.figure()
CS = plt.contour(X, Y, Z)
manual_locations = [(-1, -1.4), (-0.62, -0.7), (-2, 0.5), (1.7, 1.2), (2.0, 1.4), (2.4, 1.4)]
plt.clabel(CS, inline=1, fontsize=10, manual=manual_locations)
plt.title('labels at selected locations')

# You can force all the contours to be the same color.
plt.figure()
CS = plt.contour(X, Y, Z, 6,
                  colors='k', # negative contours will be dashed by default
                  )
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Single color - negative contours dashed')

# You can set negative contours to be solid instead of dashed:
matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
plt.figure()
CS = plt.contour(X, Y, Z, 6,
                  colors='k', # negative contours will be dashed by default
                  )
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Single color - negative contours solid')

```

```

# And you can manually specify the colors of the contour
plt.figure()
CS = plt.contour(X, Y, Z, 6,
                  linewidths=np.arange(.5, 4, .5),
                  colors=('r', 'green', 'blue', (1, 1, 0), '#afeeee', '0.5')
)
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Crazy lines')

# Or you can use a colormap to specify the colors; the default
# colormap will be used for the contour lines
plt.figure()
im = plt.imshow(Z, interpolation='bilinear', origin='lower',
                 cmap=cm.gray, extent=(-3, 3, -2, 2))
levels = np.arange(-1.2, 1.6, 0.2)
CS = plt.contour(Z, levels,
                  origin='lower',
                  linewidths=2,
                  extent=(-3, 3, -2, 2))

# Thicken the zero contour.
zc = CS.collections[6]
plt.setp(zc, linewidth=4)

plt.clabel(CS, levels[1::2], # label every second level
           inline=1,
           fmt='%1.1f',
           fontsize=14)

# make a colorbar for the contour lines
CB = plt.colorbar(CS, shrink=0.8, extend='both')

plt.title('Lines with colorbar')
#plt.hot() # Now change the colormap for the contour lines and colorbar
plt.flag()

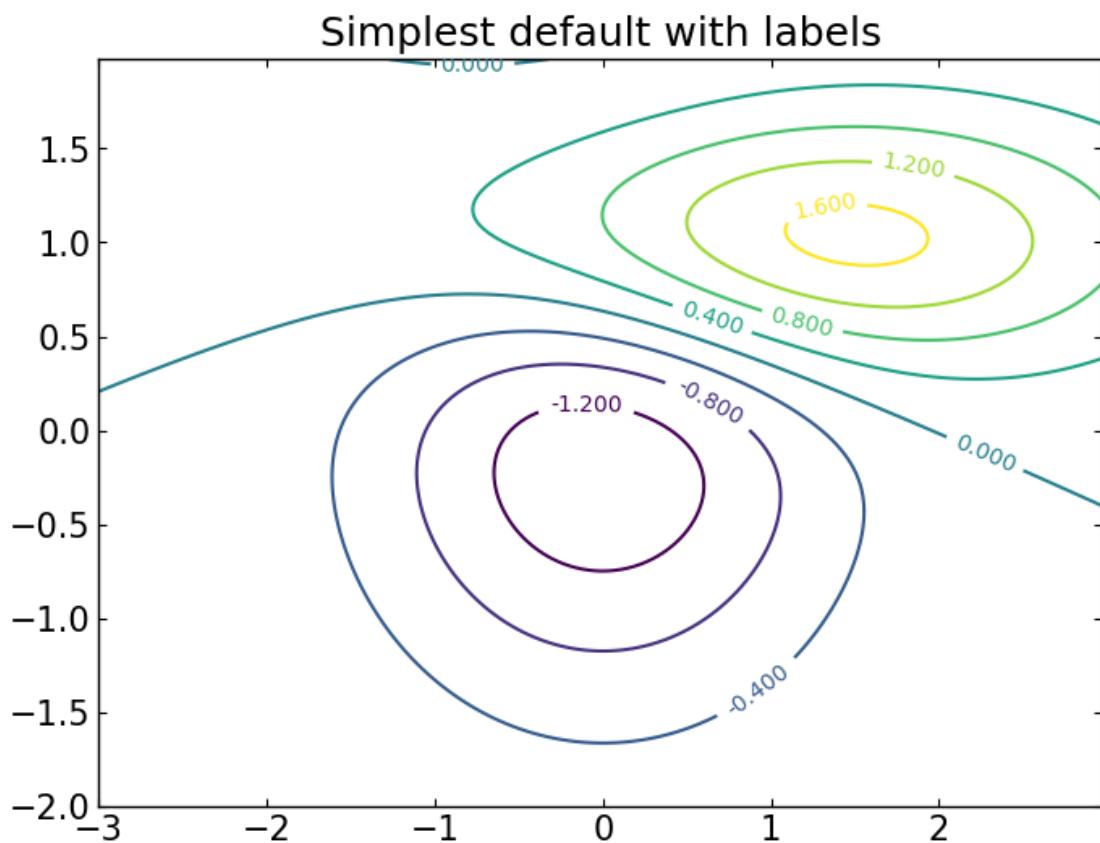
# We can still add a colorbar for the image, too.
CBI = plt.colorbar(im, orientation='horizontal', shrink=0.8)

# This makes the original colorbar look a bit out of place,
# so let's improve its position.

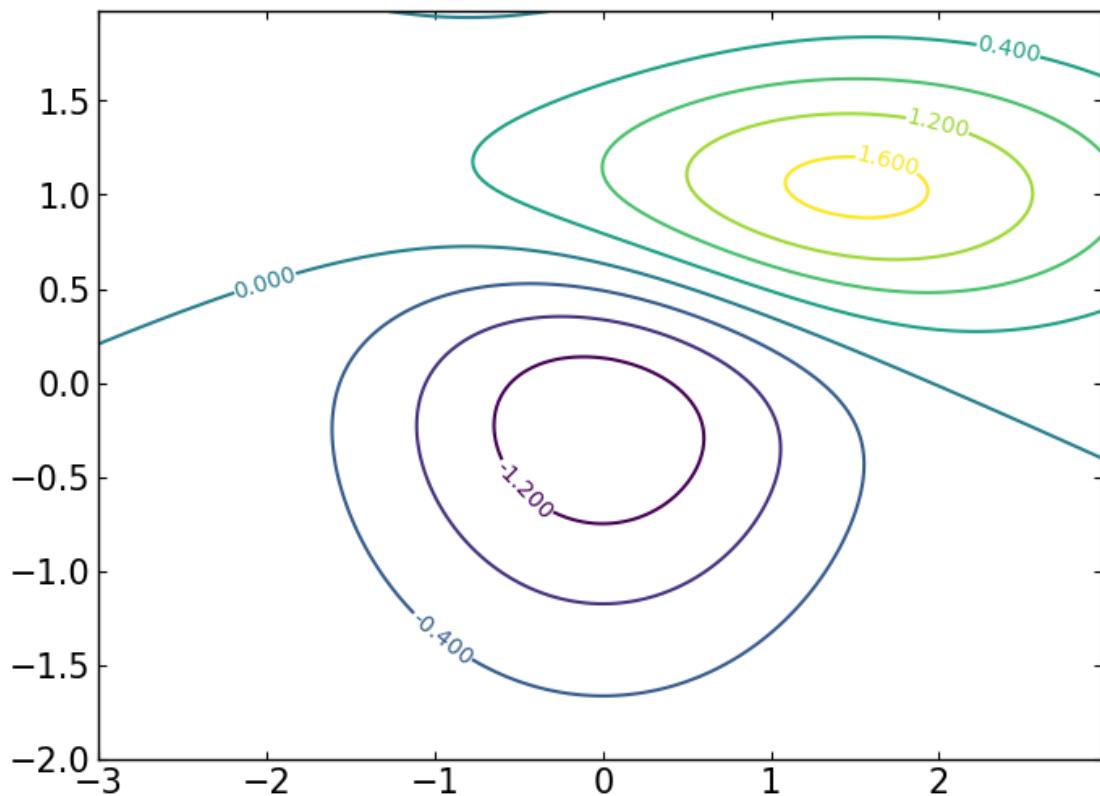
l, b, w, h = plt.gca().get_position().bounds
ll, bb, ww, hh = CB.ax.get_position().bounds
CB.ax.set_position([ll, b + 0.1*h, ww, hh*0.8])

```

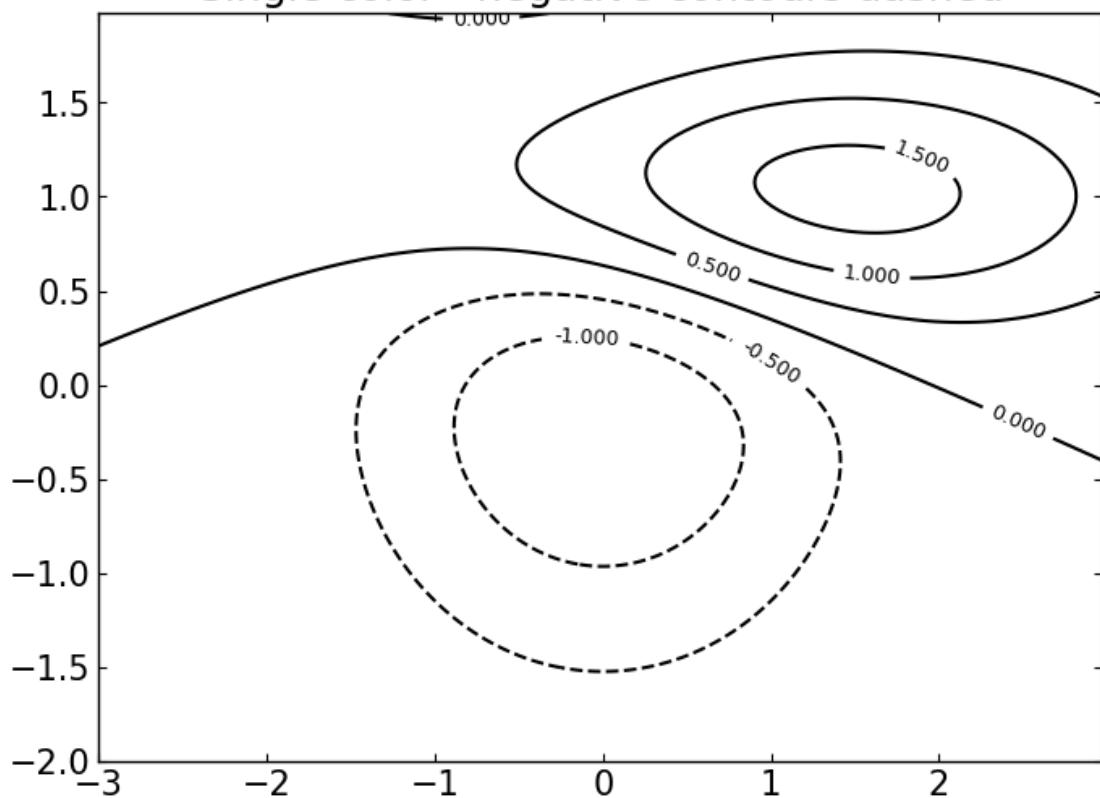
```
plt.show()
```



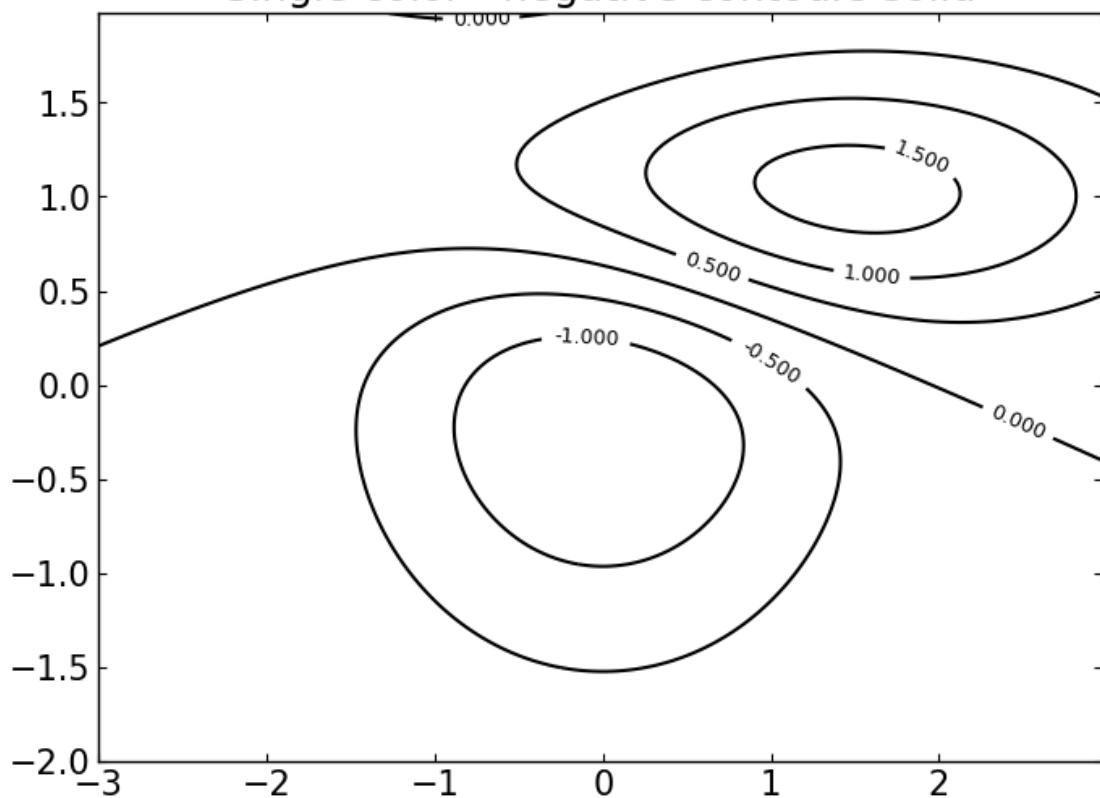
labels at selected locations



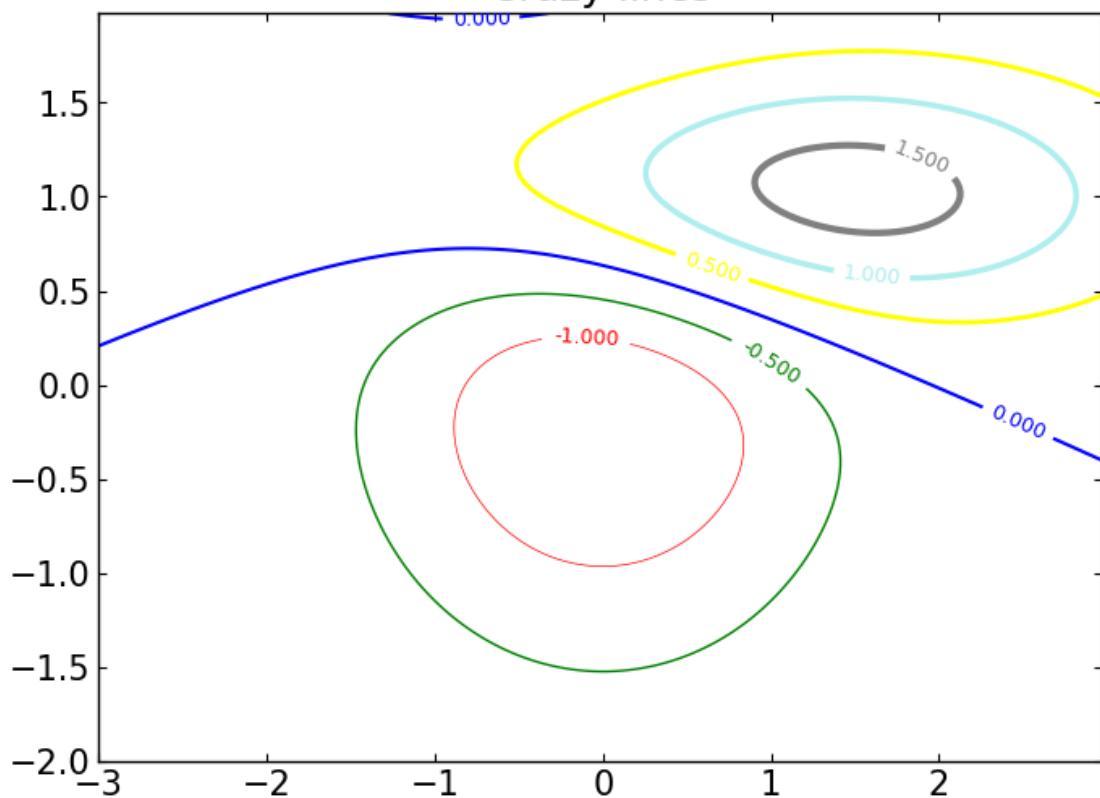
Single color - negative contours dashed

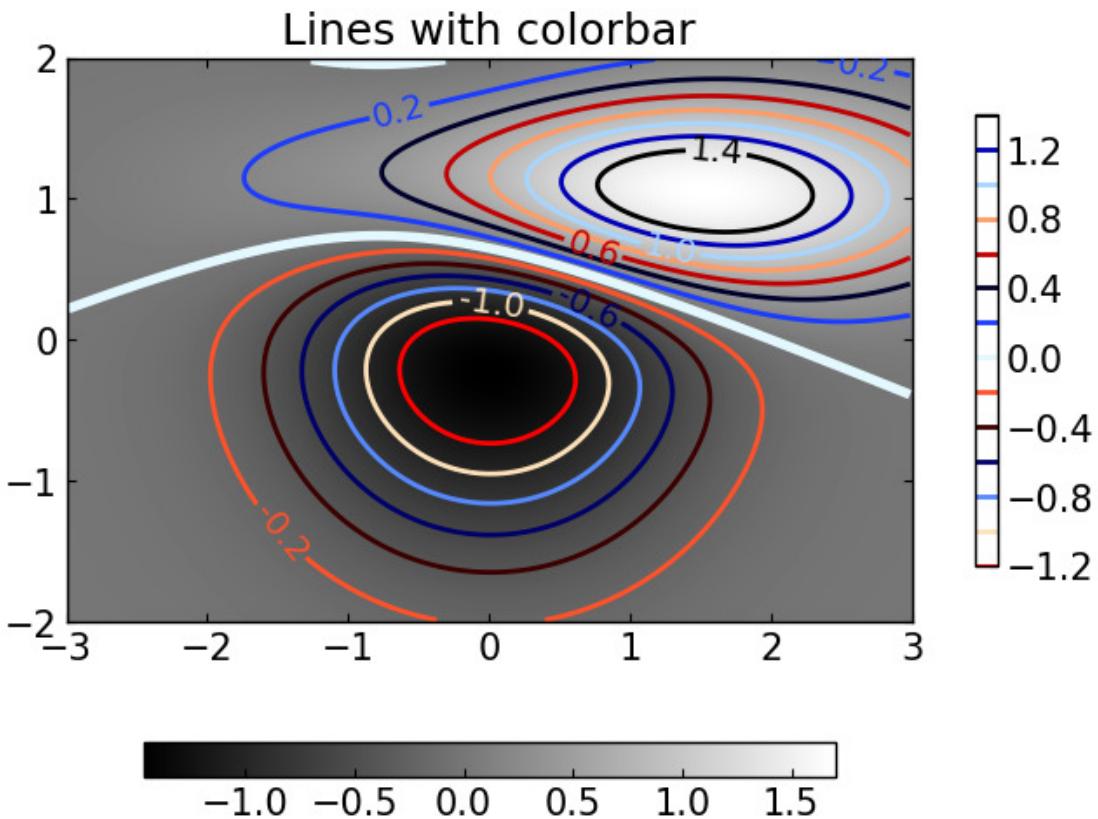


Single color - negative contours solid



Crazy lines





1.7 Logarithmic contour

```
In [22]: """
Demonstration of using logarithmic scale onto the map
"""

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib.mlab import bivariate_normal

"""
Lognorm: Instead of pcolor log10(Z1) you can have colorbars that have
the exponential labels using a norm.
"""
N = 100
X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]

# A low hump with a spike coming out of the top right. Needs to have
# z/colour axis on a log scale so we see both hump and spike. linear
# scale only shows the spike.
```

```

Z1 = bivariate_normal(X, Y, 0.1, 0.2, 1.0, 1.0) + \
    0.1 * bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)

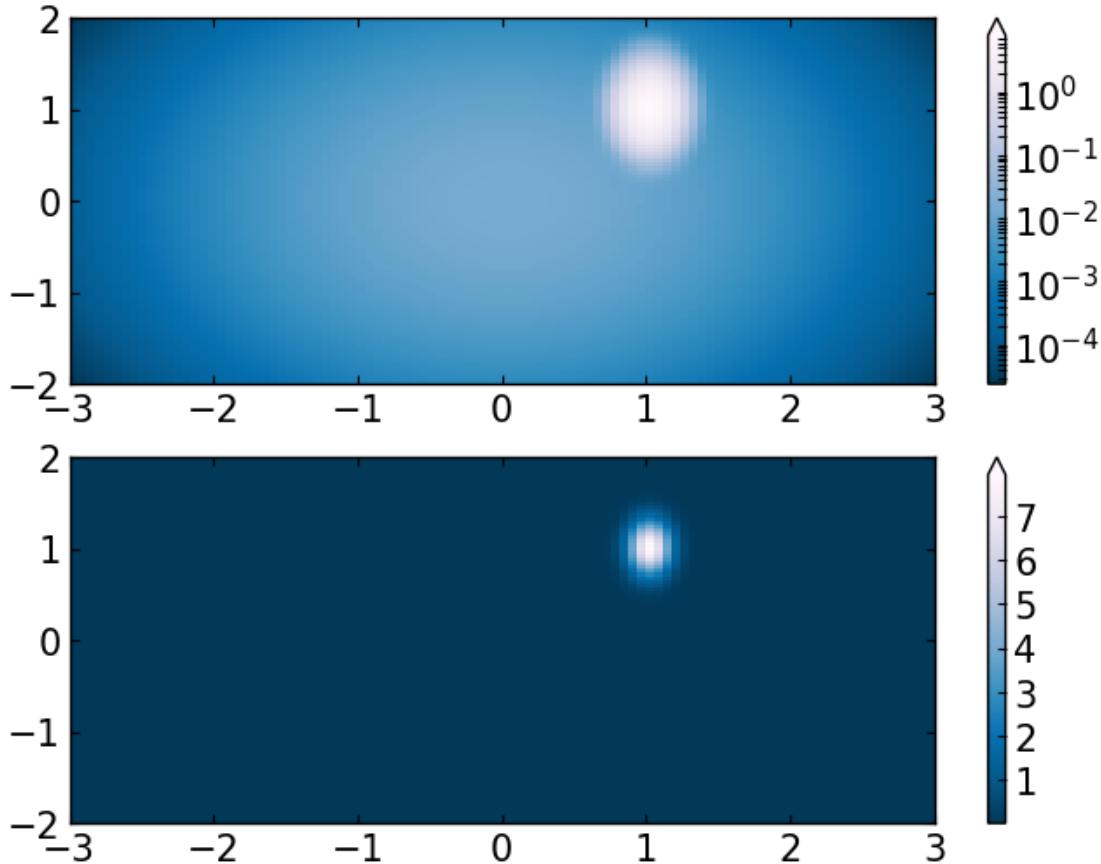
fig, ax = plt.subplots(2, 1)

pcm = ax[0].pcolor(X, Y, Z1,
                     norm=colors.LogNorm(vmin=Z1.min(), vmax=Z1.max()),
                     cmap='PuBu_r')
fig.colorbar(pcm, ax=ax[0], extend='max')

pcm = ax[1].pcolor(X, Y, Z1, cmap='PuBu_r')
fig.colorbar(pcm, ax=ax[1], extend='max')
fig.show()

```

/home/astrodoo/anaconda2/lib/python2.7/site-packages/matplotlib/figure.py:402: UserWarning: matplotlib is currently using a non-GUI backend, "



In [23]: `import numpy as np
import matplotlib.pyplot as plt`

```

import matplotlib.colors as colors
from matplotlib.mlab import bivariate_normal

"""
SymLogNorm: two humps, one negative and one positive, The positive
with 5-times the amplitude. Linearly, you cannot see detail in the
negative hump. Here we logarithmically scale the positive and
negative data separately.

Note that colorbar labels do not come out looking very good.
"""

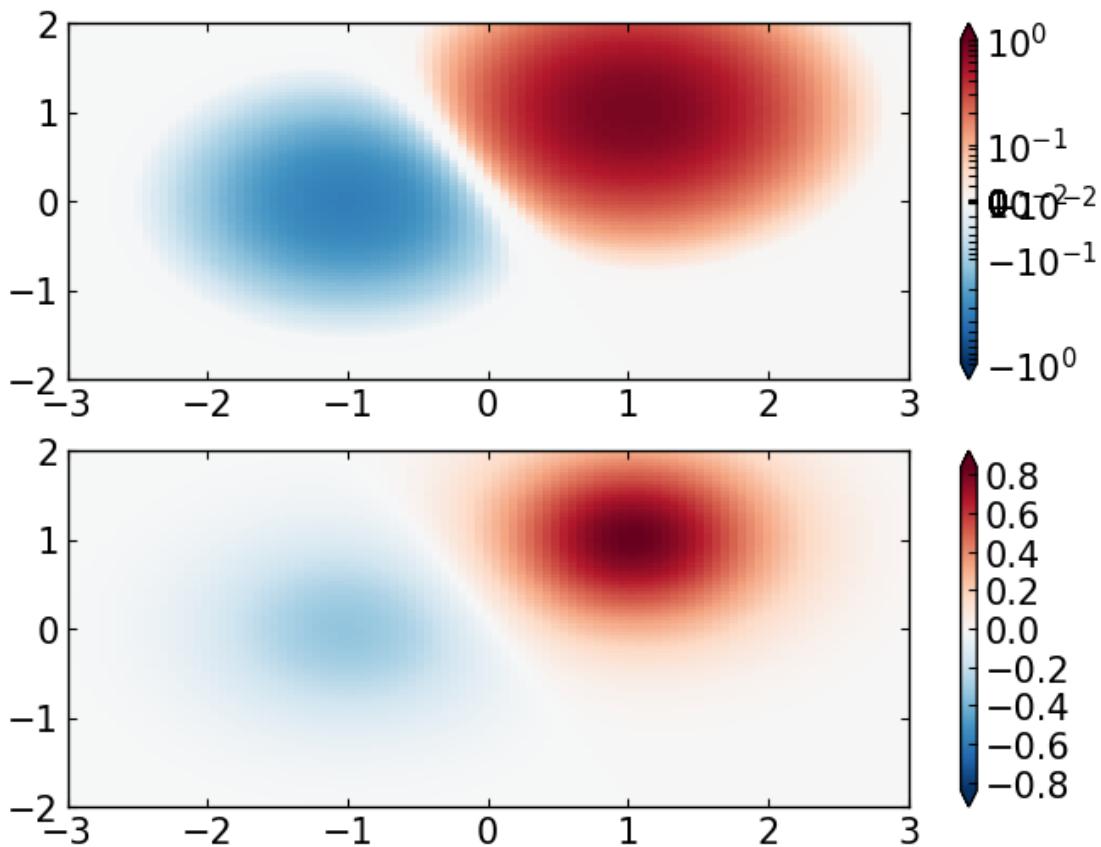
N=100
X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]
Z1 = (bivariate_normal(X, Y, 1., 1., 1.0, 1.0))**2 \
    - 0.4 * (bivariate_normal(X, Y, 1.0, 1.0, -1.0, 0.0))**2
Z1 = Z1/0.03

fig, ax = plt.subplots(2, 1)

pcm = ax[0].pcolormesh(X, Y, Z1,
                       norm=colors.SymLogNorm(linthresh=0.03, linscale=0.03,
                                              vmin=-1.0, vmax=1.0),
                       cmap='RdBu_r')
fig.colorbar(pcm, ax=ax[0], extend='both')

pcm = ax[1].pcolormesh(X, Y, Z1, cmap='RdBu_r', vmin=-np.max(Z1))
fig.colorbar(pcm, ax=ax[1], extend='both')
fig.show()

```



```
In [24]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.colors as colors
        from matplotlib.mlab import bivariate_normal

N = 100
X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]

'''

PowerNorm: Here a power-law trend in X partially obscures a rectified sine wave in Y. We can remove the power law using a PowerNorm.

X, Y = np.mgrid[0:3:complex(0, N), 0:2:complex(0, N)]
Z1 = (1 + np.sin(Y * 10.)) * X**(2.)

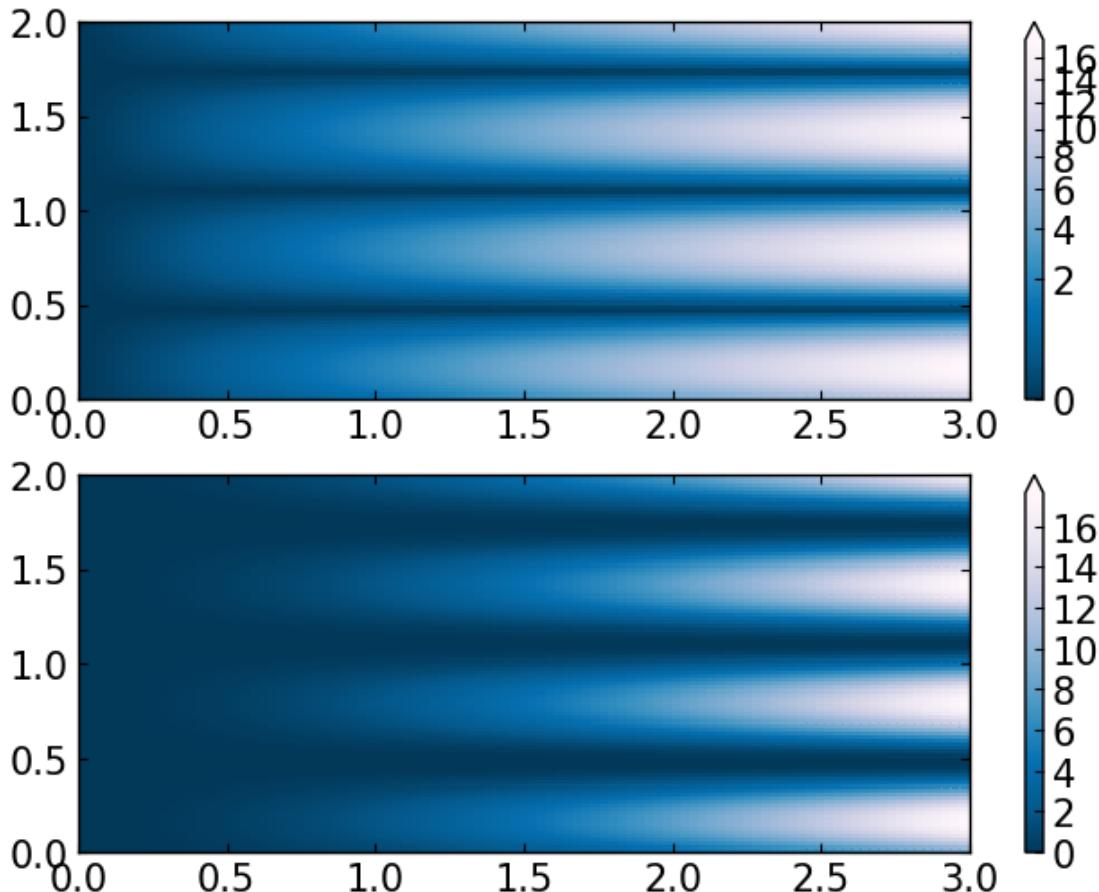
fig, ax = plt.subplots(2, 1)

pcm = ax[0].pcolormesh(X, Y, Z1, norm=colors.PowerNorm(gamma=1./2.),
                       cmap='PuBu_r')
fig.colorbar(pcm, ax=ax[0], extend='max')
```

```

pcm = ax[1].pcolormesh(X, Y, Z1, cmap='PuBu_r')
fig.colorbar(pcm, ax=ax[1], extend='max')
fig.show()

```



```

In [25]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.colors as colors
        from matplotlib.mlab import bivariate_normal

N = 100
X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]
Z1 = (bivariate_normal(X, Y, 1., 1., 1.0, 1.0))**2 \
    - 0.4 * (bivariate_normal(X, Y, 1.0, 1.0, -1.0, 0.0))**2
Z1 = Z1/0.03

    ...

```

BoundaryNorm: For this one you provide the boundaries for your colors, and the Norm puts the first color in between the first pair, the

```

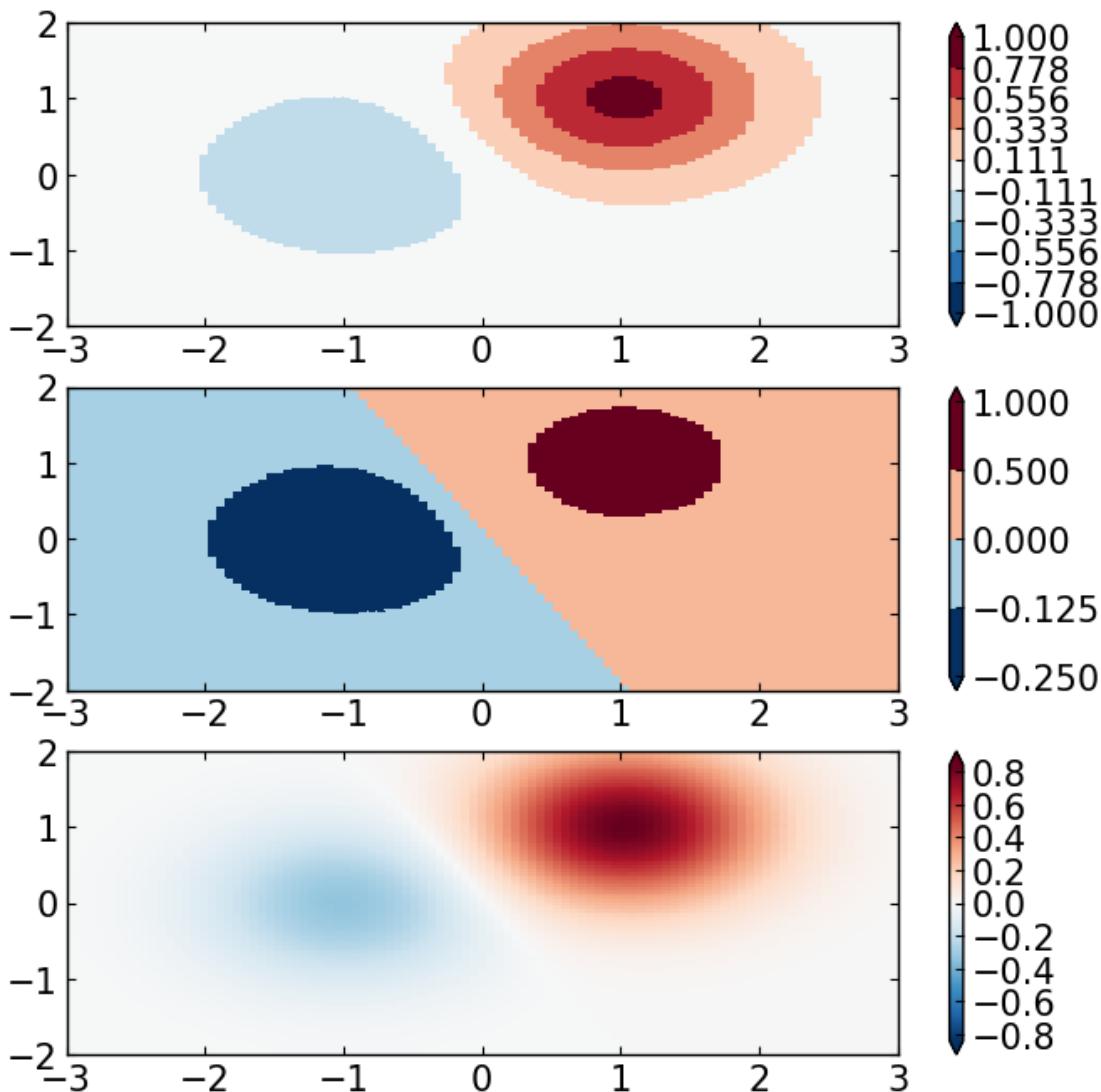
second color between the second pair, etc.
''

fig, ax = plt.subplots(3, 1, figsize=(8, 8))
ax = ax.flatten()
# even bounds gives a contour-like effect
bounds = np.linspace(-1, 1, 10)
norm = colors.BoundaryNorm(boundaries=bounds, ncolors=256)
pcm = ax[0].pcolormesh(X, Y, Z1,
                       norm=norm,
                       cmap='RdBu_r')
fig.colorbar(pcm, ax=ax[0], extend='both', orientation='vertical')

# uneven bounds changes the colormapping:
bounds = np.array([-0.25, -0.125, 0, 0.5, 1])
norm = colors.BoundaryNorm(boundaries=bounds, ncolors=256)
pcm = ax[1].pcolormesh(X, Y, Z1, norm=norm, cmap='RdBu_r')
fig.colorbar(pcm, ax=ax[1], extend='both', orientation='vertical')

pcm = ax[2].pcolormesh(X, Y, Z1, cmap='RdBu_r', vmin=-np.max(Z1))
fig.colorbar(pcm, ax=ax[2], extend='both', orientation='vertical')
fig.show()

```



```
In [26]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.colors as colors
        from matplotlib.mlab import bivariate_normal

N = 100
'''

Custom Norm: An example with a customized normalization. This one
uses the example above, and normalizes the negative data differently
from the positive.
'''

X, Y = np.mgrid[-3:3:complex(0, N), -2:2:complex(0, N)]
Z1 = (bivariate_normal(X, Y, 1., 1., 1.0, 1.0))**2 \
```

```

- 0.4 * (bivariate_normal(X, Y, 1.0, 1.0, -1.0, 0.0))**2
Z1 = Z1/0.03

# Example of making your own norm. Also see matplotlib.colors.
# From Joe Kington: This one gives two different linear ramps:

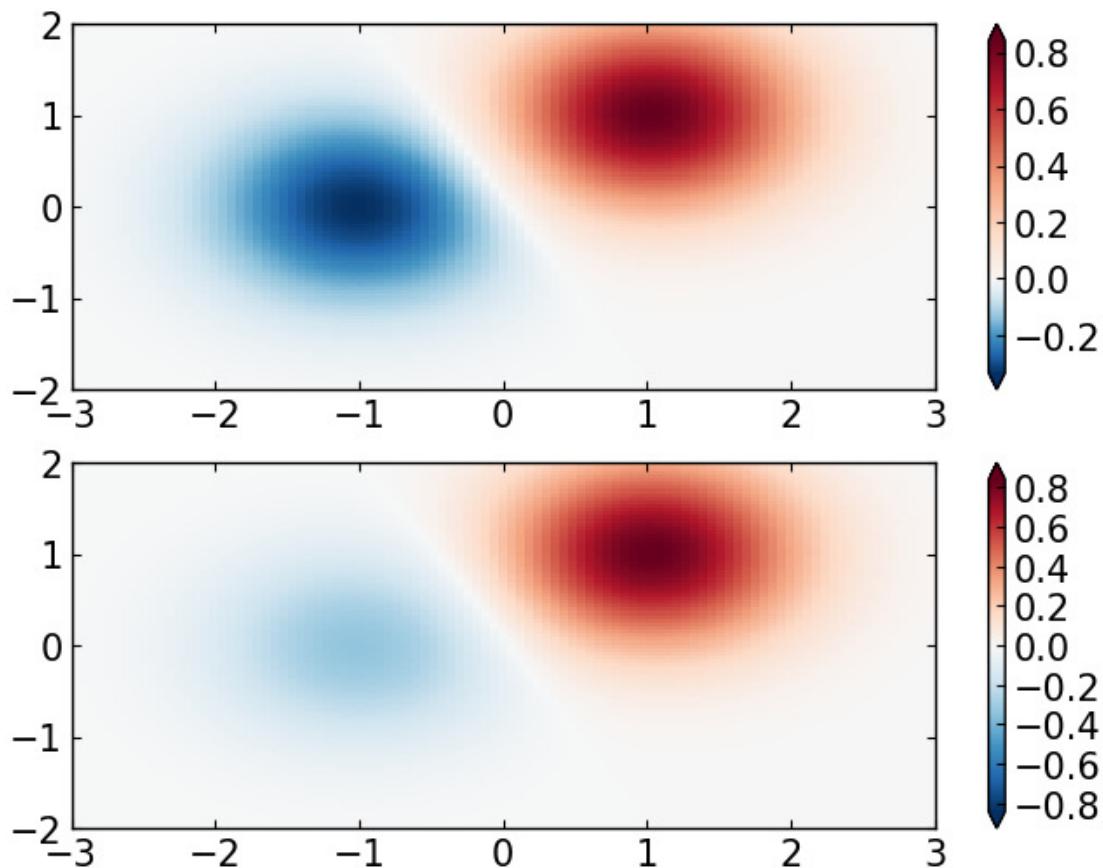
class MidpointNormalize(colors.Normalize):
    def __init__(self, vmin=None, vmax=None, midpoint=None, clip=False):
        self.midpoint = midpoint
        colors.Normalize.__init__(self, vmin, vmax, clip)

    def __call__(self, value, clip=None):
        # I'm ignoring masked values and all kinds of edge cases to make a
# simple example...
        x, y = [self.vmin, self.midpoint, self.vmax], [0, 0.5, 1]
        return np.ma.masked_array(np.interp(value, x, y))
#####
fig, ax = plt.subplots(2, 1)

pcm = ax[0].pcolormesh(X, Y, Z1,
                       norm=MidpointNormalize(midpoint=0.),
                       cmap='RdBu_r')
fig.colorbar(pcm, ax=ax[0], extend='both')

pcm = ax[1].pcolormesh(X, Y, Z1, cmap='RdBu_r', vmin=-np.max(Z1))
fig.colorbar(pcm, ax=ax[1], extend='both')
fig.show()

```



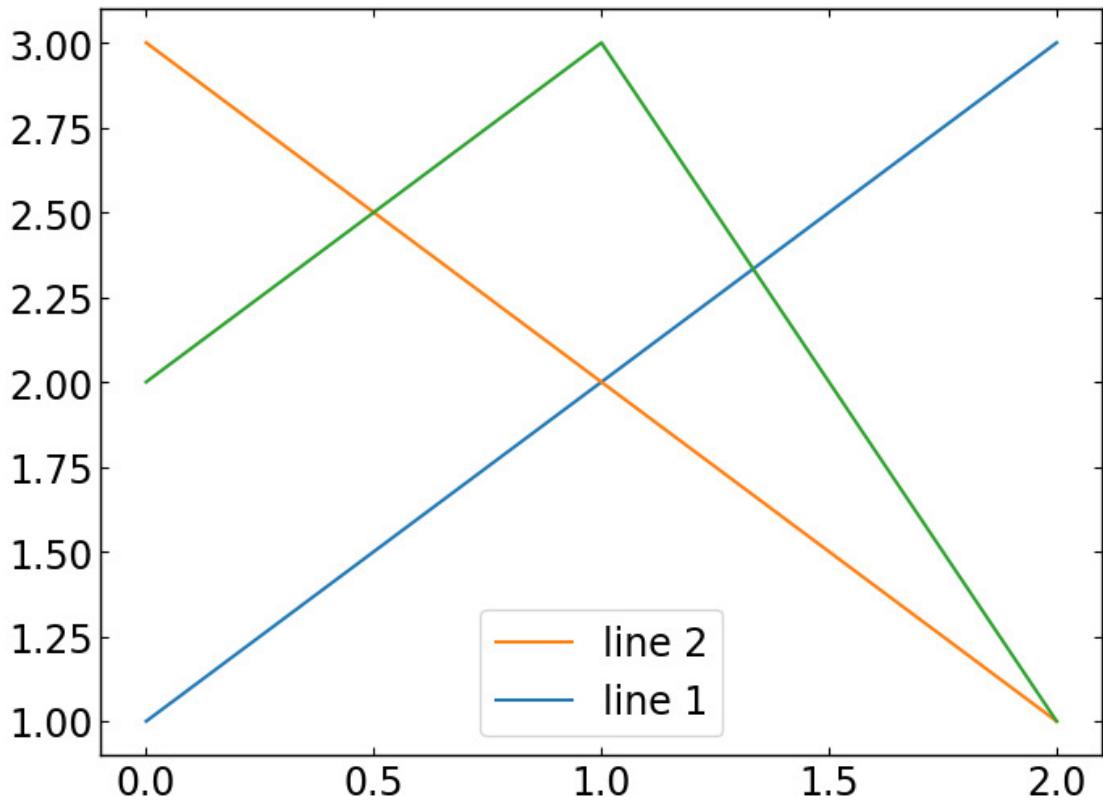
1.8 Legend

In [2]: *""" manage the label sequence in legend: 1st method """*

```
import matplotlib.pyplot as plt
%matplotlib inline

p1, = plt.plot([1,2,3])
p2, = plt.plot([3,2,1])
p3, = plt.plot([2,3,1])
plt.legend([p2,p1],['line 2','line 1'])
```

Out[2]: <matplotlib.legend.Legend at 0x7f4cb9bcd10>

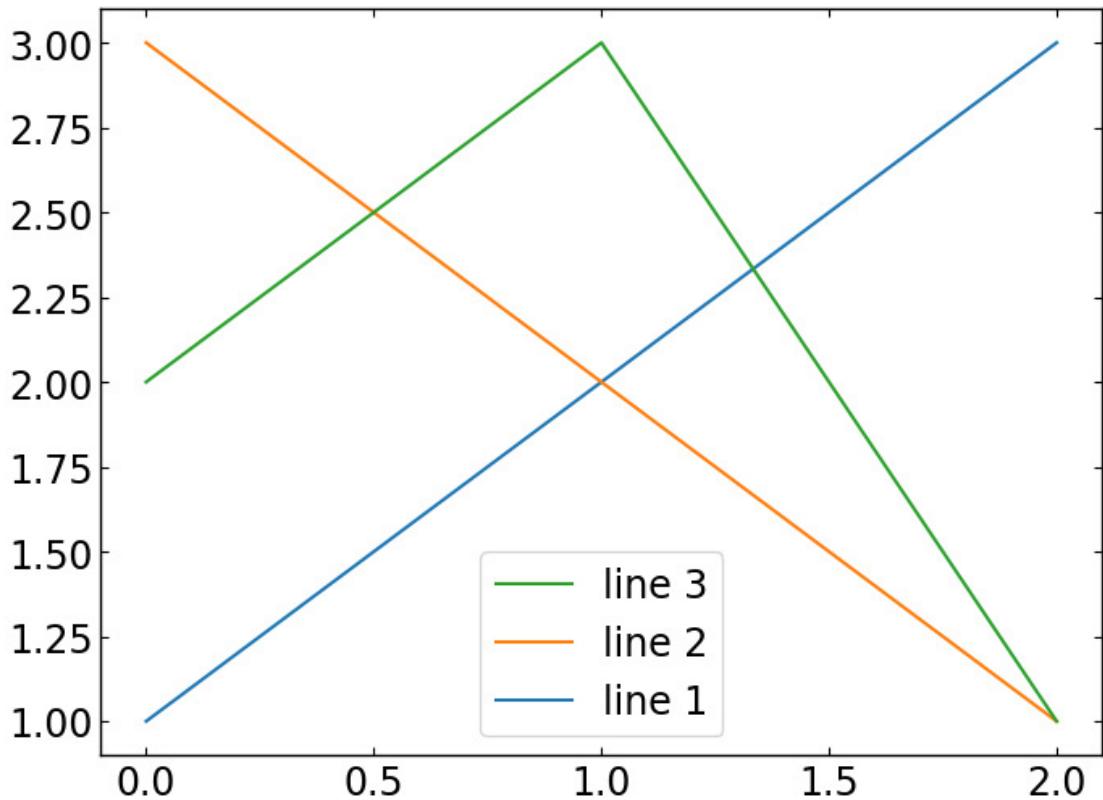


In [3]: *""" manage the label sequence in legend: 2nd method """*

```
ax = plt.subplot(111)
ax.plot([1,2,3], label="line 1")
ax.plot([3,2,1], label="line 2")
ax.plot([2,3,1], label="line 3")

handles, labels = ax.get_legend_handles_labels()
ax.legend(handles[::-1], labels[::-1])
```

Out[3]: <matplotlib.legend.Legend at 0x7f0e80158f10>

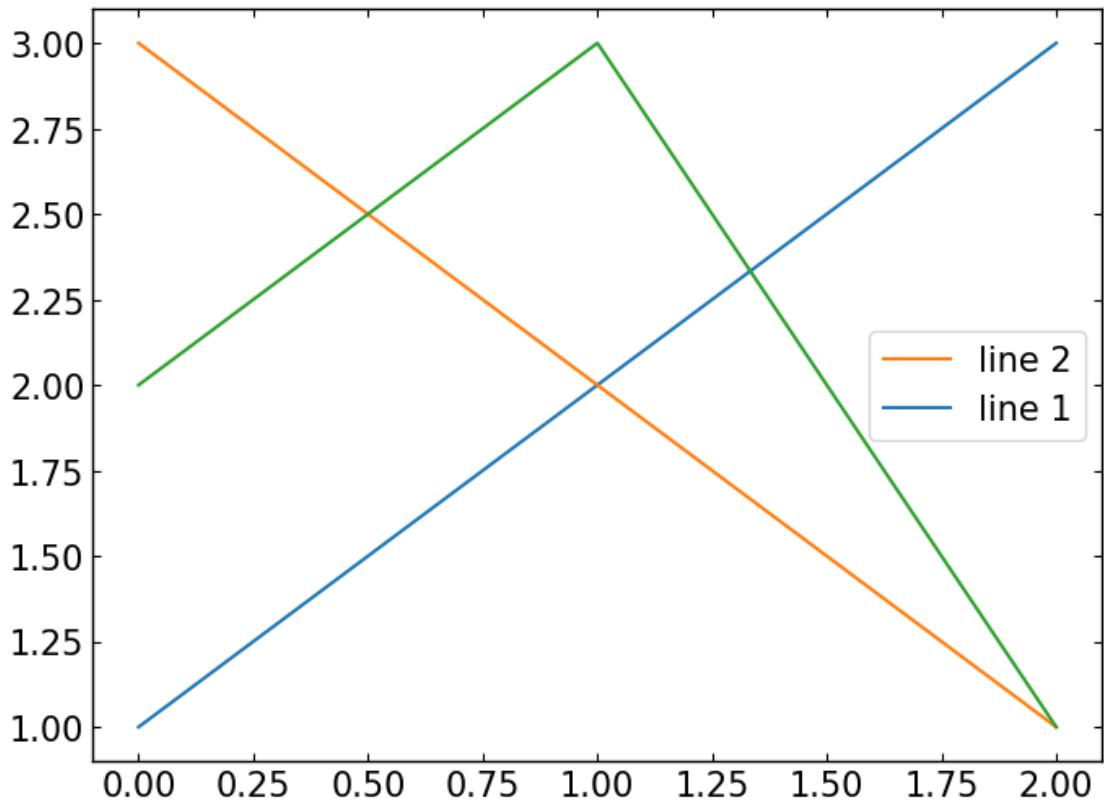


In [29]: *""" manage the label sequence in legend: 2nd-2 method """*

```
ax = plt.subplot(111)
p1, = ax.plot([1,2,3], label="line 1")
p2, = ax.plot([3,2,1], label="line 2")
p3, = ax.plot([2,3,1], label="line 3")

handles, labels = ax.get_legend_handles_labels()
ax.legend([handles[1],handles[0]], [labels[1],labels[0]])
```

Out[29]: <matplotlib.legend.Legend at 0x7f41ae850310>

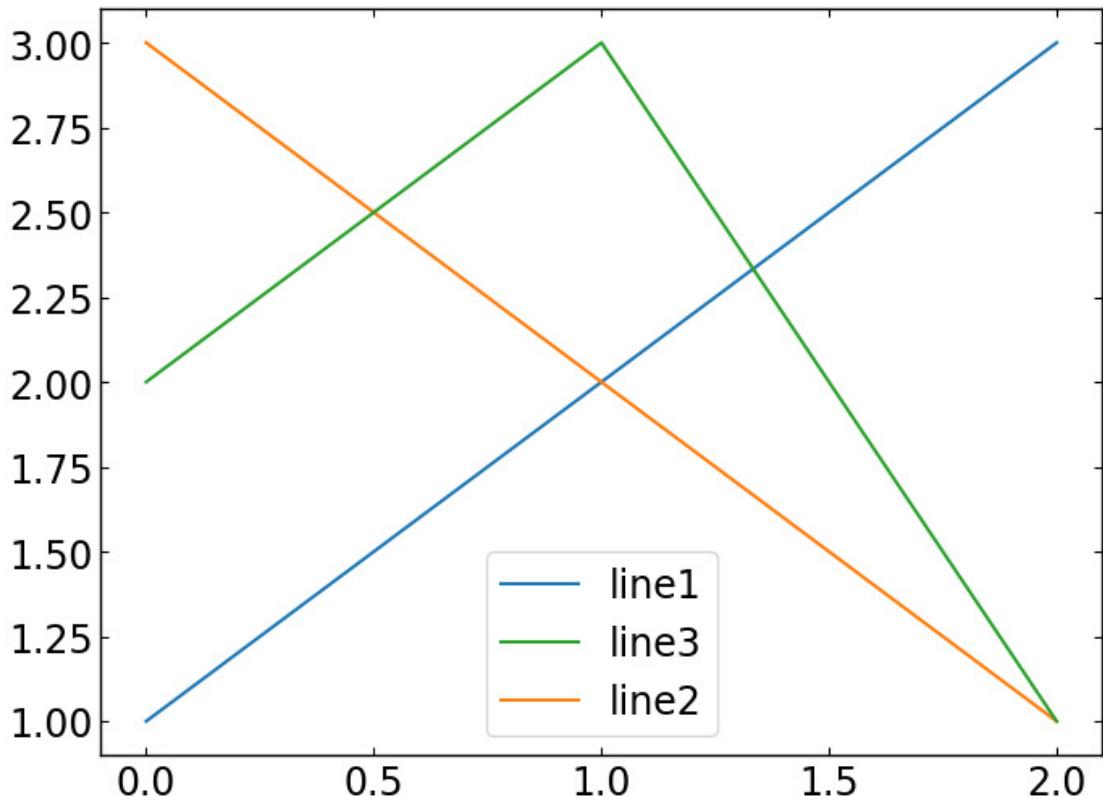


In [2]: *""" manage the label sequence in legend: 3rd method """*

```
import matplotlib.pyplot as plt
%matplotlib inline

p1, = plt.plot([1,2,3],label='line1')
p2, = plt.plot([3,2,1],label='line2')
p3, = plt.plot([2,3,1],label='line3')
plt.legend(handles=[p1,p3,p2])
```

Out[2]: <matplotlib.legend.Legend at 0x7f0e801ddcd0>



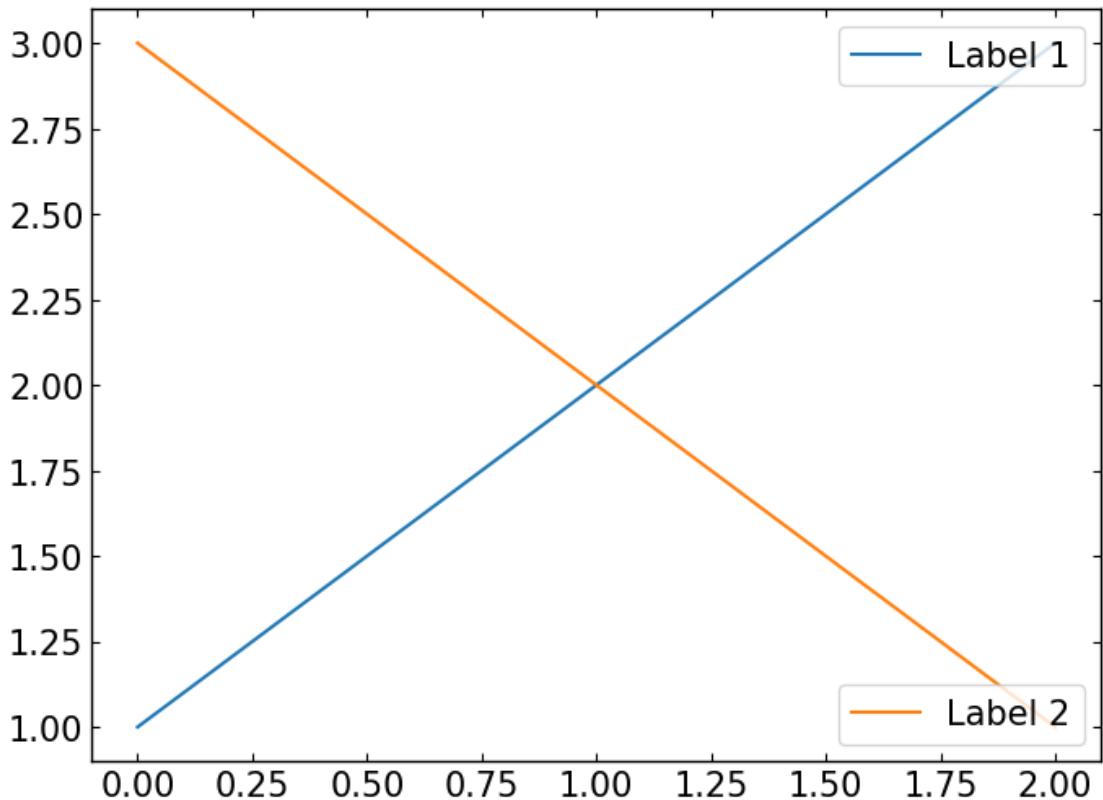
```
In [30]: import matplotlib.pyplot as plt

p1, = plt.plot([1,2,3], label='test1')
p2, = plt.plot([3,2,1], label='test2')

l1 = plt.legend([p1], ["Label 1"], loc=1)
l2 = plt.legend([p2], ["Label 2"], loc=4) # this removes l1 from the axes

# add l1 as a separate artist to the axes
plt.gca().add_artist(l1)

plt.show()
```



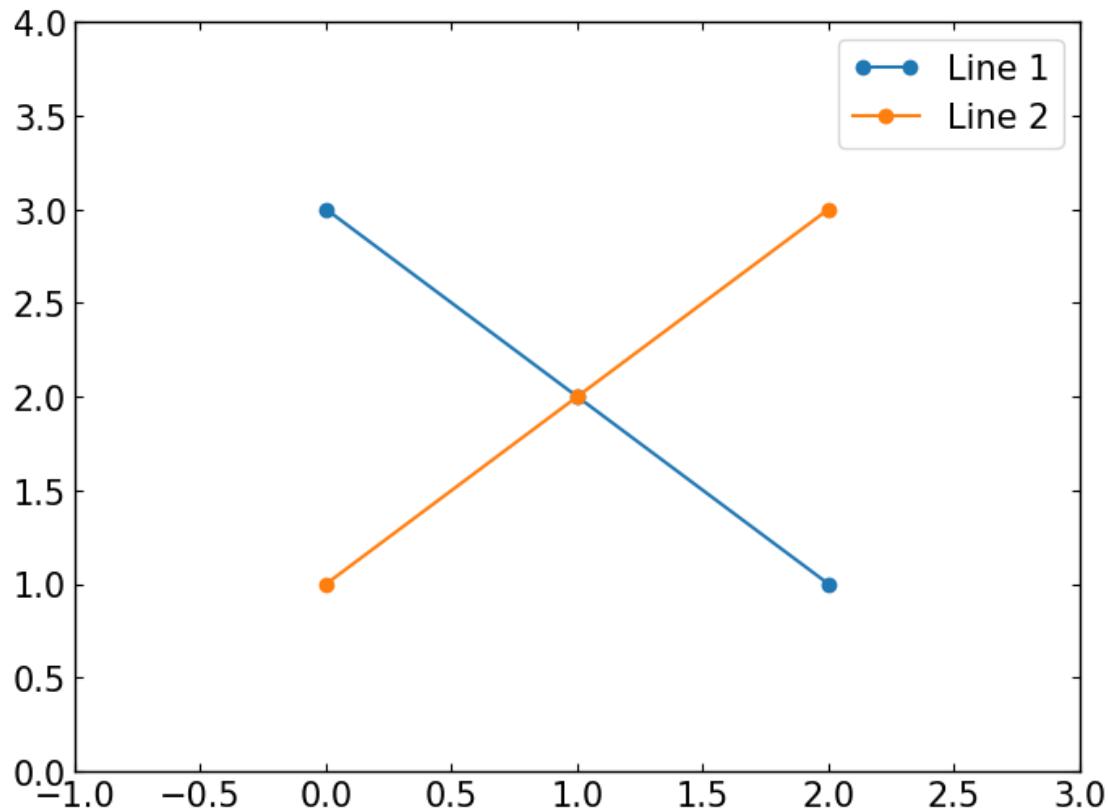
```
In [31]: import matplotlib.pyplot as plt
        from matplotlib.legend_handler import HandlerLine2D as hdl2D

        line1, = plt.plot([3,2,1], marker='o', label='Line 1')
        line2, = plt.plot([1,2,3], marker='o', label='Line 2')

        plt.axis([-1.,3.,0.,4.])

        plt.legend(handler_map={line1: hdl2D(numpoints=2)})

        plt.show()
```



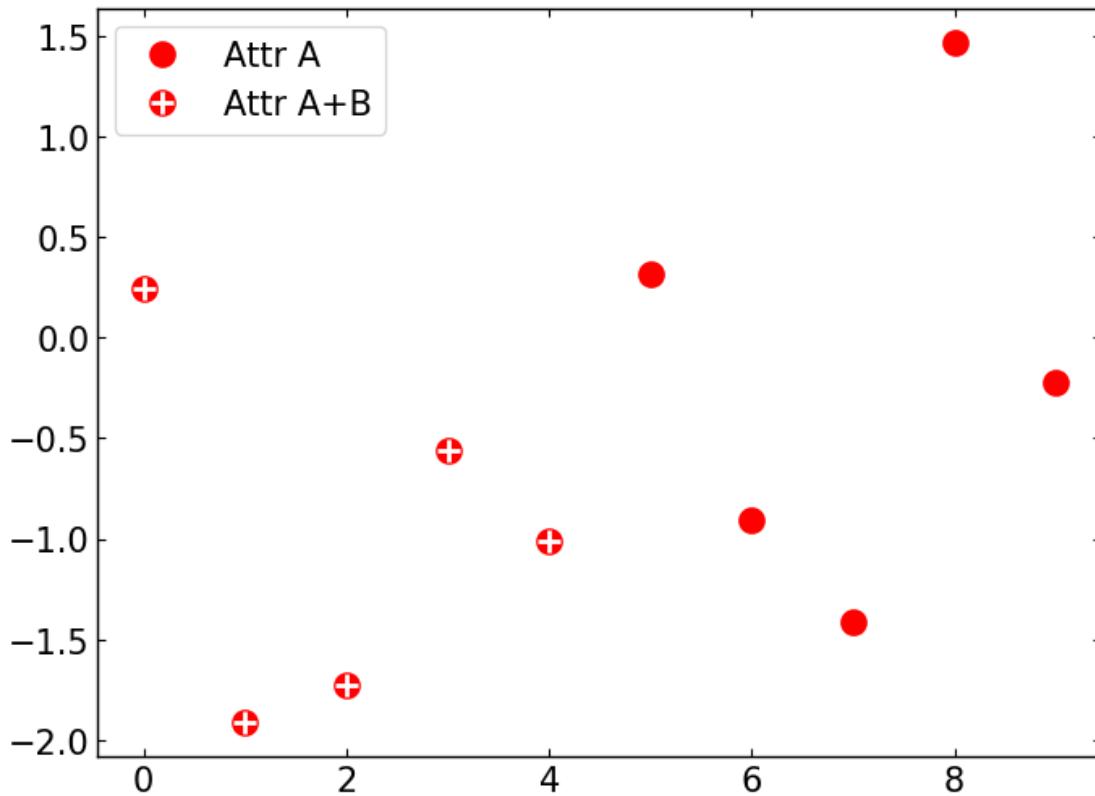
```
In [32]: import matplotlib.pyplot as plt
        import numpy as np

z = np.random.randn(10)

p1, = plt.plot(z,'ro', ms=10, mfc='r', mew=2, mec='r') # red filled circle
p2, = plt.plot(z[:5],'w+', ms=10, mfc='w',mec='w') # white cross

plt.legend([p1,(p1,p2)], ['Attr A', 'Attr A+B'])

plt.show()
```



```
In [34]: import numpy as np
        import matplotlib.pyplot as plt

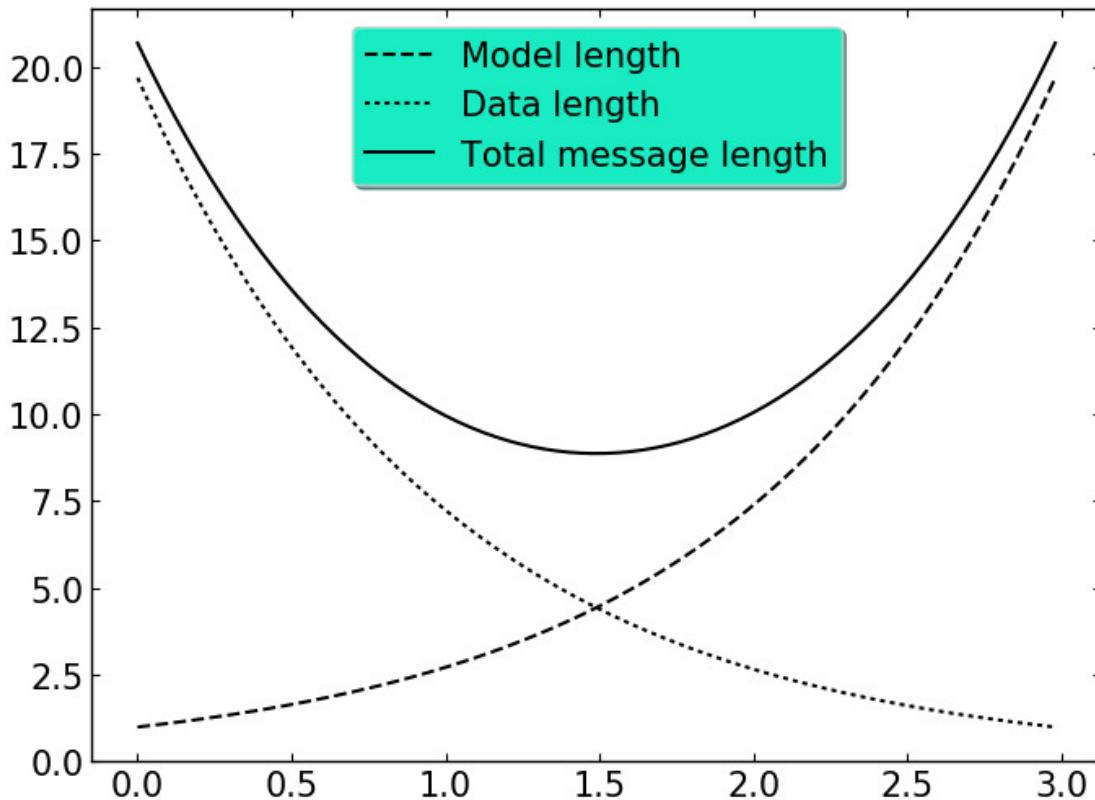
        # Make some fake data.
        a = b = np.arange(0, 3, .02)
        c = np.exp(a)
        d = c[::-1]

        # Create plots with pre-defined labels.
        plt.plot(a, c, 'k--', label='Model length')
        plt.plot(a, d, 'k:', label='Data length')
        plt.plot(a, c + d, 'k', label='Total message length')

        legend = plt.legend(loc='upper center', shadow=True, fontsize='medium')

        # Put a nicer background color on the legend.
        legend.get_frame().set_facecolor('#00FFCC')

        plt.show()
```



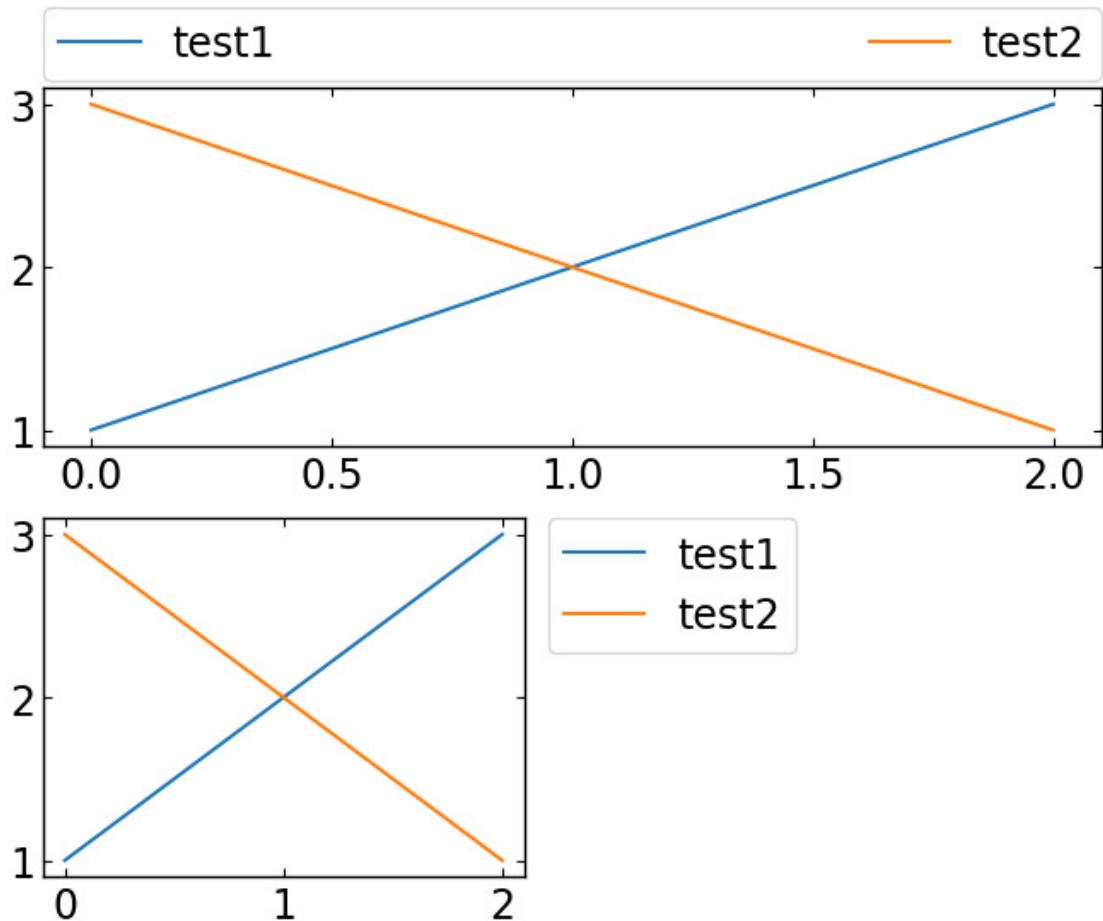
```
In [1]: import matplotlib.pyplot as plt
%matplotlib inline

plt.subplot(211)
plt.plot([1, 2, 3], label="test1")
plt.plot([3, 2, 1], label="test2")
# Place a legend above this subplot, expanding itself to
# fully use the given bounding box.
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)

plt.subplot(223)
plt.plot([1, 2, 3], label="test1")
plt.plot([3, 2, 1], label="test2")

# Place a legend to the right of this smaller subplot.
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[1]: <matplotlib.legend.Legend at 0x7ff4d0766450>

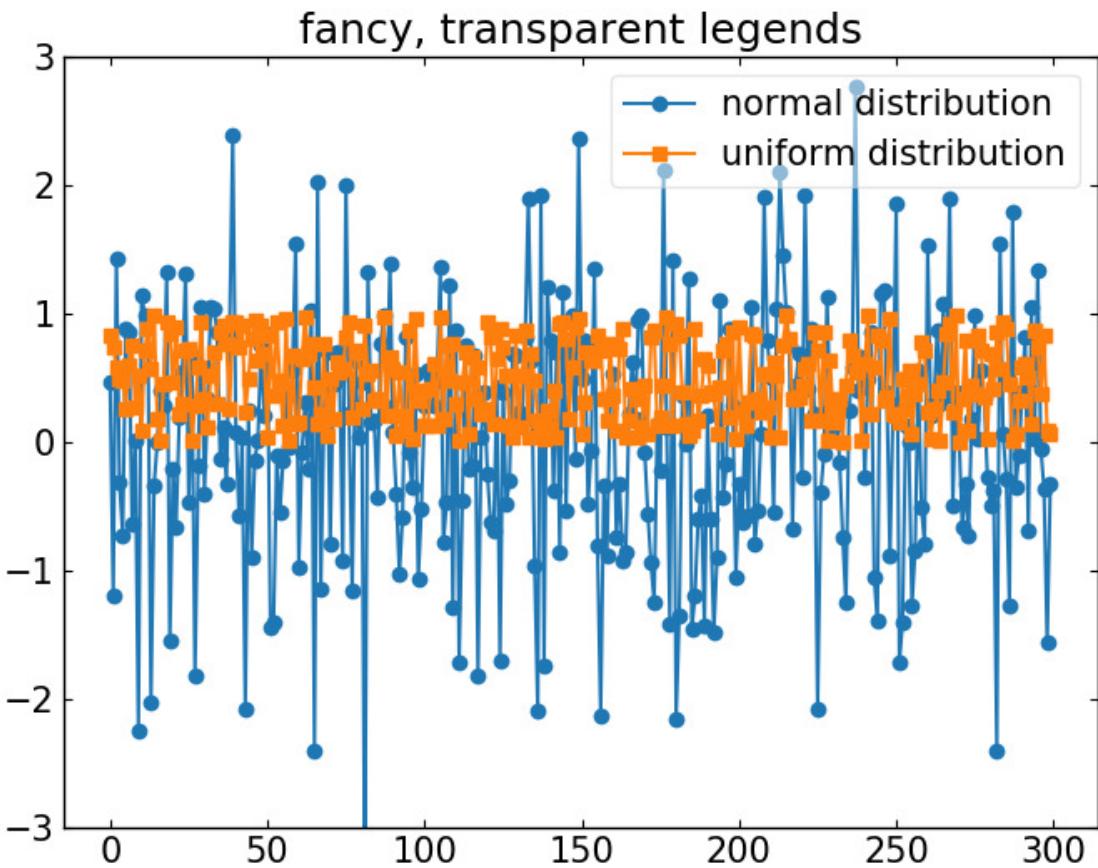


```
In [35]: import matplotlib.pyplot as plt
        import numpy as np

        np.random.seed(1234)
        fig, ax = plt.subplots(1)
        ax.plot(np.random.randn(300), 'o-', label='normal distribution')
        ax.plot(np.random.rand(300), 's-', label='uniform distribution')
        ax.set_xlim(-3, 3)
        ax.legend(loc='best', fancybox=True, framealpha=0.5)

        ax.set_title('fancy, transparent legends')
```

Out[35]: <matplotlib.text.Text at 0x7f41ad939b10>

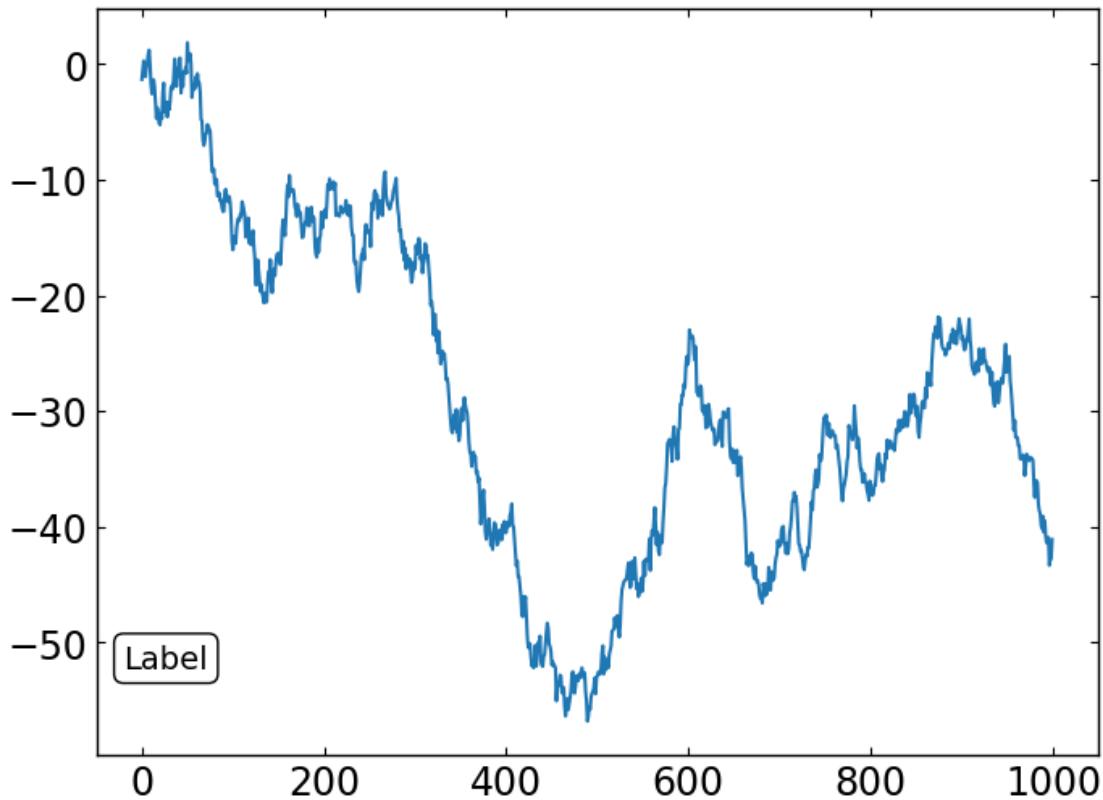


```
In [11]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        data = np.random.normal(0, 1, 1000).cumsum()

        fig, ax = plt.subplots()
        ax.plot(data)
        ax.annotate('Label', xy=(50, 50), xycoords='axes points',
                    size=14, ha='right', va='top',
                    bbox=dict(boxstyle='round', fc='w'))
```

```
Out[11]: <matplotlib.text.Annotation at 0x7f4cb8102590>
```



```
In [14]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

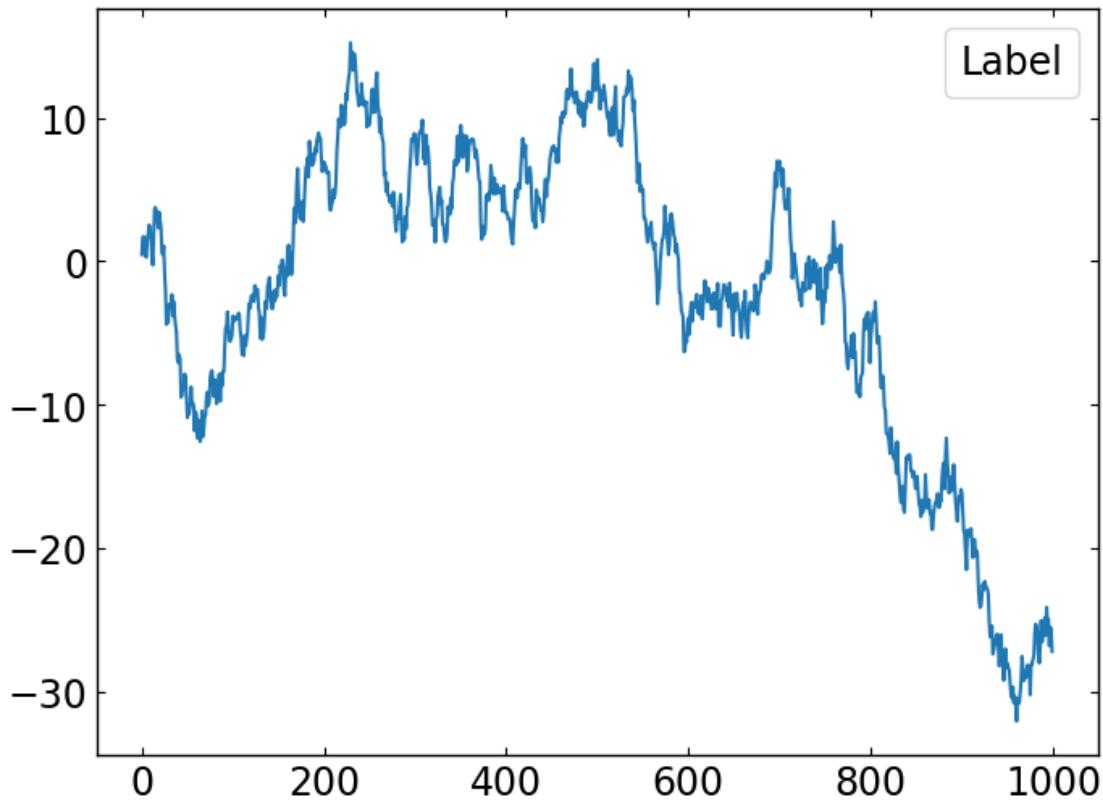
        data = np.random.normal(0, 1, 1000).cumsum()

        fig, ax = plt.subplots()
        ax.plot(data, label='Label')

        leg = ax.legend(handlelength=0, handletextpad=0, fancybox=True)

        for item in leg.legendHandles:
            item.set_visible(False)

        # plt.legend(markerscale=0)
```



1.9 Zoom effect

```
In [36]: from matplotlib.transforms import Bbox, TransformedBbox, \
blended_transform_factory

from mpl_toolkits.axes_grid1.inset_locator import BboxPatch, BboxConnector,\ 
BboxConnectorPatch

def connect_bbox(bbox1, bbox2,
                 loc1a, loc2a, loc1b, loc2b,
                 prop_lines, prop_patches=None):
    if prop_patches is None:
        prop_patches = prop_lines.copy()
        prop_patches["alpha"] = prop_patches.get("alpha", 1)*0.2

    c1 = BboxConnector(bbox1, bbox2, loc1=loc1a, loc2=loc2a, **prop_lines)
    c1.set_clip_on(False)
    c2 = BboxConnector(bbox1, bbox2, loc1=loc1b, loc2=loc2b, **prop_lines)
    c2.set_clip_on(False)
```

```

bbox_patch1 = BboxPatch(bbox1, **prop_patches)
bbox_patch2 = BboxPatch(bbox2, **prop_patches)

p = BboxConnectorPatch(bbox1, bbox2,
                       # loc1a=3, loc2a=2, loc1b=4, loc2b=1,
                       loc1a=loc1a, loc2a=loc2a, loc1b=loc1b, loc2b=loc2b,
                       **prop_patches)
p.set_clip_on(False)

return c1, c2, bbox_patch1, bbox_patch2, p


def zoom_effect01(ax1, ax2, xmin, xmax, **kwargs):
    """
    ax1 : the main axes
    ax1 : the zoomed axes
    (xmin, xmax) : the limits of the colored area in both plot axes.

    connect ax1 & ax2. The x-range of (xmin, xmax) in both axes will
    be marked. The keywords parameters will be used to create
    patches.

    """
    trans1 = blended_transform_factory(ax1.transData, ax1.transAxes)
    trans2 = blended_transform_factory(ax2.transData, ax2.transAxes)

    bbox = Bbox.from_extents(xmin, 0, xmax, 1)

    mybbox1 = TransformedBbox(bbox, trans1)
    mybbox2 = TransformedBbox(bbox, trans2)

    prop_patches = kwargs.copy()
    prop_patches["ec"] = "none"
    prop_patches["alpha"] = 0.2

    c1, c2, bbox_patch1, bbox_patch2, p = \
        connect_bbox(mybbox1, mybbox2,
                     loc1a=3, loc2a=2, loc1b=4, loc2b=1,
                     prop_lines=kwargs, prop_patches=prop_patches)

    ax1.add_patch(bbox_patch1)
    ax2.add_patch(bbox_patch2)
    ax2.add_patch(c1)
    ax2.add_patch(c2)
    ax2.add_patch(p)

return c1, c2, bbox_patch1, bbox_patch2, p

```

```

def zoom_effect02(ax1, ax2, **kwargs):
    """
    ax1 : the main axes
    ax1 : the zoomed axes

    Similar to zoom_effect01. The xmin & xmax will be taken from the
    ax1.viewLim.
    """
    tt = ax1.transScale + (ax1.transLimits + ax2.transAxes)
    trans = blended_transform_factory(ax2.transData, tt)

    mybbox1 = ax1.bbox
    mybbox2 = TransformedBbox(ax1.viewLim, trans)

    prop_patches = kwargs.copy()
    prop_patches["ec"] = "none"
    prop_patches["alpha"] = 0.2

    c1, c2, bbox_patch1, bbox_patch2, p = \
        connect_bbox(mybbox1, mybbox2,
                     loc1a=3, loc2a=2, loc1b=4, loc2b=1,
                     prop_lines=kwargs, prop_patches=prop_patches)

    ax1.add_patch(bbox_patch1)
    ax2.add_patch(bbox_patch2)
    ax2.add_patch(c1)
    ax2.add_patch(c2)
    ax2.add_patch(p)

    return c1, c2, bbox_patch1, bbox_patch2, p


import matplotlib.pyplot as plt

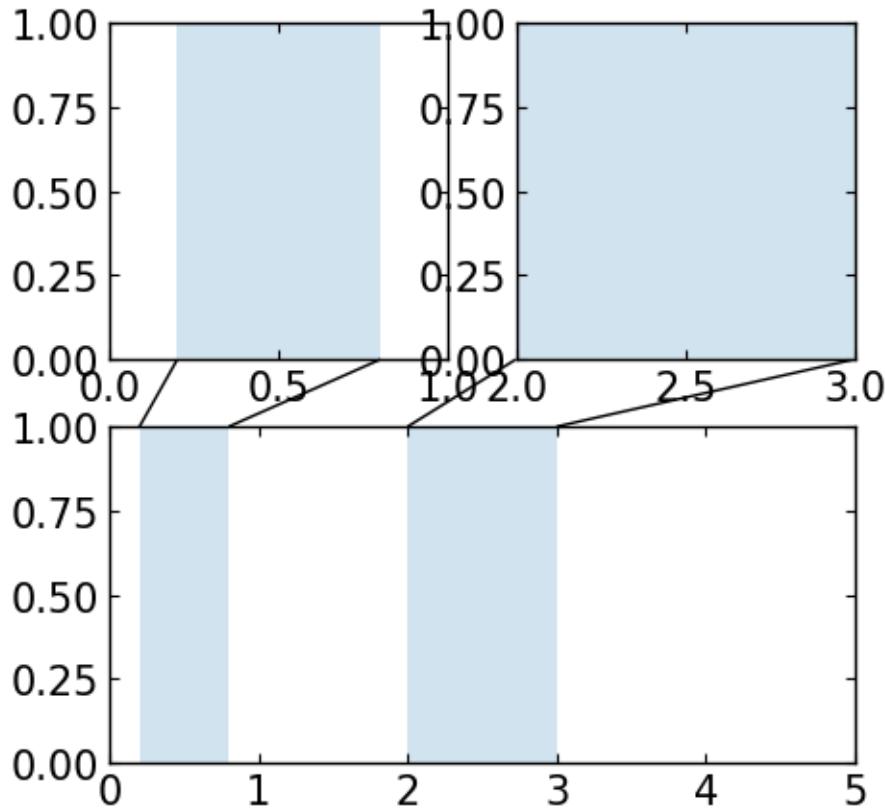
plt.figure(1, figsize=(5, 5))
ax1 = plt.subplot(221)
ax2 = plt.subplot(212)
ax2.set_xlim(0, 1)
ax2.set_xlim(0, 5)
zoom_effect01(ax1, ax2, 0.2, 0.8)

ax1 = plt.subplot(222)
ax1.set_xlim(2, 3)
ax2.set_xlim(0, 5)

```

```
zoom_effect02(ax1, ax2)
```

```
plt.show()
```



1.10 Scatter

```
In [37]: """
```

```
Demo of scatter plot with varying marker colors and sizes.
```

```
"""
```

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
```

```
# Load a numpy record array from yahoo csv data with fields date,
# open, close, volume, adj_close from the mpl-data/example directory.
# The record array stores python datetime.date as an object array in
# the date column
```

```
datafile = cbook.get_sample_data('goog.npy')
```

```
try:
```

```
    # Python3 cannot load python2 .npy files with datetime(object) arrays
    # unless the encoding is set to bytes. However this option was
```

```

# not added until numpy 1.10 so this example will only work with
# python 2 or with numpy 1.10 and later
price_data = np.load(datafile, encoding='bytes').view(np.recarray)
except TypeError:
    price_data = np.load(datafile).view(np.recarray)
price_data = price_data[-250:] # get the most recent 250 trading days

delta1 = np.diff(price_data.adj_close)/price_data.adj_close[:-1]

# Marker size in units of points^2
volume = (15 * price_data.volume[:-2] / price_data.volume[0])**2
close = 0.003 * price_data.close[:-2] / 0.003 * price_data.open[:-2]

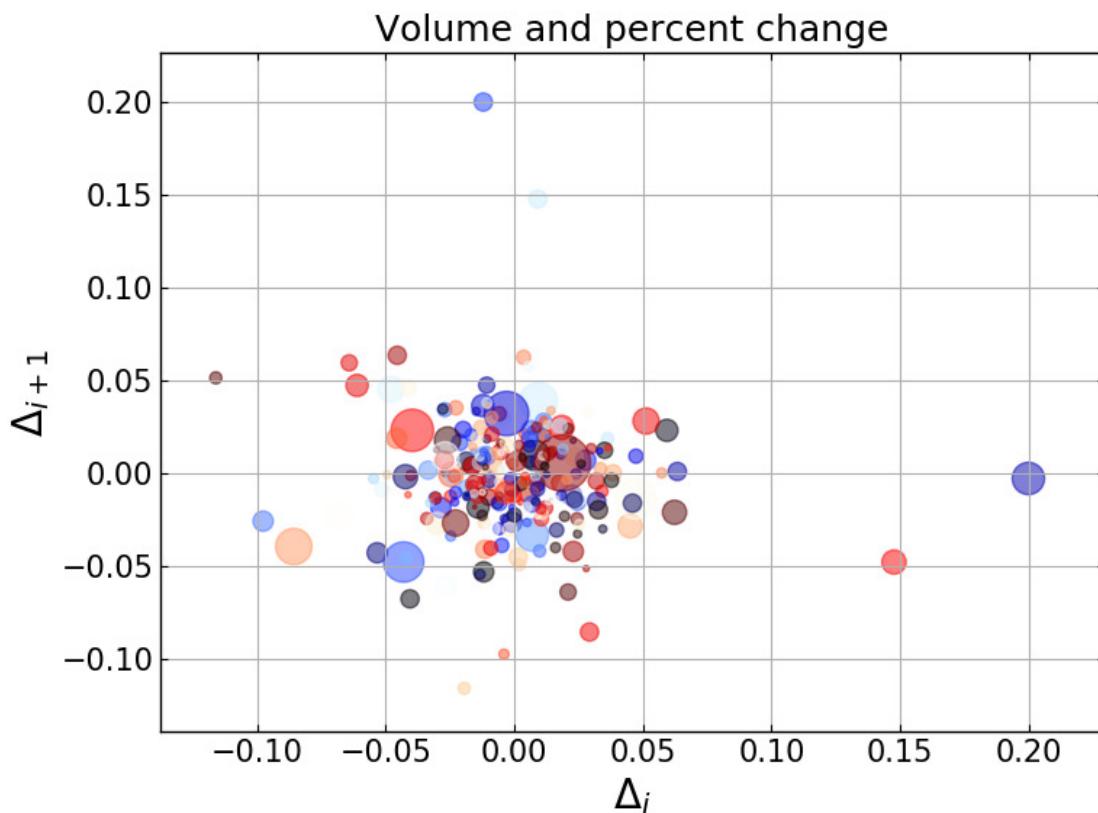
fig, ax = plt.subplots()
ax.scatter(delta1[:-1], delta1[1:], c=close, s=volume, alpha=0.5)

ax.set_xlabel(r'$\Delta_i$', fontsize=20)
ax.set_ylabel(r'$\Delta_{i+1}$', fontsize=20)
ax.set_title('Volume and percent change')

ax.grid(True)
fig.tight_layout()

plt.show()

```

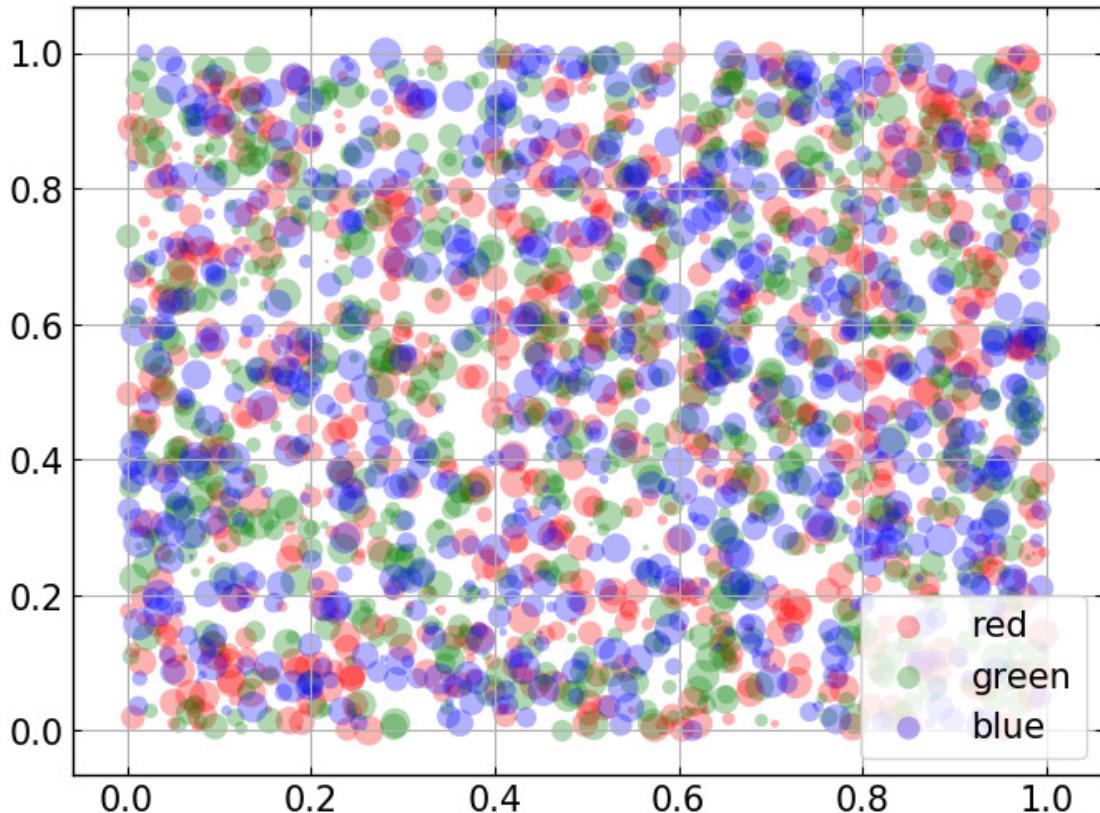


```
In [38]: import matplotlib.pyplot as plt
        from numpy.random import rand

        for color in ['red','green','blue']:
            n = 750
            x, y = rand(2, n)
            scale = 200. * rand(n)
            plt.scatter(x, y, c=color, s=scale, label=color,
                        alpha=0.3, edgecolors='none')

        plt.legend()
        plt.grid(True)

        plt.show()
```



1.11 Fill

```
In [39]: """
Simple demo of the fill function.
```

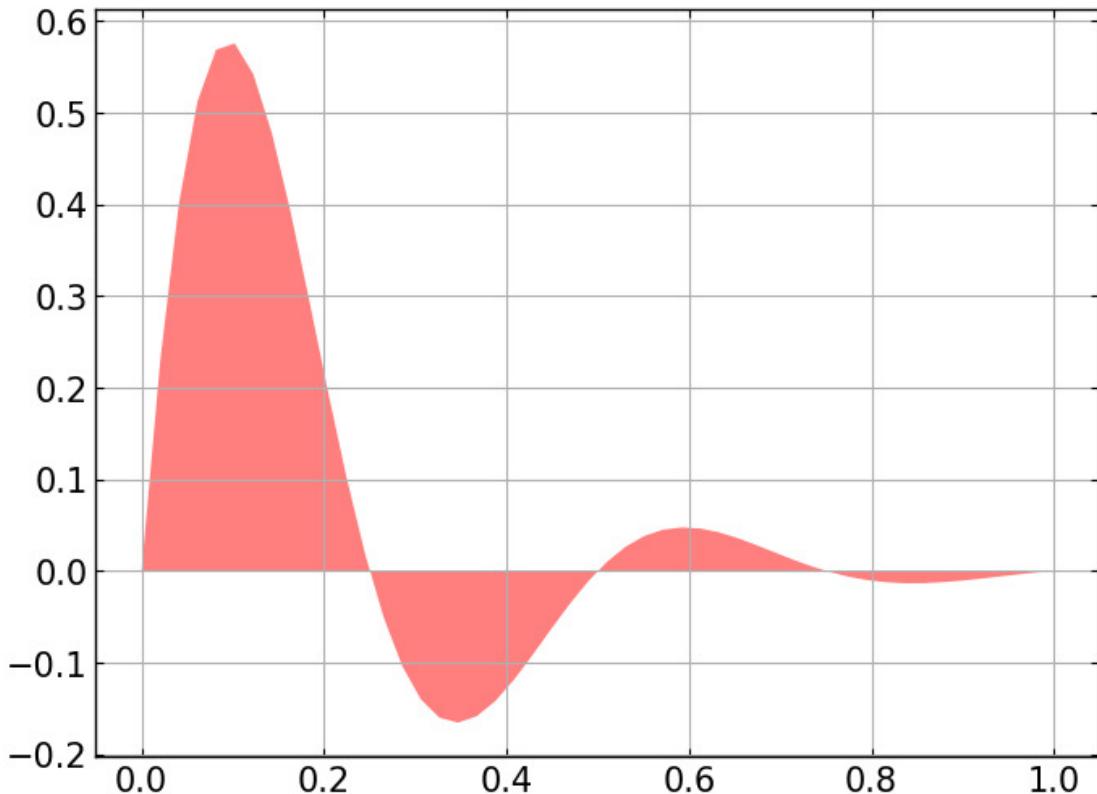
```

"""
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 1)
y = np.sin(4 * np.pi * x) * np.exp(-5 * x)

plt.fill(x, y, 'r', alpha=0.5)
plt.grid(True)
plt.show()

```



In [40]: """

Demo of the fill function with a few features.

In addition to the basic fill plot, this demo shows a few optional features:

- * Multiple curves with a single command.
- * Setting the fill color.
- * Setting the opacity (alpha value).

"""

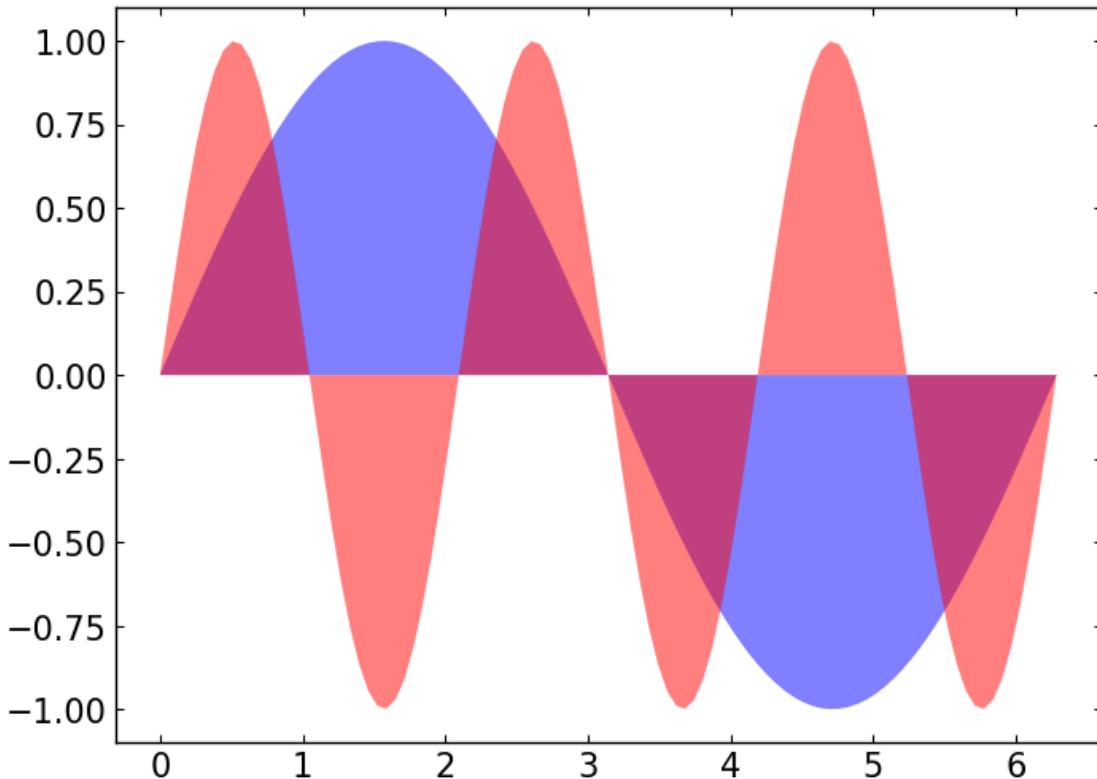
```

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.sin(3.*x)

plt.fill(x,y1,'b', x,y2,'r', alpha=0.5)
plt.show()

```



In [41]:

```

import matplotlib.pyplot as plt
import numpy as np

Nsteps, Nwalkers = 100, 250
t = np.arange(Nsteps)

# an (Nsteps x Nwalkers) array of random walk steps
S1 = 0.002 + 0.01*np.random.randn(Nsteps, Nwalkers)
S2 = 0.004 + 0.02*np.random.randn(Nsteps, Nwalkers)

# an (Nsteps x Nwalkers) array of random walker positions
X1 = S1.cumsum(axis=0)

```

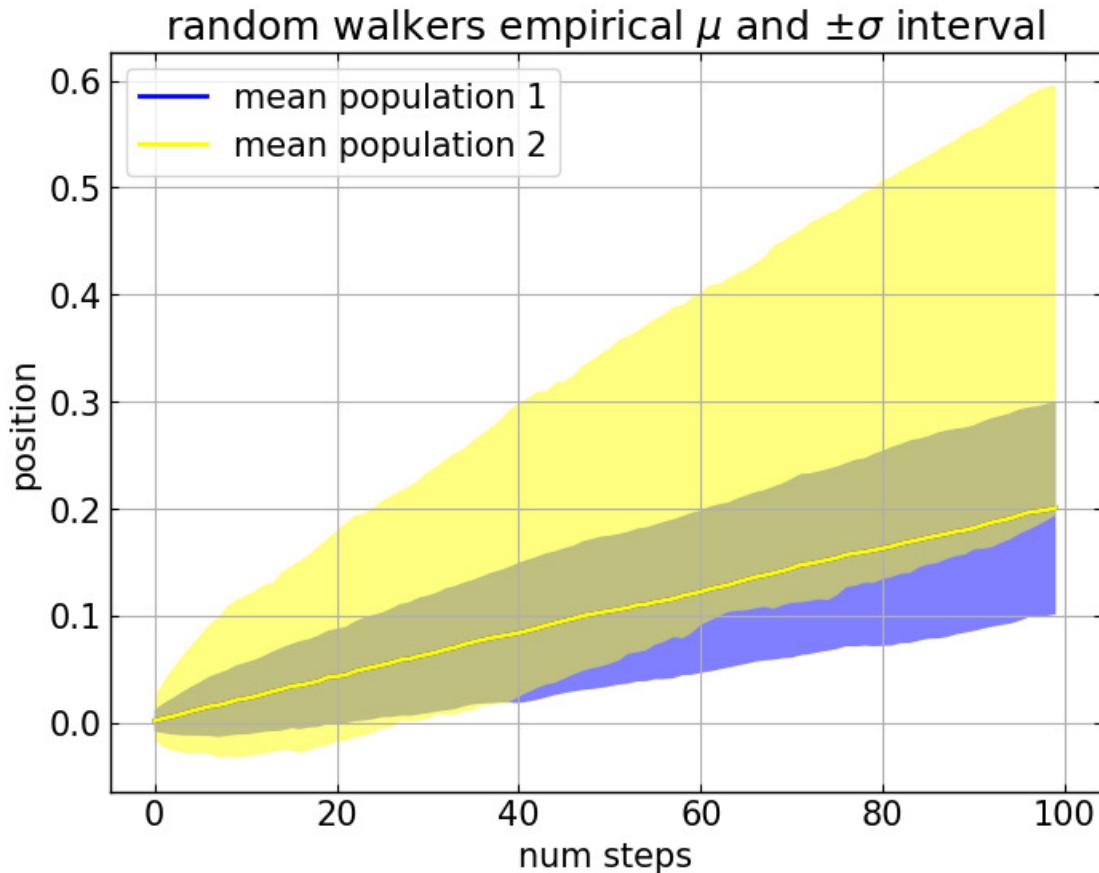
```

X2 = S2.cumsum(axis=0)

# Nsteps length arrays empirical means and standard deviations of both
# populations over time
mu1 = X1.mean(axis=1)
sigma1 = X1.std(axis=1)
mu2 = X2.mean(axis=1)
sigma2 = X2.std(axis=1)

# plot it!
fig, ax = plt.subplots(1)
ax.plot(t, mu1, lw=2, label='mean population 1', color='blue')
ax.plot(t, mu2, lw=2, label='mean population 2', color='yellow')
ax.fill_between(t, mu1+sigma1, mu1-sigma1, facecolor='blue', alpha=0.5)
ax.fill_between(t, mu2+sigma2, mu2-sigma2, facecolor='yellow', alpha=0.5)
ax.set_title('random walkers empirical  $\mu$  and  $\pm\sigma$  interval')
ax.legend(loc='upper left')
ax.set_xlabel('num steps')
ax.set_ylabel('position')
ax.grid()

```



```
In [42]: import matplotlib.pyplot as plt
        import numpy as np

        np.random.seed(1234)

        Nsteps = 500
        t = np.arange(Nsteps)

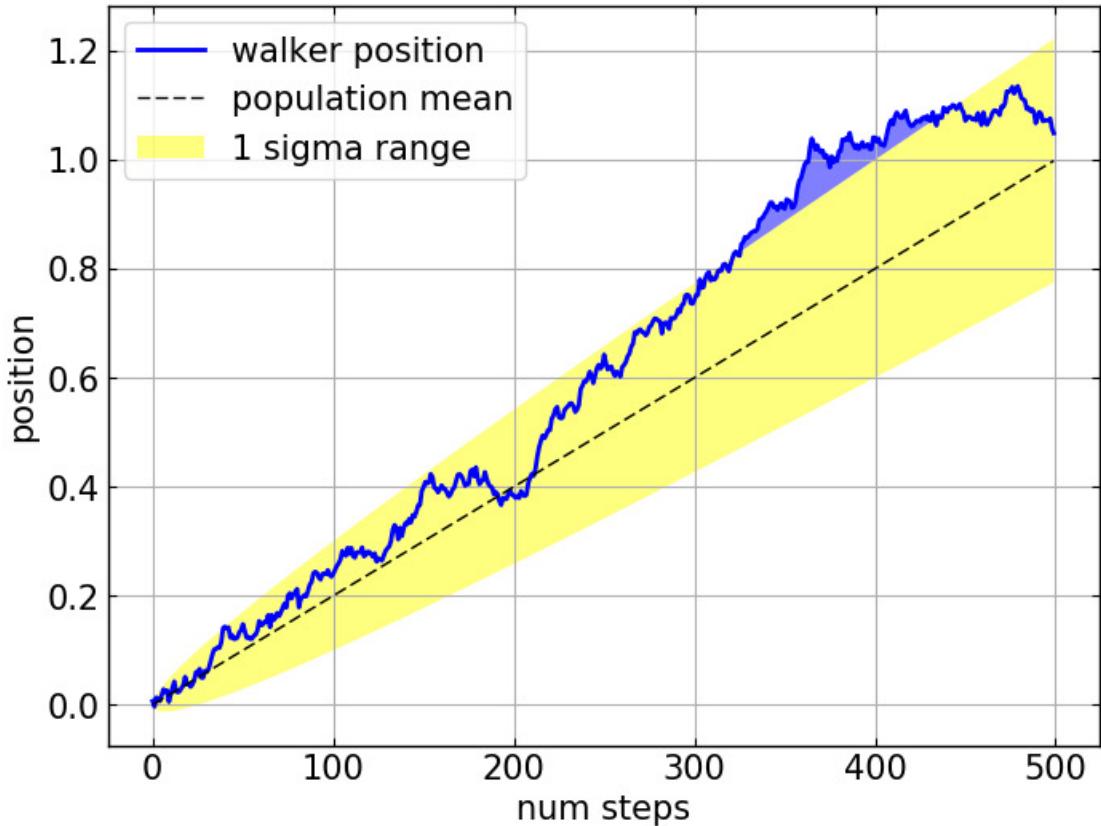
        mu = 0.002
        sigma = 0.01

        # the steps and position
        S = mu + sigma*np.random.randn(Nsteps)
        X = S.cumsum()

        # the 1 sigma upper and lower analytic population bounds
        lower_bound = mu*t - sigma*np.sqrt(t)
        upper_bound = mu*t + sigma*np.sqrt(t)

        fig, ax = plt.subplots(1)
        ax.plot(t, X, lw=2, label='walker position', color='blue')
        ax.plot(t, mu*t, lw=1, label='population mean', color='black', ls='--')
        ax.fill_between(t, lower_bound, upper_bound, facecolor='yellow', alpha=0.5,
                        label='1 sigma range')
        ax.legend(loc='upper left', fontsize='medium')

        # here we use the where argument to only fill the region where the
        # walker is above the population 1 sigma boundary
        ax.fill_between(t, upper_bound, X, where=X>upper_bound, facecolor='blue', alpha=0.5)
        ax.set_xlabel('num steps')
        ax.set_ylabel('position')
        ax.grid()
```



1.12 Patches

```
In [43]: import matplotlib.pyplot as plt
import numpy.random as rnd
from matplotlib.patches import Ellipse

NUM = 250

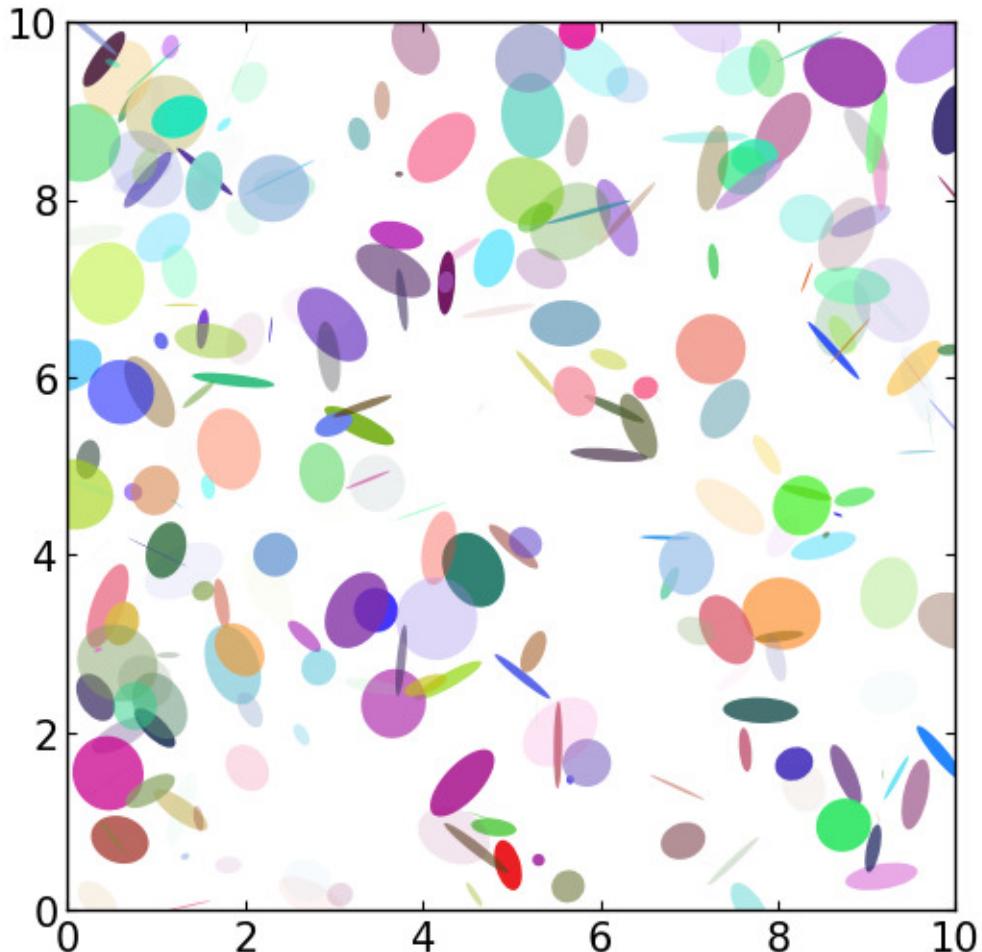
ells = [Ellipse(xy=rnd.rand(2)*10, width=rnd.rand(), height=rnd.rand(), angle=rnd.rand()
               for i in range(NUM))

fig = plt.figure(0)
ax = fig.add_subplot(111, aspect='equal')
for e in ells:
    ax.add_artist(e)
    e.set_clip_box(ax.bbox)
    e.set_alpha(rnd.rand())
    e.set_facecolor(rnd.rand(3))

ax.set_xlim(0, 10)
```

```
ax.set_ylim(0, 10)
```

```
plt.show()
```



```
In [44]: from matplotlib.path import Path
from matplotlib.patches import BoxStyle
import matplotlib.pyplot as plt
```

```
# we may derive from matplotlib.patches.BoxStyle._Base class.
# You need to override transmute method in this case.
```

```
class MyStyle(BoxStyle._Base):
```

```
    """
```

```
    A simple box.
```

```
    """
```

```
def __init__(self, pad=0.3):
```

```

"""
The arguments need to be floating numbers and need to have
default values.

*pad*
    amount of padding
"""

self.pad = pad
super(MyStyle, self).__init__()

def transmute(self, x0, y0, width, height, mutation_size):
    """
    Given the location and size of the box, return the path of
    the box around it.

    - *x0*, *y0*, *width*, *height* : location and size of the box
    - *mutation_size* : a reference scale for the mutation.

    Often, the *mutation_size* is the font size of the text.
    You don't need to worry about the rotation as it is
    automatically taken care of.
    """

# padding
pad = mutation_size * self.pad

# width and height with padding added.
width, height = width + 2.*pad, \
                height + 2.*pad,

# boundary of the padded box
x0, y0 = x0-pad, y0-pad,
x1, y1 = x0+width, y0 + height

cp = [(x0, y0),
       (x1, y0), (x1, y1), (x0, y1),
       (x0-pad, (y0+y1)/2.), (x0, y0),
       (x0, y0)]

com = [Path.MOVETO,
       Path.LINETO, Path.LINETO, Path.LINETO,
       Path.LINETO, Path.LINETO,
       Path.CLOSEPOLY]

path = Path(cp, com)

return path

```

```

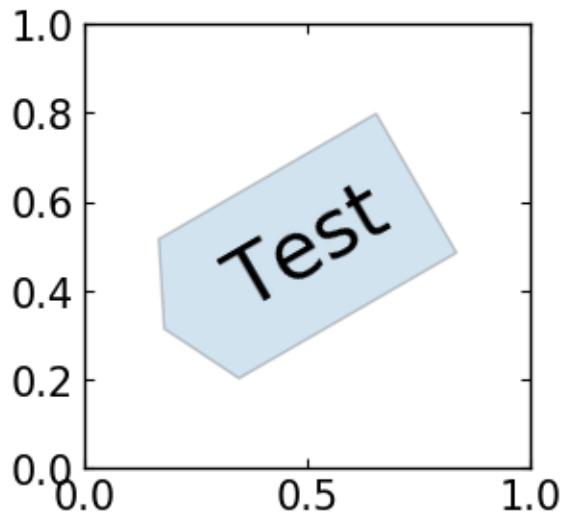
# register the custom style
BoxStyle._style_list["angled"] = MyStyle

plt.figure(1, figsize=(3,3))
ax = plt.subplot(111)
ax.text(0.5, 0.5, "Test", size=30, va="center", ha="center", rotation=30,
        bbox=dict(boxstyle="angled", pad=0.5, alpha=0.2))

del BoxStyle._style_list["angled"]

plt.show()

```



```

In [45]: import numpy.random
         import matplotlib.pyplot as plt

fig = plt.figure(1, figsize=(5,5))
fig.clf()

ax = fig.add_subplot(111)
ax.set_aspect(1)

x1 = -1 + numpy.random.randn(100)
y1 = -1 + numpy.random.randn(100)
x2 = 1. + numpy.random.randn(100)
y2 = 1. + numpy.random.randn(100)

ax.scatter(x1, y1, color="r")

```

```

ax.scatter(x2, y2, color="g")

bbox_props = dict(boxstyle="round", fc="w", ec="0.5", alpha=0.9)
ax.text(-2, -2, "Sample A", ha="center", va="center", size=20,
        bbox=bbox_props)
ax.text(2, 2, "Sample B", ha="center", va="center", size=20,
        bbox=bbox_props)

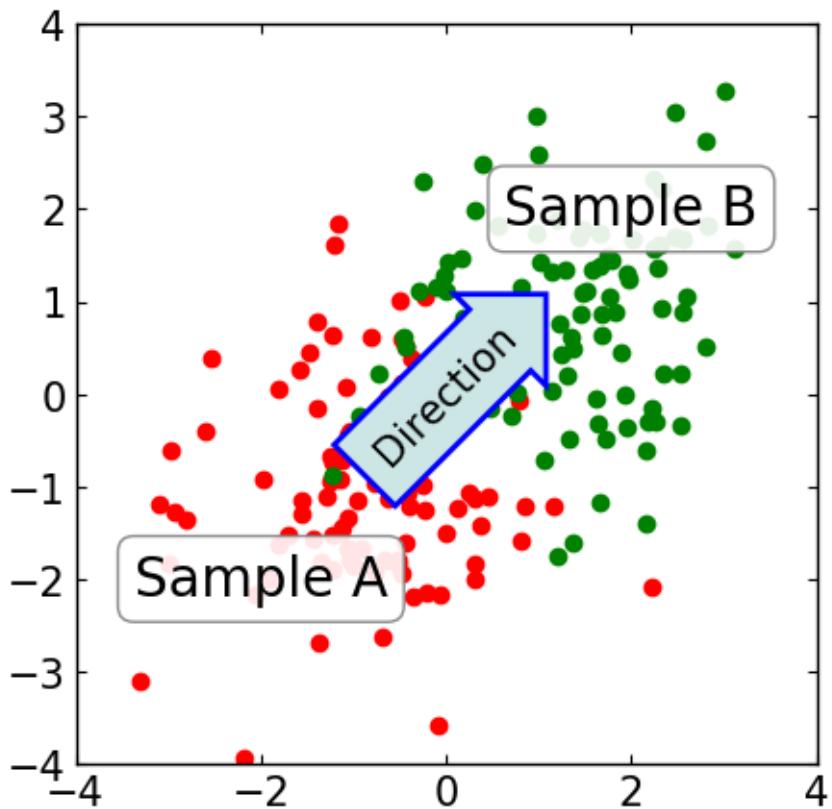
bbox_props = dict(boxstyle="rarrow", fc=(0.8,0.9,0.9), ec="b", lw=2)
t = ax.text(0, 0, "Direction", ha="center", va="center", rotation=45,
            size=15,
            bbox=bbox_props)

bb = t.get_bbox_patch()
bb.set_boxstyle("rarrow", pad=0.6)

ax.set_xlim(-4, 4)
ax.set_ylim(-4, 4)

plt.draw()
plt.show()

```



```
In [46]: import matplotlib.pyplot as plt
        import matplotlib.patches as mpatches

x1, y1 = 0.3, 0.3
x2, y2 = 0.7, 0.7

fig = plt.figure(1, figsize=(8,3))
fig.clf()
from mpl_toolkits.axes_grid.axes_grid import AxesGrid
from mpl_toolkits.axes_grid.anchored_artists import AnchoredText

#from matplotlib.font_manager import FontProperties

def add_at(ax, t, loc=2):
    fp = dict(size=10)
    _at = AnchoredText(t, loc=loc, prop=fp)
    ax.add_artist(_at)
    return _at


grid = AxesGrid(fig, 111, (1, 4), label_mode="1", share_all=True)

grid[0].set_autoscale_on(False)

ax = grid[0]
ax.plot([x1, x2], [y1, y2], ".")
el = mpatches.Ellipse((x1, y1), 0.3, 0.4, angle=30, alpha=0.2)
ax.add_artist(el)
ax.annotate("", xy=(x1, y1), xycoords='data',
            xytext=(x2, y2), textcoords='data',
            arrowprops=dict(arrowstyle="-", #linestyle="dashed",
                           color="0.5",
                           patchB=None,
                           shrinkB=0,
                           connectionstyle="arc3,rad=0.3",
                           ),
            )
add_at(ax, "connect", loc=2)

ax = grid[1]
ax.plot([x1, x2], [y1, y2], ".")
el = mpatches.Ellipse((x1, y1), 0.3, 0.4, angle=30, alpha=0.2)
ax.add_artist(el)
ax.annotate("",
```

```

        xy=(x1, y1), xycoords='data',
        xytext=(x2, y2), textcoords='data',
        arrowprops=dict(arrowstyle="-", #linestyle="dashed",
                       color="0.5",
                       patchB=el,
                       shrinkB=0,
                       connectionstyle="arc3,rad=0.3",
                       ),
    )
)

add_at(ax, "clip", loc=2)

ax = grid[2]
ax.plot([x1, x2], [y1, y2], ".")
el = mpatches.Ellipse((x1, y1), 0.3, 0.4, angle=30, alpha=0.2)
ax.add_artist(el)
ax.annotate("",

        xy=(x1, y1), xycoords='data',
        xytext=(x2, y2), textcoords='data',
        arrowprops=dict(arrowstyle="-", #linestyle="dashed",
                       color="0.5",
                       patchB=el,
                       shrinkB=5,
                       connectionstyle="arc3,rad=0.3",
                       ),
    )
)

add_at(ax, "shrink", loc=2)

ax = grid[3]
ax.plot([x1, x2], [y1, y2], ".")
el = mpatches.Ellipse((x1, y1), 0.3, 0.4, angle=30, alpha=0.2)
ax.add_artist(el)
ax.annotate("",

        xy=(x1, y1), xycoords='data',
        xytext=(x2, y2), textcoords='data',
        arrowprops=dict(arrowstyle="fancy", #linestyle="dashed",
                       color="0.5",
                       patchB=el,
                       shrinkB=5,
                       connectionstyle="arc3,rad=0.3",
                       ),
    )
)

add_at(ax, "mutate", loc=2)

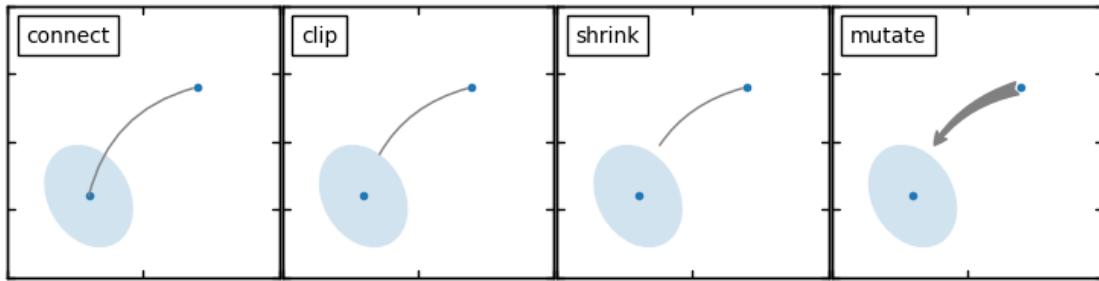
```

```

grid[0].set_xlim(0, 1)
grid[0].set_ylim(0, 1)
grid[0].axis["bottom"].toggle(ticklabels=False)
grid[0].axis["left"].toggle(ticklabels=False)
fig.subplots_adjust(left=0.05, right=0.95, bottom=0.05, top=0.95)

plt.draw()
plt.show()

```



```

In [47]: from matplotlib.patches import Circle
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid.anchored_artists import AnchoredDrawingArea

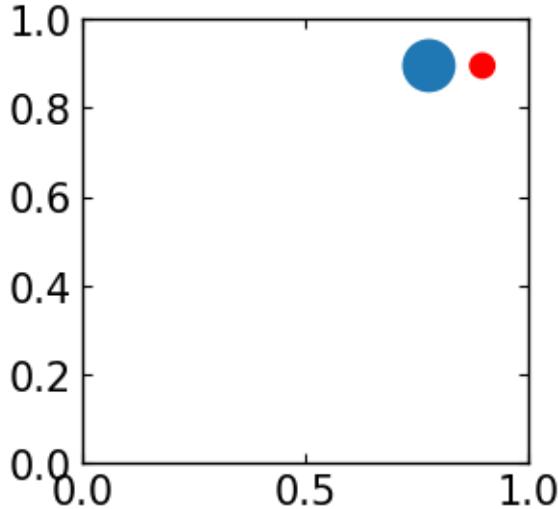
fig=plt.figure(1, figsize=(3,3))
ax = plt.subplot(111)

ada = AnchoredDrawingArea(40, 20, 0, 0,
                         loc=1, pad=0., frameon=False)
p1 = Circle((10, 10), 10)
ada.drawing_area.add_artist(p1)
p2 = Circle((30, 10), 5, fc="r")
ada.drawing_area.add_artist(p2)

ax.add_artist(ada)

plt.show()

```



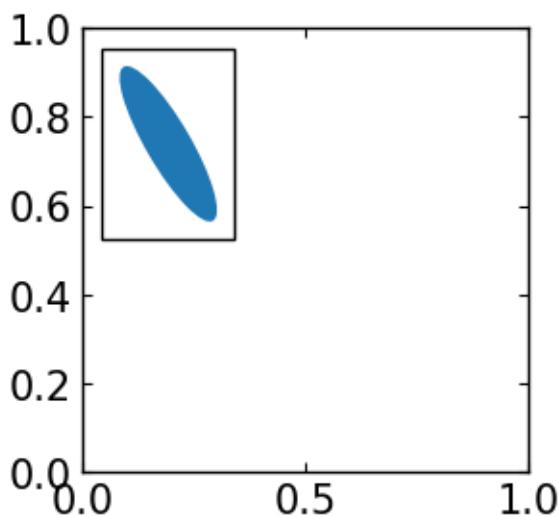
```
In [48]: from matplotlib.patches import Ellipse
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid.anchored_artists import AnchoredAuxTransformBox

fig=plt.figure(1, figsize=(3,3))
ax = plt.subplot(111)

box = AnchoredAuxTransformBox(ax.transData, loc=2)
el = Ellipse((0,0), width=0.1, height=0.4, angle=30) # in data coordinates!
box.drawing_area.add_artist(el)

ax.add_artist(box)

plt.show()
```



1.13 Path

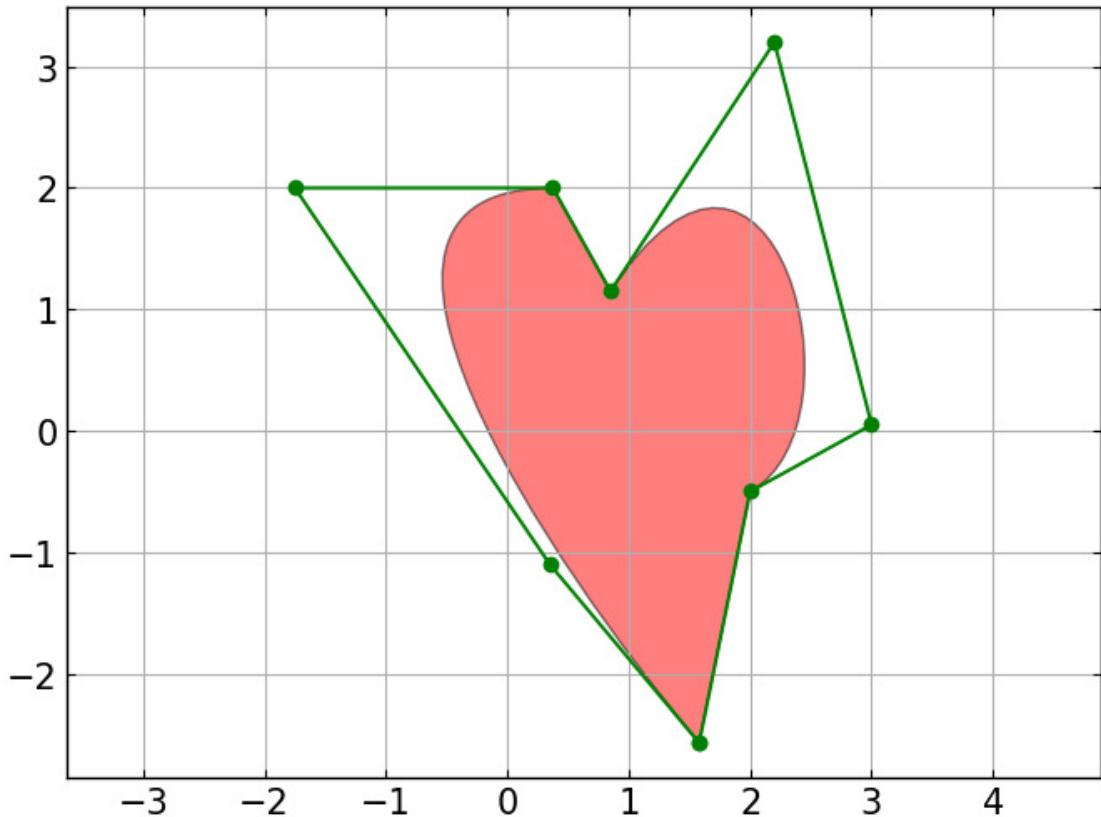
```
In [49]: import matplotlib.path as mpath
         import matplotlib.patches as mpatches
         import matplotlib.pyplot as plt

fig, ax = plt.subplots()

Path = mpath.Path
path_data = [
    (Path.MOVETO, (1.58, -2.57)),
    (Path.CURVE4, (0.35, -1.1)),
    (Path.CURVE4, (-1.75, 2.0)),
    (Path.CURVE4, (0.375, 2.0)),
    (Path.LINETO, (0.85, 1.15)),
    (Path.CURVE4, (2.2, 3.2)),
    (Path.CURVE4, (3, 0.05)),
    (Path.CURVE4, (2.0, -0.5)),
    (Path.CLOSEPOLY, (1.58, -2.57)),
]
codes, verts = zip(*path_data)
path = mpath.Path(verts, codes)
patch = mpatches.PathPatch(path, facecolor='r', alpha=0.5)
ax.add_patch(patch)

# plot control points and connecting lines
x, y = zip(*path.vertices)
line, = ax.plot(x, y, 'go-')

ax.grid()
ax.axis('equal')
plt.show()
```



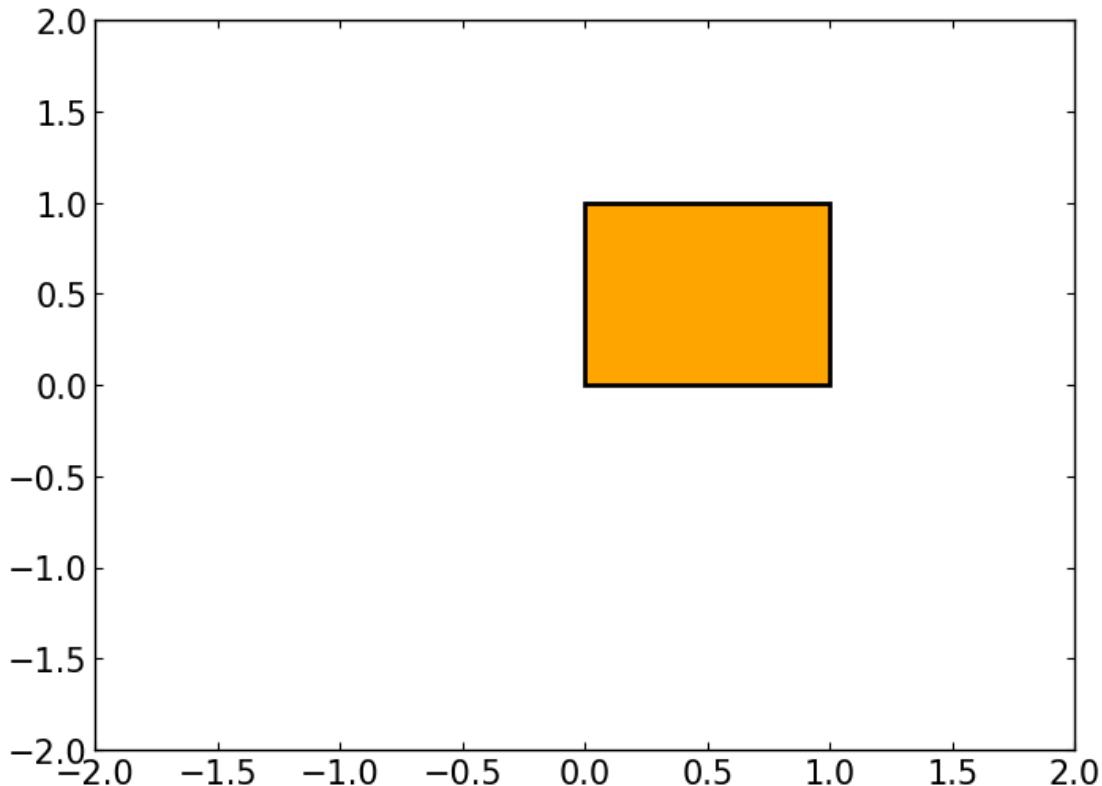
```
In [50]: import matplotlib.pyplot as plt
        from matplotlib.path import Path
        import matplotlib.patches as patches

verts = [
    (0., 0.), # left, bottom
    (0., 1.), # left, top
    (1., 1.), # right, top
    (1., 0.), # right, bottom
    (0., 0.), # ignored
]

codes = [Path.MOVETO,
         Path.LINETO,
         Path.LINETO,
         Path.LINETO,
         Path.CLOSEPOLY,
         ]

path = Path(verts, codes)
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
patch = patches.PathPatch(path, facecolor='orange', lw=2)
ax.add_patch(patch)
ax.set_xlim(-2,2)
ax.set_ylim(-2,2)
plt.show()
```



```
In [51]: import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches

verts = [
    (0., 0.), # P0
    (0.2, 1.), # P1
    (1., 0.8), # P2
    (0.8, 0.), # P3
]

codes = [Path.MOVETO,
         Path.CURVE4,
         Path.CURVE4,
```

```

    Path.CURVE4,
]

path = Path(verts, codes)

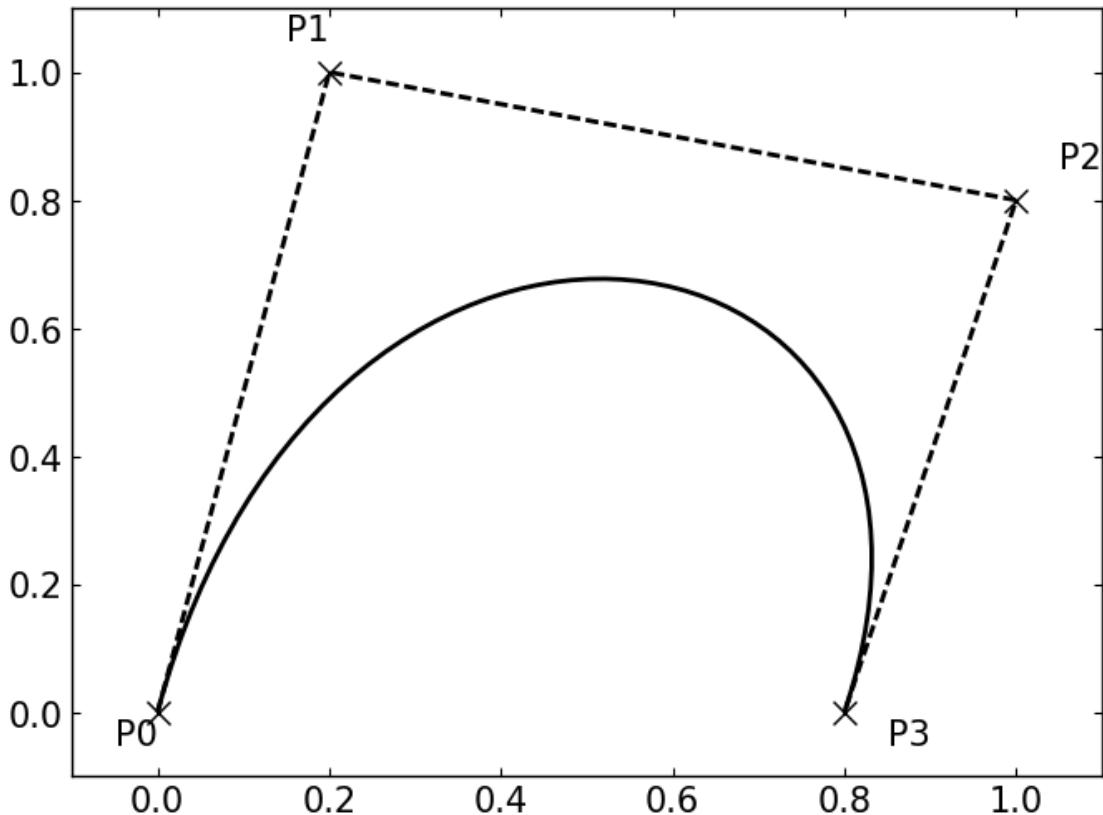
fig = plt.figure()
ax = fig.add_subplot(111)
patch = patches.PathPatch(path, facecolor='none', lw=2)
ax.add_patch(patch)

xs, ys = zip(*verts)
ax.plot(xs, ys, 'x--', lw=2, color='black', ms=10)

ax.text(-0.05, -0.05, 'P0')
ax.text(0.15, 1.05, 'P1')
ax.text(1.05, 0.85, 'P2')
ax.text(0.85, -0.05, 'P3')

ax.set_xlim(-0.1, 1.1)
ax.set_ylim(-0.1, 1.1)
plt.show()

```

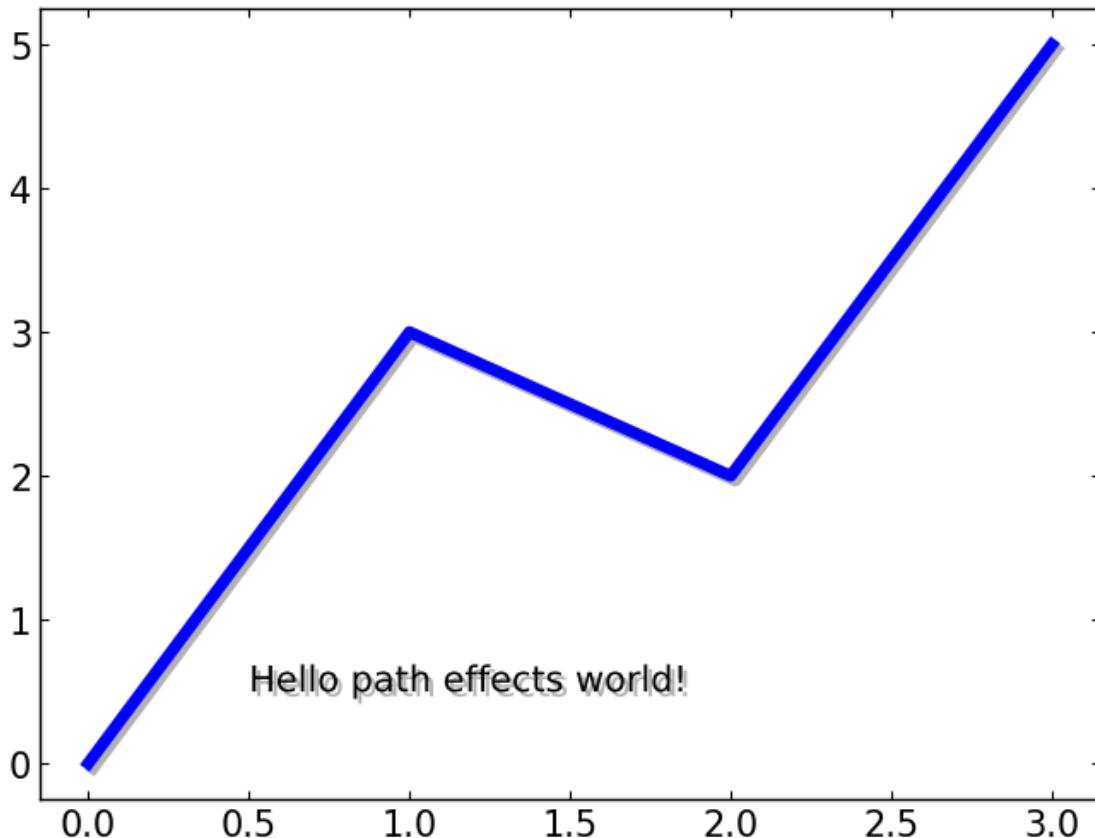


1.13.1 Adding a shadow

```
In [52]: import matplotlib.pyplot as plt
        import matplotlib.path_effects as path_effects

        text = plt.text(0.5, 0.5, 'Hello path effects world!',
                        path_effects=[path_effects.withSimplePatchShadow()])

        plt.plot([0, 3, 2, 5], linewidth=5, color='blue',
                 path_effects=[path_effects.SimpleLineShadow(),
                               path_effects.Normal()])
        plt.show()
```



1.14 Plot3D

```
In [53]: from mpl_toolkits.mplot3d import Axes3D
        from matplotlib import cm
        from matplotlib.ticker import LinearLocator, FormatStrFormatter
        import matplotlib.pyplot as plt
        import numpy as np
```

```

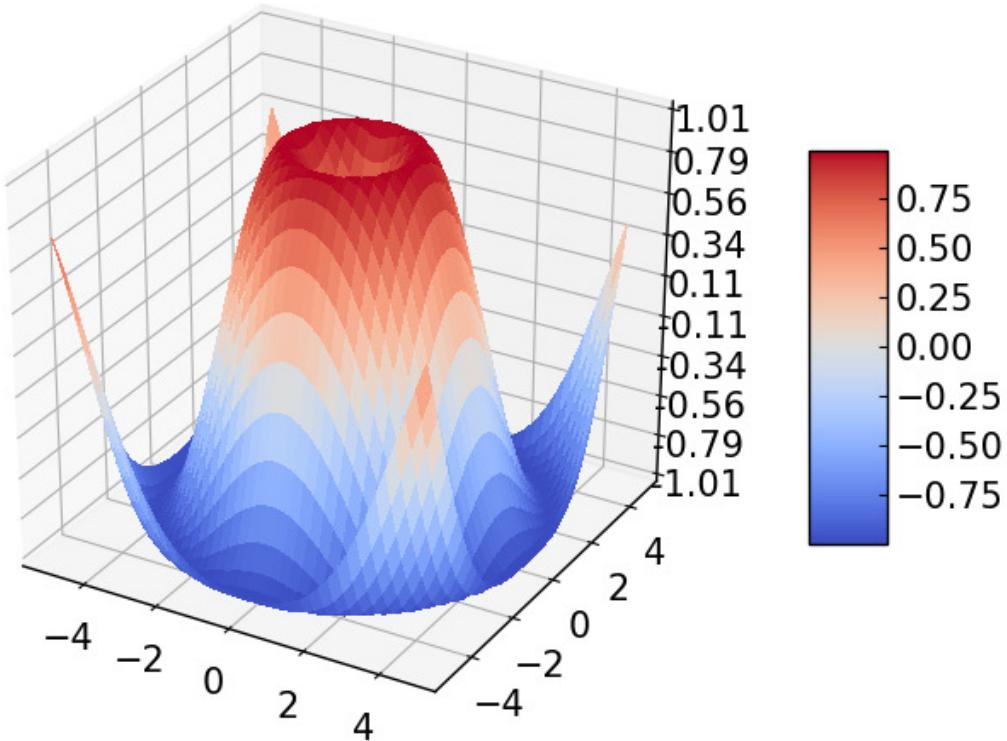
fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.set_zlim(-1.01, 1.01)

ax.xaxis.set_major_locator(LinearLocator(10))
ax.xaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()

```



1.15 Stream line

In [54]: *....*

Demo of the `streamplot` function.

A streamplot, or streamline plot, is used to display 2D vector fields. This example shows a few features of the stream plot function:

```
* Varying the color along a streamline.
* Varying the density of streamlines.
* Varying the line width along a stream line.

"""
import numpy as np
import matplotlib.pyplot as plt

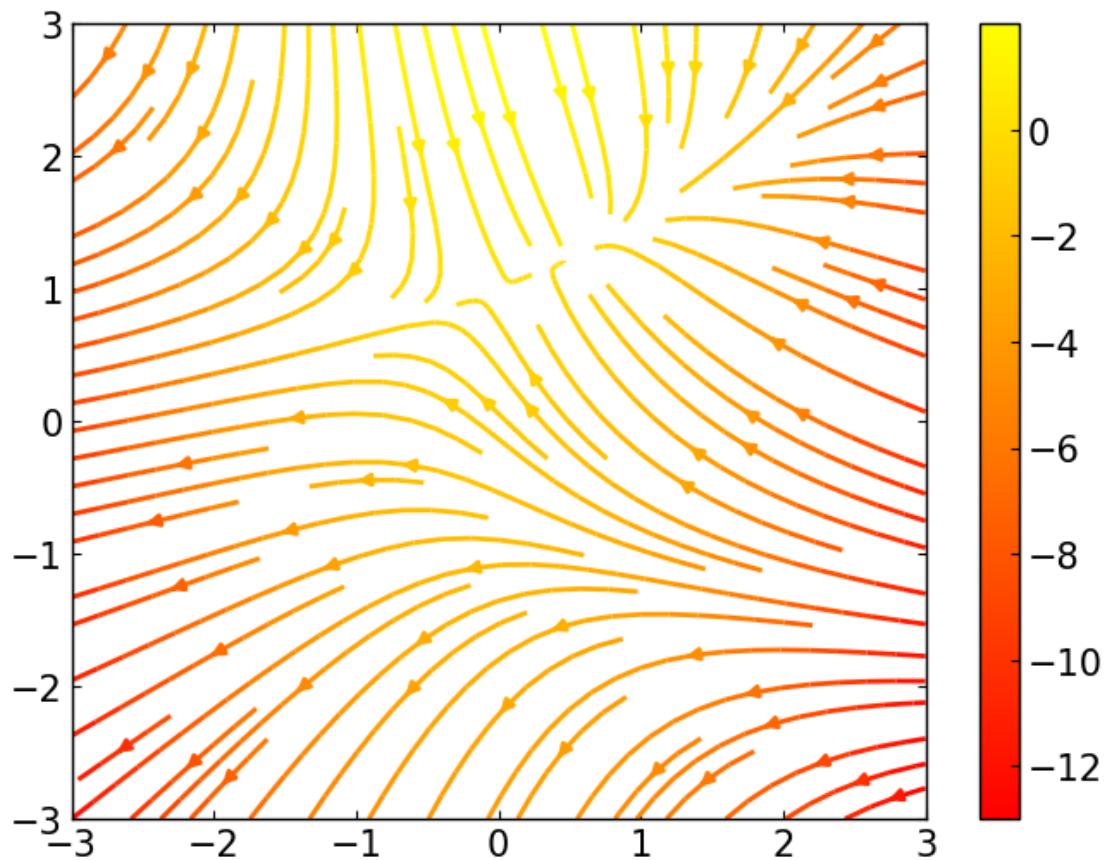
Y, X = np.mgrid[-3:3:100j, -3:3:100j]
U = -1 - X**2 + Y
V = 1 + X - Y**2
speed = np.sqrt(U*U + V*V)

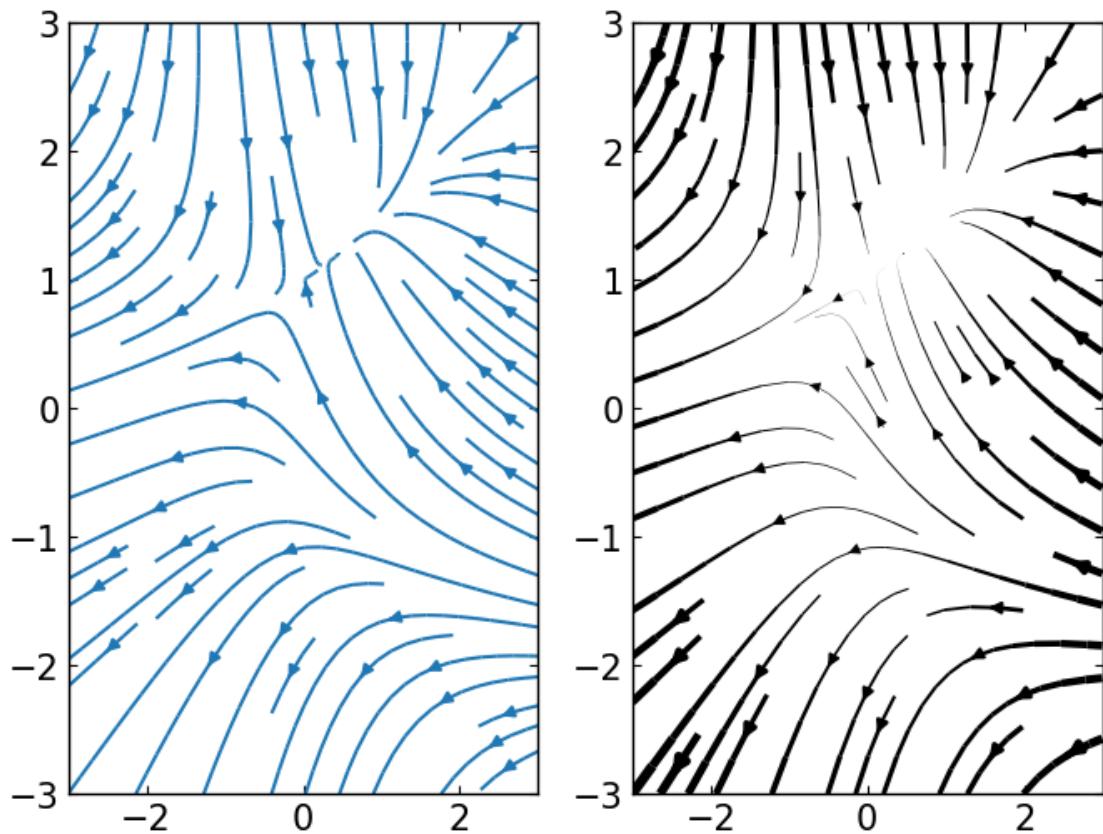
fig0, ax0 = plt.subplots()
strm = ax0.streamplot(X, Y, U, V, color=U, linewidth=2, cmap=plt.cm.autumn)
fig0.colorbar(strm.lines)

fig1, (ax1, ax2) = plt.subplots(ncols=2)
ax1.streamplot(X, Y, U, V, density=[0.5, 1])

lw = 5*speed / speed.max()
ax2.streamplot(X, Y, U, V, density=0.6, color='k', linewidth=lw)

plt.show()
```





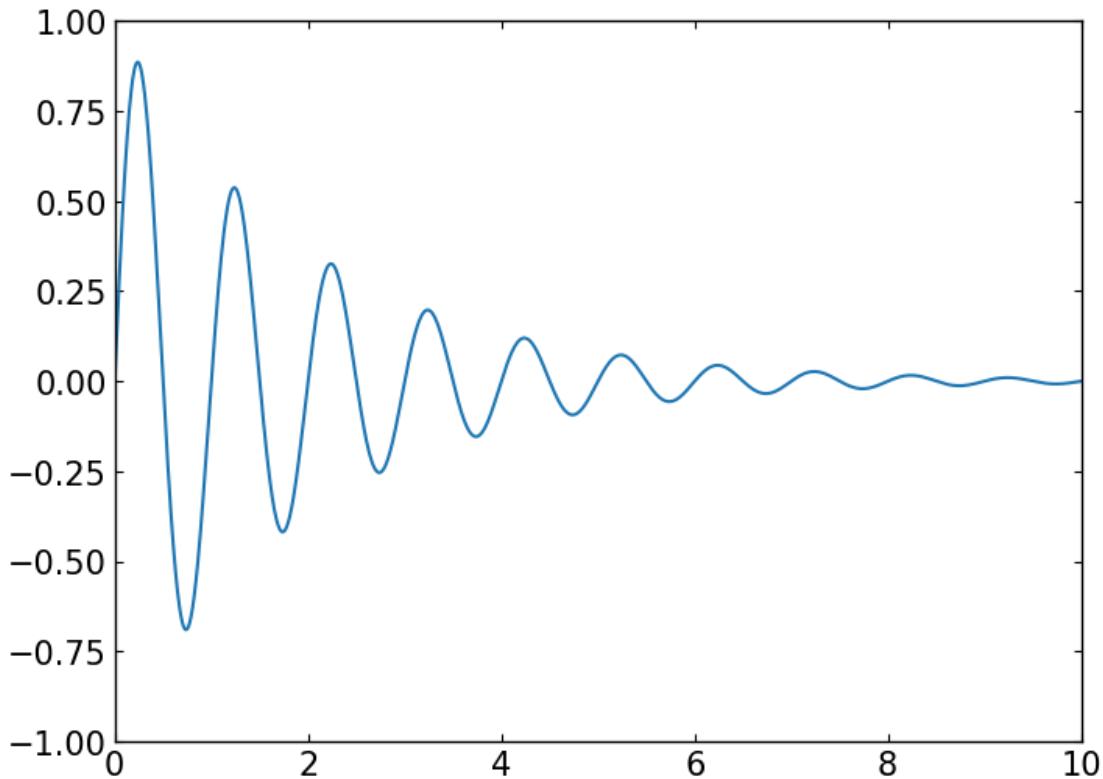
1.16 Transformations

```
In [55]: import numpy as np
        import matplotlib.pyplot as plt

x = np.arange(0, 10, 0.005)
y = np.exp(-x/2.) * np.sin(2*np.pi*x)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x, y)
ax.set_xlim(0, 10)
ax.set_ylim(-1, 1)

plt.show()
```



```
In [56]: type(ax.transData)
```

```
Out[56]: matplotlib.transforms.CompositeGenericTransform
```

```
In [57]: # transform from the data coordinates to display coordinates  
ax.transData.transform((5, 0))
```

```
Out[57]: array([ 410.,  297.])
```

```
In [58]: import numpy as np  
import matplotlib.pyplot as plt  
  
x = np.arange(0, 10, 0.005)  
y = np.exp(-x/2.) * np.sin(2*np.pi*x)  
  
fig = plt.figure()  
ax = fig.add_subplot(111)  
ax.plot(x, y)  
ax.set_xlim(0, 10)  
ax.set_ylim(-1, 1)  
  
xdata, ydata = 5, 0  
xdisplay, ydisplay = ax.transData.transform_point((xdata, ydata))
```

```

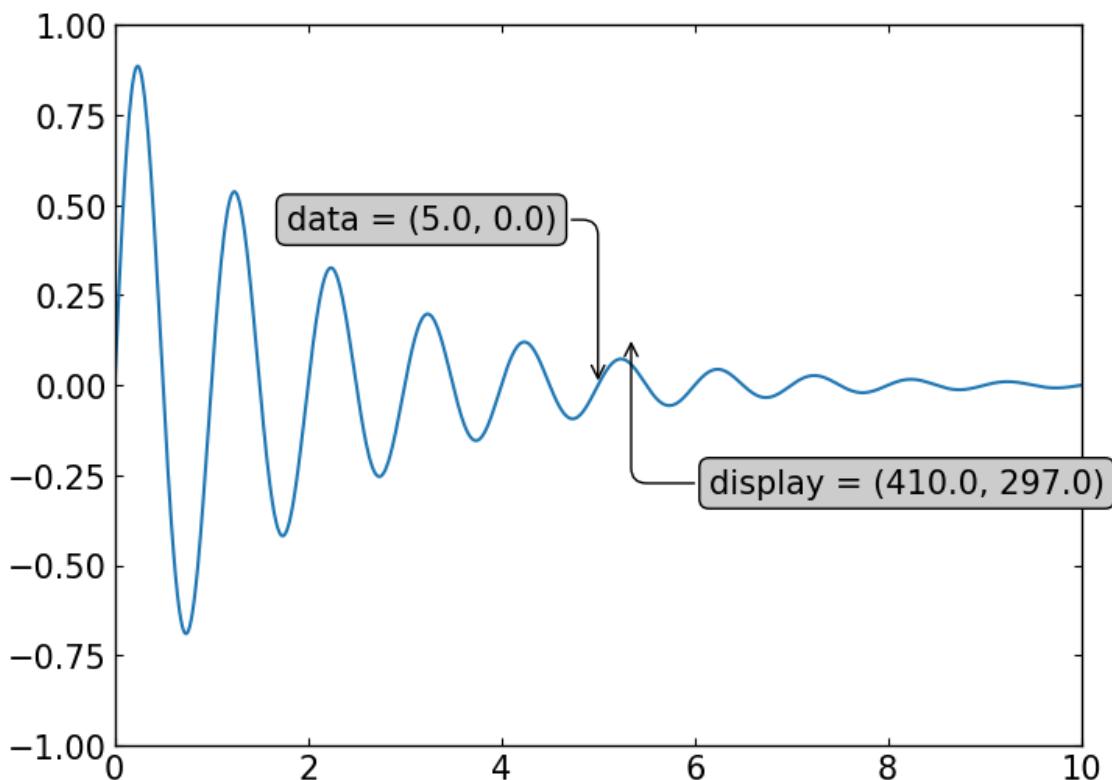
bbox = dict(boxstyle="round", fc="0.8")
arrowprops = dict(
    arrowstyle = "->",
    connectionstyle = "angle,angleA=0,angleB=90,rad=10")

offset = 72
ax.annotate('data = (%.1f, %.1f)'%(xdata, ydata),
            (xdata, ydata), xytext=(-2*offset, offset), textcoords='offset points',
            bbox=bbox, arrowprops=arrowprops)

disp = ax.annotate('display = (%.1f, %.1f)'%(xdisplay, ydisplay),
                    (xdisplay, ydisplay), xytext=(0.5*offset, -offset),
                    xycoords='figure pixels',
                    textcoords='offset points',
                    bbox=bbox, arrowprops=arrowprops)

plt.show()

```



In [59]: *"""*
Axes coordinates

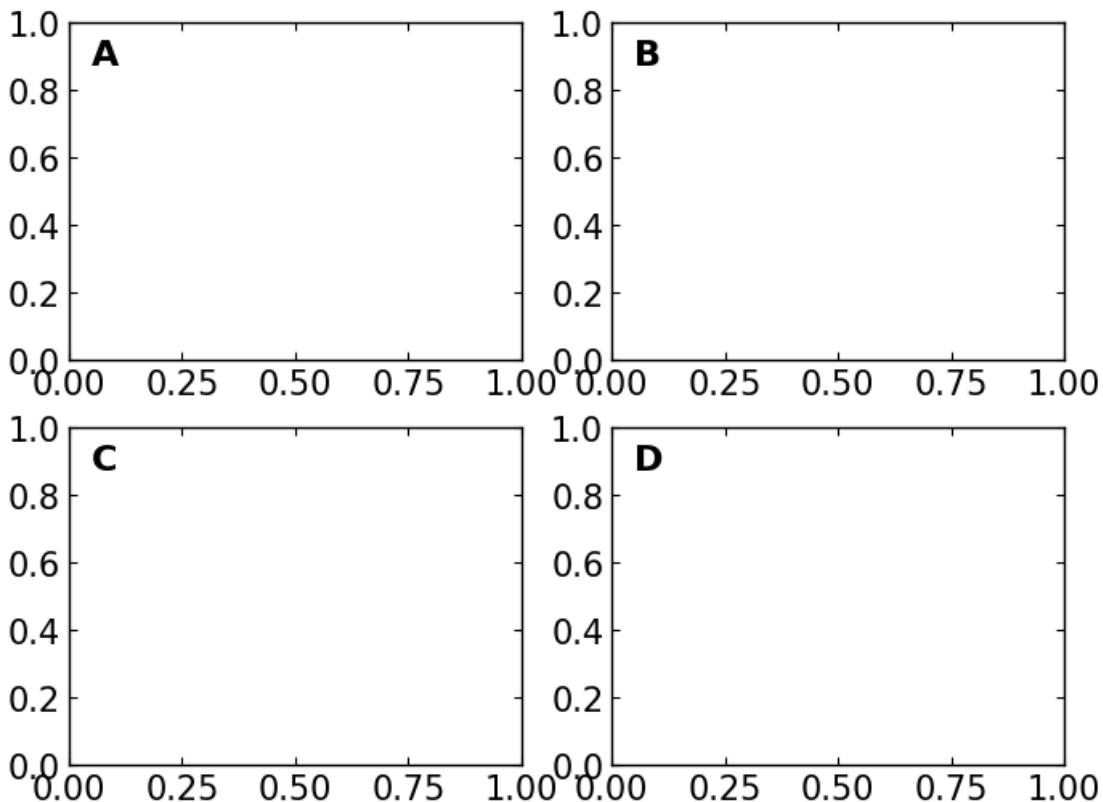
```

"""
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
for i, label in enumerate(('A', 'B', 'C', 'D')):
    ax = fig.add_subplot(2,2,i+1)
    ax.text(0.05, 0.95, label, transform=ax.transAxes,
            fontsize=16, fontweight='bold', va='top')

plt.show()

```



```

In [60]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.patches as patches
        fig = plt.figure()
        ax = fig.add_subplot(111)
        x, y = 10*np.random.rand(2, 1000)
        ax.plot(x, y, 'go')  # plot some data in data coordinates

        circ = patches.Circle((0.5, 0.5), 0.25, transform=ax.transAxes,

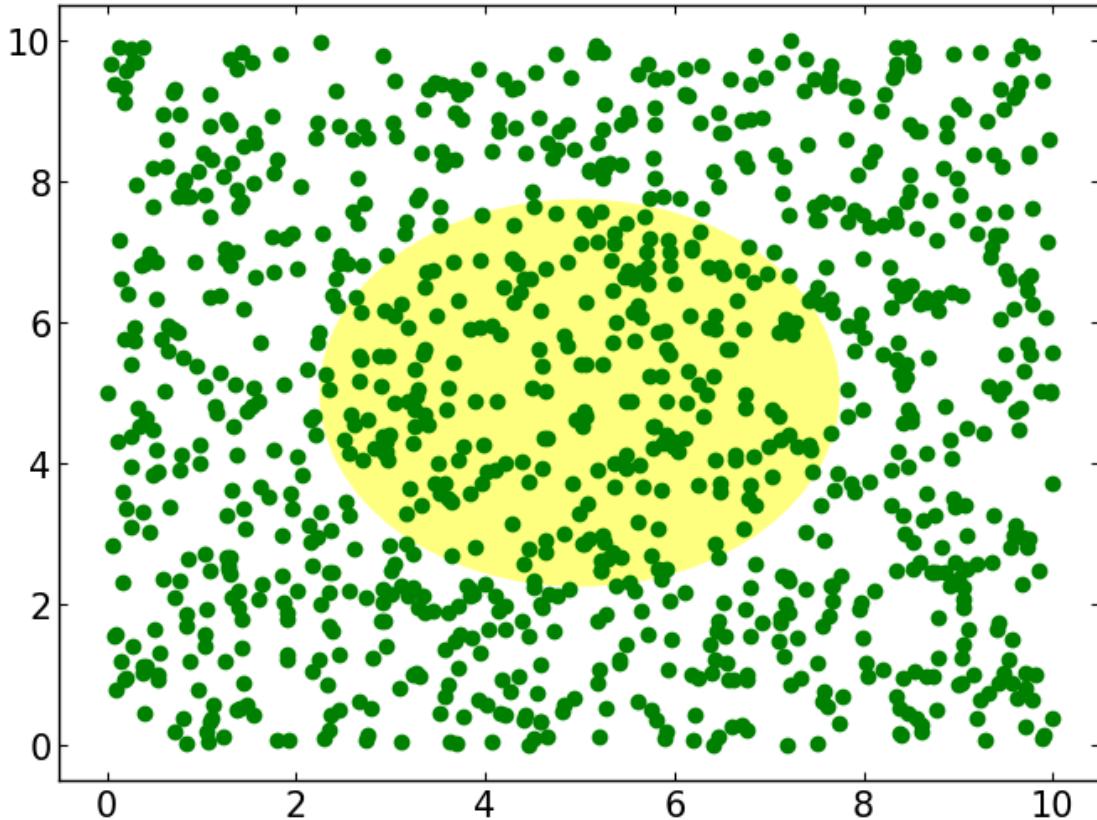
```

```

        facecolor='yellow', alpha=0.5)
ax.add_patch(circ)

plt.show()

```



```

In [61]: """
Blended transformations
"""

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.transforms as transforms

fig = plt.figure()
ax = fig.add_subplot(111)

x = np.random.randn(1000)

ax.hist(x, 30)
ax.set_title(r'$\sigma=1 \wedge \dots \wedge \sigma=2$', fontsize=16)

```

```

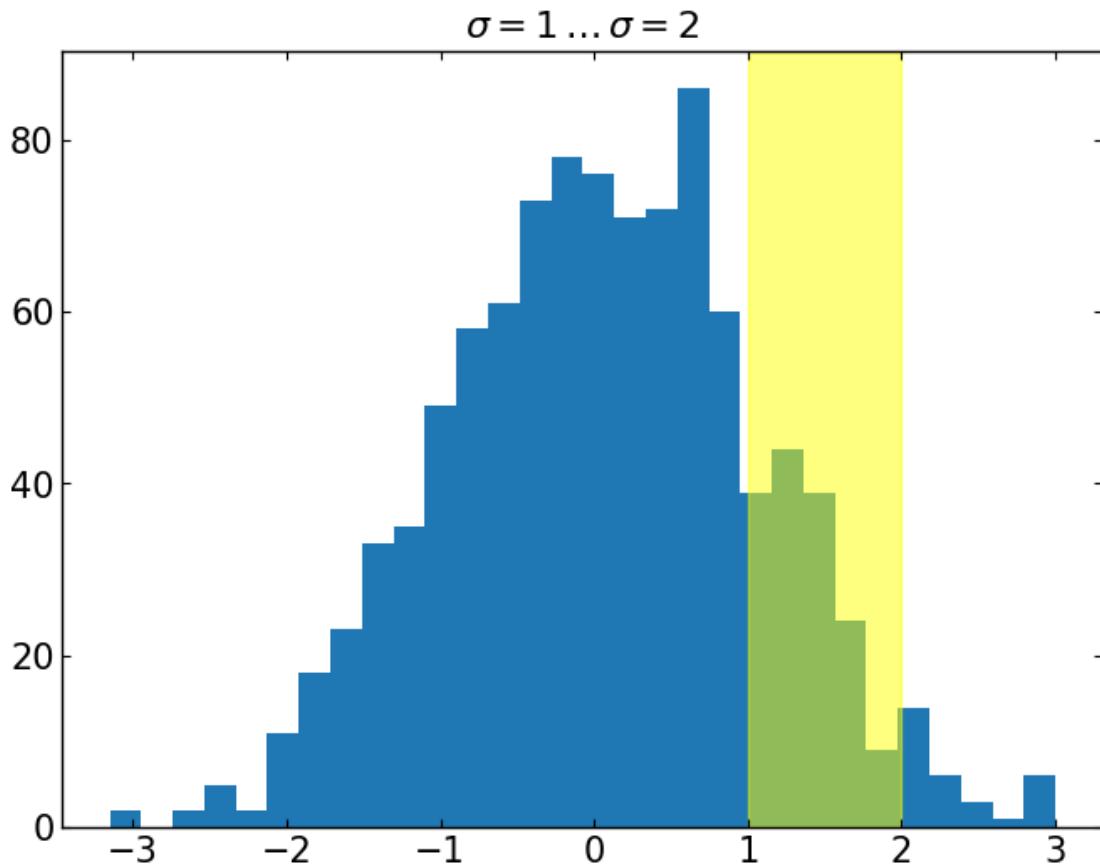
# the x coords of this transformation are data, and the
# y coord are axes
trans = transforms.blended_transform_factory(
    ax.transData, ax.transAxes)

# highlight the 1..2 stddev region with a span.
# We want x to be in data coordinates and y to
# span from 0..1 in axes coords
rect = patches.Rectangle((1,0), width=1, height=1,
                        transform=trans, color='yellow',
                        alpha=0.5)

ax.add_patch(rect)

plt.show()

```



```

In [62]: """
offset transform to create a shadow effect
"""

import numpy as np

```

```

import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.transforms as transforms

fig = plt.figure()
ax = fig.add_subplot(111)

# make a simple sine wave
x = np.arange(0., 2., 0.01)
y = np.sin(2*np.pi*x)
line, = ax.plot(x, y, lw=3, color='blue')

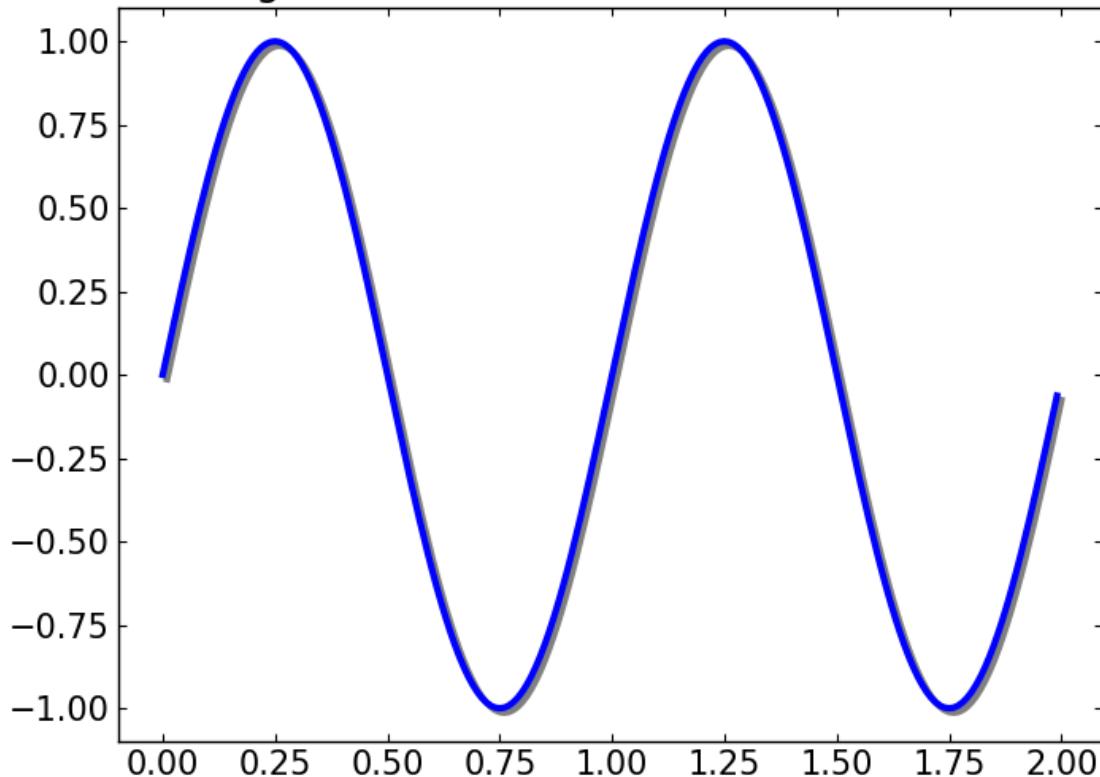
# shift the object over 2 points, and down 2 points
dx, dy = 2/72., -2/72.
offset = transforms.ScaledTranslation(dx, dy,
    fig.dpi_scale_trans)
shadow_transform = ax.transData + offset

# now plot the same data with our offset transform;
# use the zorder to make sure we are below the line
ax.plot(x, y, lw=3, color='gray',
    transform=shadow_transform,
    zorder=0.5*line.get_zorder())

ax.set_title('creating a shadow effect with an offset transform')
plt.show()

```

creating a shadow effect with an offset transform



matplotlib2

January 30, 2019

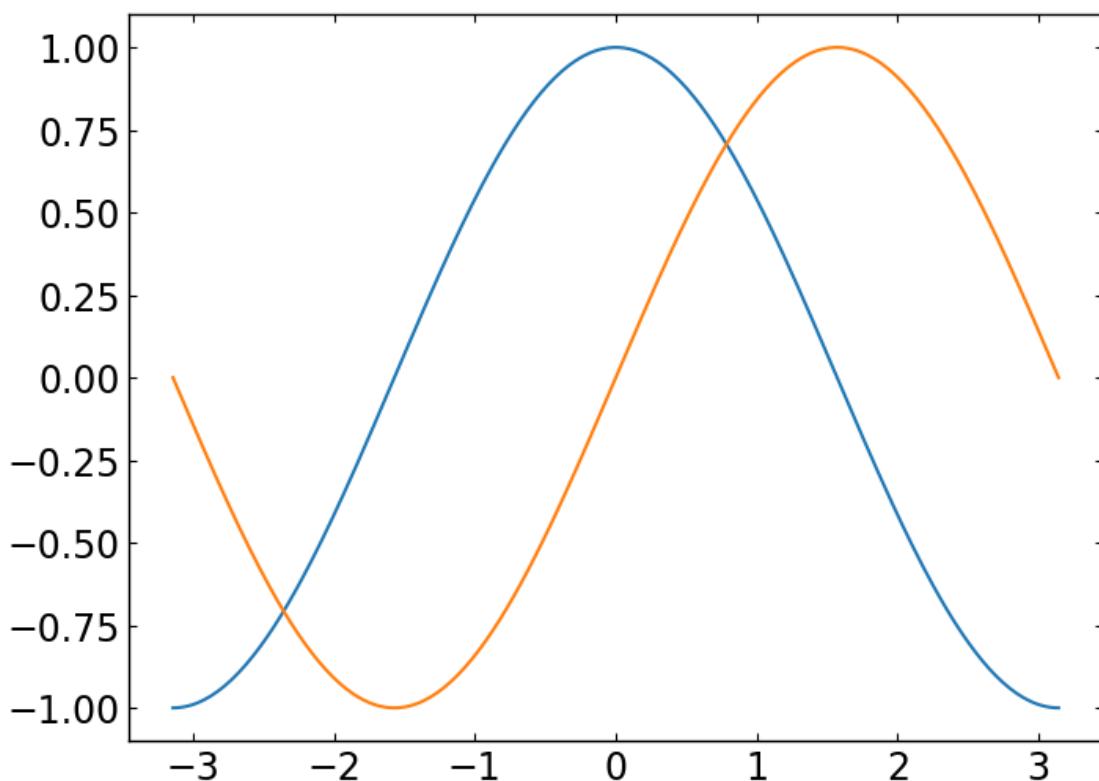
```
In [1]: %matplotlib inline
```

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

```
x = np.linspace(-np.pi, np.pi, 256, endpoint=True)
c, s = np.cos(x), np.sin(x)

plt.plot(x,c)
plt.plot(x,s)

plt.show()
```



```
In [3]: import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(8,6), dpi=80)
plt.subplot(1,1,1)

x = np.linspace(-np.pi, np.pi, 256, endpoint=True)
c, s = np.cos(x), np.sin(x)

plt.plot(x,c, color="blue", linewidth=1., linestyle="--", label="cos")
plt.plot(x,s, color="red", linewidth=1., linestyle="--", label="sin")

#plt.xlim(-4.,4.)
plt.xlim(x.min()*1.5, x.max()*1.5)
#plt.xticks(np.linspace(-4,4,9, endpoint=True))
plt.xticks([-np.pi, -np.pi/2, 0., np.pi/2., np.pi],
           [r'$-\pi$', r'$-\frac{\pi}{2}$', r'$0$', r'$\frac{\pi}{2}$', r'$+\pi$'])
#plt.ylim(-1.,1.)
plt.ylim(c.min()*1.5, c.max()*1.5)
plt.yticks(np.linspace(-1,1,5, endpoint=True))

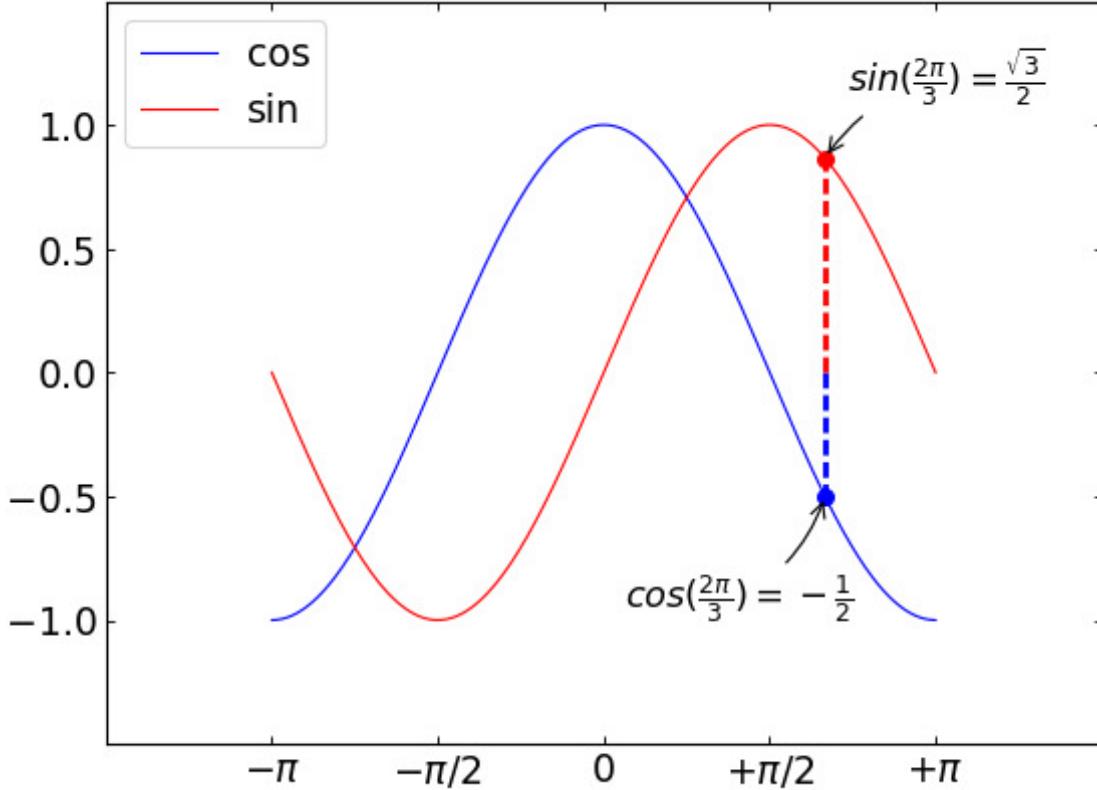
plt.legend(loc='upper left')

t = 2*np.pi/3.
plt.plot([t,t],[0,np.cos(t)], color='blue', linewidth=2.5, linestyle="--")
plt.scatter([t,], [np.cos(t),], 50, color='blue')

plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
            xy=(t, np.sin(t)), xycoords='data',
            xytext=(+10, +30), textcoords='offset points', fontsize=16,
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t,t],[0,np.sin(t)], color='red', linewidth=2.5, linestyle="--")
plt.scatter([t,], [np.sin(t),], 50, color='red')
plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
            xy=(t, np.cos(t)), xycoords='data',
            xytext=(-90, -50), textcoords='offset points', fontsize=16,
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.show()
```



1 matplotlibrc file (initial configuration)

- From `matplotlib.matplotlib_fname()`, we can check the location of the initial configuration (see below)
- If you want to manage the initial setting locally, you can copy the file "matplotlibrc" to `~/.config/matplotlib/`
- The local folder should be in priority to read.
- change the setting in the configuration

Note that "%pylab inline" uses its own configuration for some parts (not all), you may need to force it down if you want to use the configuration regardless of "inline option"

- execute "ipython profile create"
- then "ipython_config.py" & "ipython_kernel_config.py" will be generated in "`$HOME/.ipython/profile_default/`" folder
- In the "ipython_kernel_config.py", add the line "c.InlineBackend.rc = {}"
- To remove the grey background color in the figure, you may need to set `figure.facecolor` to "white" in matplotlibrc configuration file.

```
In [4]: import matplotlib as mpl
```

```
print mpl.matplotlib_fname()
```

```
/home/astrodoo/anaconda2/lib/python2.7/site-packages/matplotlib/mpl-data/matplotlibrc
```

```
In [5]: !cat /home/astrodoo/.config/matplotlib/matplotlibrc
```

```
### MATPLOTLIBRC FORMAT
```

```
# This is a sample matplotlib configuration file - you can find a copy  
# of it on your system in  
# site-packages/matplotlib/mpl-data/matplotlibrc. If you edit it  
# there, please note that it will be overwritten in your next install.  
# If you want to keep a permanent local copy that will not be  
# overwritten, place it in the following location:  
  
# unix/linux:  
  
#     $HOME/.config/matplotlib/matplotlibrc or  
#     $XDG_CONFIG_HOME/matplotlib/matplotlibrc (if $XDG_CONFIG_HOME is set)  
  
# other platforms:  
  
#     $HOME/.matplotlib/matplotlibrc  
  
#  
  
# See http://matplotlib.org/users/customizing.html#the-matplotlibrc-file for  
# more details on the paths which are checked for the configuration file.  
  
#  
  
# This file is best viewed in a editor which supports python mode  
# syntax highlighting. Blank lines, or lines starting with a comment  
# symbol, are ignored, as are trailing comments. Other lines must  
# have the format  
  
#     key : val # optional comment
```

```

#
# Colors: for the color values below, you can either use - a
# matplotlib color string, such as r, k, or b - an rgb tuple, such as
# (1.0, 0.5, 0.0) - a hex string, such as ff00ff - a scalar
# grayscale intensity such as 0.75 - a legal html color name, e.g., red,
# blue, darkslategray

#####
# CONFIGURATION BEGINS HERE

# The default backend; one of GTK GTKAgg GTKCairo GTK3Agg GTK3Cairo
# MacOSX Qt4Agg Qt5Agg TkAgg WX WXAagg Agg Cairo GDK PS PDF SVG
# Template.

# You can also deploy your own backend outside of matplotlib by
# referring to the module name (which must be in the PYTHONPATH) as
# 'module://my_backend'.

backend      : Qt5Agg

# If you are using the Qt4Agg backend, you can choose here
# to use the PyQt4 bindings or the newer PySide bindings to
# the underlying Qt4 toolkit.

#backend.qt4 : PyQt4          # PyQt4 | PySide

# Note that this can be overridden by the environment variable
# QT_API used by Enthought Tool Suite (ETS); valid values are

```

```
# "pyqt" and "pyside". The "pyqt" setting has the side effect of
# forcing the use of Version 2 API for QString and QVariant.

# The port to use for the web server in the WebAgg backend.

# webagg.port : 8888

# If webagg.port is unavailable, a number of other random ports will
# be tried until one that is available is found.

# webagg.port_retries : 50

# When True, open the webbrowser to the plot that is shown

# webagg.open_in_browser : True

# When True, the figures rendered in the nbagg backend are created with
# a transparent background.

# nbagg.transparent : False

# if you are running pyplot inside a GUI and your backend choice
# conflicts, we will automatically try to find a compatible one for
# you if backend_fallback is True

#backendFallback: True

#interactive : False

#toolbar      : toolbar2 # None | toolbar2 ("classic" is deprecated)
```

```

#timezone      : UTC          # a pytz timezone string, e.g., US/Central or Europe/Paris

# Where your matplotlib data lives if you installed to a non-default
# location. This is where the matplotlib fonts, bitmaps, etc reside
#datapath : /home/jdhunter/mpldata

### LINES

# See http://matplotlib.org/api/artist_api.html#module-matplotlib.lines for more
# information on line properties.

#lines.linewidth    : 1.5      # line width in points

#lines.linestyle    : -        # solid line

#lines.color        : C0        # has no affect on plot(); see axes.prop_cycle

#lines.marker       : None     # the default marker

#lines.markeredgewidth : 1.0    # the line width around the marker symbol

#lines.markersize   : 6         # markersize, in points

#lines.dash_joinstyle : miter      # miter|round|bevel

#lines.dash_capstyle : butt      # butt|round|projecting

#lines.solid_joinstyle : miter      # miter|round|bevel

#lines.solid_capstyle : projecting # butt|round|projecting

#lines.antialiased : True       # render lines in antialiased (no jaggies)

# The three standard dash patterns. These are scaled by the linewidth.

#lines.dashed_pattern : 2.8, 1.2

```

```

#lines.dashdot_pattern : 4.8, 1.2, 0.8, 1.2

#lines.dotted_pattern : 1.1, 1.1

#lines.scale_dashes : True


#markers.fillstyle: full # full|left|right|bottom|top|none


### PATCHES

# Patches are graphical objects that fill 2D space, like polygons or

# circles. See

# http://matplotlib.org/api/artist_api.html#module-matplotlib.patches

# information on patch properties

#patch.linewidth      : 1          # edge width in points.

#patch.facecolor      : C0

#patch.edgecolor       : black    # if forced, or patch is not filled

#patch.force_edgecolor : False    # True to always use edgecolor

#patch.antialiased     : True     # render patches in antialiased (no jaggies)


### HATCHES

#hatch.color      : k

#hatch.linewidth : 1.0


### Boxplot

#boxplot.notch      : False

#boxplot.vertical   : True

```

```
#boxplot.whiskers      : 1.5
#boxplot.bootstrap      : None
#boxplot.patchartist   : False
#boxplot.showmeans      : False
#boxplot.showcaps       : True
#boxplot.showbox        : True
#boxplot.showfliers     : True
#boxplot.meanline       : False

#boxplot.flierprops.color          : 'k'
#boxplot.flierprops.marker        : 'o'
#boxplot.flierprops.markerfacecolor : 'none'
#boxplot.flierprops.markeredgecolor : 'k'
#boxplot.flierprops.markersize    : 6
#boxplot.flierprops.linestyle     : 'none'
#boxplot.flierprops.linewidth     : 1.0

#boxplot.boxprops.color          : 'k'
#boxplot.boxprops.linewidth      : 1.0
#boxplot.boxprops.linestyle      : '-'

#boxplot.whiskerprops.color      : 'k'
#boxplot.whiskerprops.linewidth  : 1.0
#boxplot.whiskerprops.linestyle  : '-'
```

```

#boxplot.capprops.color      : 'k'
#boxplot.capprops.linewidth : 1.0
#boxplot.capprops.linestyle : '-'

#boxplot.medianprops.color    : 'C1'
#boxplot.medianprops.linewidth : 1.0
#boxplot.medianprops.linestyle : '-'

#boxplot.meanprops.color       : 'C2'
#boxplot.meanprops.marker      : '^'
#boxplot.meanprops.markerfacecolor : 'C2'
#boxplot.meanprops.markeredgecolor : 'C2'
#boxplot.meanprops.markersize     : 6
#boxplot.meanprops.linestyle      : 'none'
#boxplot.meanprops.linewidth       : 1.0

### FONT
#
# font properties used by text.Text. See
# http://matplotlib.org/api/font_manager_api.html for more
# information on font properties. The 6 font properties used for font
# matching are given below with their default values.

#

```

```

# The font.family property has five values: 'serif' (e.g., Times),
# 'sans-serif' (e.g., Helvetica), 'cursive' (e.g., Zapf-Chancery),
# 'fantasy' (e.g., Western), and 'monospace' (e.g., Courier). Each of
# these font families has a default list of font names in decreasing
# order of priority associated with them. When text.usetex is False,
# font.family may also be one or more concrete font names.

#
# The font.style property has three values: normal (or roman), italic
# or oblique. The oblique style will be used for italic, if it is not
# present.

#
# The font.variant property has two values: normal or small-caps. For
# TrueType fonts, which are scalable fonts, small-caps is equivalent
# to using a font size of 'smaller', or about 83% of the current font
# size.

#
# The font.weight property has effectively 13 values: normal, bold,
# bolder, lighter, 100, 200, 300, ..., 900. Normal is the same as
# 400, and bold is 700. bolder and lighter are relative values with
# respect to the current weight.

#
# The font.stretch property has 11 values: ultra-condensed,
# extra-condensed, condensed, semi-condensed, normal, semi-expanded,
# expanded, extra-expanded, ultra-expanded, wider, and narrower. This

```

```

# property is not currently implemented.

#
# The font.size property is the default font size for text, given in pts.
# 10 pt is the standard value.

#
#font.family      : sans-serif
#font.style       : normal
#font.variant    : normal
#font.weight     : medium
#font.stretch     : normal

# note that font.size controls default text sizes. To configure
# special text sizes tick labels, axes, labels, title, etc, see the rc
# settings for axes and ticks. Special text sizes can be defined
# relative to font.size, using the following values: xx-small, x-small,
# small, medium, large, x-large, xx-large, larger, or smaller

#font.size        : 10.0
font.size         : 17.0

#font.serif       : DejaVu Serif, Bitstream Vera Serif, New Century Schoolbook, Century Sch
#font.sans-serif   : DejaVu Sans, Bitstream Vera Sans, Lucida Grande, Verdana, Geneva, Lucid
#font.cursive      : Apple Chancery, Textile, Zapf Chancery, Sand, Script MT, Felipa, cursive
#font.fantasy      : Comic Sans MS, Chicago, Charcoal, Impact, Western, Humor Sans, xkcd, fanta
#font.monospace    : DejaVu Sans Mono, Bitstream Vera Sans Mono, Andale Mono, Nimbus Mono L, monospace

### TEXT

```

```

# text properties used by text.Text. See
# http://matplotlib.org/api/artist_api.html#module-matplotlib.text for more
# information on text properties

#text.color          : black

### LaTeX customizations. See http://wiki.scipy.org/Cookbook/Matplotlib/UsingTex

#text.usetex        : False # use latex for all text handling. The following fonts
# are supported through the usual rc parameter settings:
# new century schoolbook, bookman, times, palatino,
# zapf chancery, charter, serif, sans-serif, helvetica,
# avant garde, courier, monospace, computer modern roman,
# computer modern sans serif, computer modern typewriter
# If another font is desired which can loaded using the
# LaTeX \usepackage command, please inquire at the
# matplotlib mailing list

#text.latex.unicode : False # use "ucs" and "inputenc" LaTeX packages for handling
# unicode strings.

#text.latex.preamble : # IMPROPER USE OF THIS FEATURE WILL LEAD TO LATEX FAILURES
# AND IS THEREFORE UNSUPPORTED. PLEASE DO NOT ASK FOR HELP
# IF THIS FEATURE DOES NOT DO WHAT YOU EXPECT IT TO.
# preamble is a comma separated list of LaTeX statements
# that are included in the LaTeX document preamble.
# An example:

```

```

# text.latex.preamble : \usepackage{bm},\usepackage{euler}

# The following packages are always loaded with usetex, so

# beware of package collisions: color, geometry, graphicx,

# type1cm, textcomp. Adobe Postscript (PSSNFS) font packages

# may also be loaded, depending on your font settings

#text.dvipnghack : None      # some versions of dvipng don't handle alpha

# channel properly. Use True to correct

# and flush ~/.matplotlib/tex.cache

# before testing and False to force

# correction off. None will try and

# guess based on your dvipng version

#text.hinting : auto    # May be one of the following:

#   'none': Perform no hinting

#   'auto': Use FreeType's autohinter

#   'native': Use the hinting information in the

#               font file, if available, and if your

#               FreeType library supports it

#   'either': Use the native hinting information,

#               or the autohinter if none is available.

# For backward compatibility, this value may also be

# True === 'auto' or False === 'none'.

#text.hinting_factor : 8 # Specifies the amount of softness for hinting in the

```

```

# horizontal direction. A value of 1 will hint to full
# pixels. A value of 2 will hint to half pixels etc.

#text.antialiased : True # If True (default), the text will be antialiased.

# This only affects the Agg backend.

# The following settings allow you to select the fonts in math mode.

# They map from a TeX font name to a fontconfig font pattern.

# These settings are only used if mathtext.fontset is 'custom'.

# Note that this "custom" mode is unsupported and may go away in the
# future.

#mathtext.cal : cursive

#mathtext.rm  : serif

#mathtext.tt  : monospace

#mathtext.it  : serif:italic

#mathtext.bf  : serif:bold

#mathtext.sf  : sans

#mathtext.fontset : dejavusans # Should be 'dejavusans' (default),
# 'dejavuserif', 'cm' (Computer Modern), 'stix',
# 'stixsans' or 'custom'

#mathtext.fallback_to_cm : True # When True, use symbols from the Computer Modern
# fonts when a symbol can not be found in one of
# the custom math fonts.

```

```

#mathtext.default : it # The default font to use for math.

                    # Can be any of the LaTeX font names, including

                    # the special name "regular" for the same font

                    # used in regular text.

### AXES

# default face and edge color, default tick sizes,
# default fontsizes for ticklabels, and so on. See
# http://matplotlib.org/api/axes_api.html#module-matplotlib.axes

#axes.facecolor      : white    # axes background color
#axes.edgecolor       : black     # axes edge color
#axes.linewidth       : 0.8      # edge linewidth
axes.linewidth       : 1.        # edge linewidth
#axes.grid            : False     # display grid or not
#axes.titlesize       : large     # fontsize of the axes title
#axes.titlepad         : 6.0      # pad between axes and title in points
#axes.labelsize        : medium    # fontsize of the x any y labels
#axes.labelpad         : 4.0      # space between label and axis
#axes.labelweight      : normal    # weight of the x and y labels
#axes.labelcolor        : black
#axes.axisbelow        : 'line'   # draw axis gridlines and ticks below
                                # patches (True); above patches but below
                                # lines ('line'); or above all (False)

```

```

#axes.formatter.limits : -7, 7 # use scientific notation if log10

                                # of the axis range is smaller than the

                                # first or larger than the second

#axes.formatter.use_locale : False # When True, format tick labels

                                # according to the user's locale.

                                # For example, use ',' as a decimal

                                # separator in the fr_FR locale.

#axes.formatter.use_mathtext : False # When True, use mathtext for scientific

                                # notation.

#axes.formatter.useoffset      : True     # If True, the tick label formatter

                                # will default to labeling ticks relative

                                # to an offset when the data range is

                                # small compared to the minimum absolute

                                # value of the data.

#axes.formatter.offset_threshold : 4       # When useoffset is True, the offset

                                # will be used when it can remove

                                # at least this number of significant

                                # digits from tick labels.

# axes.spines.left   : True   # display axis spines

# axes.spines.bottom : True

# axes.spines.top    : True

# axes.spines.right  : True

```

```

#axes.unicode_minus : True      # use unicode for the minus symbol

# rather than hyphen. See
# http://en.wikipedia.org/wiki/Plus_and_minus_signs#Character_code

#axes.prop_cycle : cycler('color',
#
#                           ['1f77b4', 'ff7f0e', '2ca02c', 'd62728',
#
#                            '9467bd', '8c564b', 'e377c2', '7f7f7f',
#
#                            'bcbd22', '17becf']))

# color cycle for plot lines

# as list of string colorspecs:

# single letter, long name, or

# web-style hex

#axes.autolimit_mode : data # How to scale axes limits to the data.

# Use "data" to use data limits, plus some margin

# Use "round_number" move to the nearest "round" number

#axes.xmargin : .05 # x margin. See `axes.Axes.margins`

#axes.ymargin : .05 # y margin See `axes.Axes.margins`


#polaraxes.grid : True      # display grid on polar axes

#axes3d.grid : True      # display grid on 3d axes


### DATES

# These control the default format strings used in AutoDateFormatter.

# Any valid format datetime format string can be used (see the python

```

```

# `datetime` for details). For example using '%x' will use the locale date representation

# '%X' will use the locale time representation and '%c' will use the full locale datetime

# representation.

# These values map to the scales:

# {'year': 365, 'month': 30, 'day': 1, 'hour': 1/24, 'minute': 1 / (24 * 60)}


# date.autoformatter.year      : %Y
# date.autoformatter.month    : %Y-%m
# date.autoformatter.day      : %Y-%m-%d
# date.autoformatter.hour     : %m-%d %H
# date.autoformatter.minute   : %d %H:%M
# date.autoformatter.second   : %H:%M:%S
# date.autoformatter.microsecond : %M:%S.%f

### TICKS

# see http://matplotlib.org/api/axis_api.html#matplotlib.axis.Tick

#xtick.top          : False    # draw ticks on the top side
xtick.top          : True     # draw ticks on the top side
#xtick.bottom        : True     # draw ticks on the bottom side
#xtick.major.size   : 3.5      # major tick size in points
#xtick.minor.size   : 2         # minor tick size in points
#xtick.major.width  : 0.8      # major tick width in points
#xtick.minor.width  : 0.6      # minor tick width in points
#xtick.major.pad    : 3.5      # distance to major tick label in points

```

```

#xtick.minor.pad      : 3.4      # distance to the minor tick label in points

#xtick.color          : k        # color of the tick labels

#xtick.labelsize       : medium # fontsize of the tick labels

#xtick.direction       : out     # direction: in, out, or inout

xtick.direction       : in      # direction: in, out, or inout

#xtick.minor.visible  : False   # visibility of minor ticks on x-axis

#xtick.major.top       : True    # draw x axis top major ticks

#xtick.major.bottom    : True    # draw x axis bottom major ticks

#xtick.minor.top       : True    # draw x axis top minor ticks

#xtick.minor.bottom    : True    # draw x axis bottom minor ticks


#ytick.left            : True    # draw ticks on the left side

#ytick.right           : False   # draw ticks on the right side

ytick.right            : True    # draw ticks on the right side

#ytick.major.size      : 3.5     # major tick size in points

#ytick.minor.size      : 2        # minor tick size in points

#ytick.major.width     : 0.8     # major tick width in points

#ytick.minor.width     : 0.6     # minor tick width in points

#ytick.major.pad       : 3.5     # distance to major tick label in points

#ytick.minor.pad       : 3.4     # distance to the minor tick label in points

#ytick.color           : k        # color of the tick labels

#ytick.labelsize        : medium # fontsize of the tick labels

#ytick.direction        : out     # direction: in, out, or inout

ytick.direction        : in      # direction: in, out, or inout

```

```
#ytick.minor.visible : False # visibility of minor ticks on y-axis

#xtick.major.left : True # draw y axis left major ticks

#xtick.major.right : True # draw y axis right major ticks

#xtick.minor.left : True # draw y axis left minor ticks

#xtick.minor.right : True # draw y axis right minor ticks
```

GRIDS

```
#grid.color : b0b0b0 # grid color

#grid.linestyle : - # solid

#grid.linewidth : 0.8 # in points

#grid.alpha : 1.0 # transparency, between 0.0 and 1.0
```

Legend

```
#legend.loc : best

#legend.frameon : True # if True, draw the legend on a background patch

#legend.framealpha : 0.8 # legend patch transparency

#legend.facecolor : inherit # inherit from axes.facecolor; or color spec

#legend.edgecolor : 0.8 # background patch boundary color

#legend.fancybox : True # if True, use a rounded box for the

# legend background, else a rectangle

#legend.shadow : False # if True, give background a shadow effect

#legend.numpoints : 1 # the number of marker points in the legend line

#legend.scatterpoints : 1 # number of scatter points
```

```

#legend.markerscale : 1.0      # the relative size of legend markers vs. original

#legend.fontsize : medium

# Dimensions as fraction of fontsize:

#legend.borderpad : 0.4      # border whitespace

#legend.labelspacing : 0.5     # the vertical space between the legend entries

#legend.handlelength : 2.0     # the length of the legend lines

#legend.handleheight : 0.7     # the height of the legend handle

#legend.handletextpad : 0.8     # the space between the legend line and legend text

#legend.borderaxespad : 0.5     # the border between the axes and legend edge

#legend.columnspacing : 2.0     # column separation

```

FIGURE

```

# See http://matplotlib.org/api/figure_api.html#matplotlib.figure.Figure

#figure.titlesize : large      # size of the figure title (Figure.suptitle())

#figure.titleweight : normal    # weight of the figure title

#figure.figsize   : 6.4, 4.8    # figure size in inches

figure.figsize   : 8, 6      # figure size in inches

#figure.dpi       : 100        # figure dots per inch

#figure.facecolor : white     # figure facecolor; 0.75 is scalar gray

#figure.edgecolor : white     # figure edgecolor

#figure.autolayout : False    # When True, automatically adjust subplot

                                # parameters to make the plot fit the figure

#figure.max_open_warning : 20   # The maximum number of figures to open through

                                # the pyplot interface before emitting a warning.

```

```

# If less than one this feature is disabled.

# The figure subplot parameters. All dimensions are a fraction of the
# figure width and height.
#figure.subplot.left      : 0.125    # the left side of the subplots of the figure
#figure.subplot.right     : 0.9      # the right side of the subplots of the figure
#figure.subplot.bottom    : 0.11     # the bottom of the subplots of the figure
#figure.subplot.top       : 0.88     # the top of the subplots of the figure
#figure.subplot.wspace   : 0.2      # the amount of width reserved for blank space between subplots
# expressed as a fraction of the average axis width
#figure.subplot.hspace   : 0.2      # the amount of height reserved for white space between subplots
# expressed as a fraction of the average axis height

### IMAGES

#image.aspect : equal           # equal | auto | a number
#image.interpolation : nearest # see help(imshow) for options
#image.cmap   : viridis         # A colormap name, gray etc...
#image.lut    : 256              # the size of the colormap lookup table
#image.origin : upper            # lower | upper
#image.resample : True
#image.composite_image : True   # When True, all the images on a set of axes are
#                               # combined into a single composite image before
#                               # saving a figure as a vector graphics file,
#                               # such as a PDF.

```

```

### CONTOUR PLOTS

#contour.negative_linestyle : dashed # dashed | solid
#contour.corner_mask      : True   # True | False | legacy


### ERRORBAR PLOTS

#errorbar.capsize : 0           # length of end cap on error bars in pixels


### HISTOGRAM PLOTS

#hist.bins : 10                 # The default number of histogram bins.

# If Numpy 1.11 or later is
# installed, may also be `auto`


### SCATTER PLOTS

#scatter.marker : o            # The default marker type for scatter plots.


### Agg rendering

### Warning: experimental, 2008/10/10

#agg.path.chunksize : 0          # 0 to disable; values in the range
# 10000 to 100000 can improve speed slightly
# and prevent an Agg rendering failure
# when plotting very large data sets,
# especially if they are very gappy.

# It may cause minor artifacts, though.

```

```

        # A value of 20000 is probably a good

        # starting point.

### SAVING FIGURES

#path.simplify : True    # When True, simplify paths by removing "invisible"

        # points to reduce file size and increase rendering

        # speed

#path.simplify_threshold : 0.1  # The threshold of similarity below which

        # vertices will be removed in the simplification

        # process

#path.snap : True # When True, rectilinear axis-aligned paths will be snapped to

        # the nearest pixel when certain criteria are met. When False,

        # paths will never be snapped.

#path.sketch : None # May be none, or a 3-tuple of the form (scale, length,

        # randomness).

        # *scale* is the amplitude of the wiggle

        # perpendicular to the line (in pixels). *length*

        # is the length of the wiggle along the line (in

        # pixels). *randomness* is the factor by which

        # the length is randomly scaled.

# the default savefig params can be different from the display params

# e.g., you may want a higher resolution, or to make the figure

# background white

#savefig.dpi      : figure    # figure dots per inch or 'figure'

```

```

#savefig.facecolor    : white      # figure facecolor when saving
#savefig.edgecolor    : white      # figure edgecolor when saving
#savefig.format       : png        # png, ps, pdf, svg
#savefig.bbox          : standard # 'tight' or 'standard'.
                                         # 'tight' is incompatible with pipe-based animation
                                         # backends but will workd with temporary file based ones:
                                         # e.g. setting animation.writer to ffmpeg will not work,
                                         # use ffmpeg_file instead
#savefig.pad_inches   : 0.1        # Padding to be used when bbox is set to 'tight'
#savefig.jpeg_quality: 95         # when a jpeg is saved, the default quality parameter.
#savefig.directory    : ~          # default directory in savefig dialog box,
                                         # leave empty to always use current working directory
#savefig.transparent  : False      # setting that controls whether figures are saved with a
                                         # transparent background by default

# tk backend params
#tk.window_focus     : False      # Maintain shell focus for TkAgg

# ps backend params
#ps.papersize        : letter    # auto, letter, legal, ledger, A0-A10, B0-B10
#ps.useafm           : False      # use of afm fonts, results in small files
#ps.usedistiller     : False      # can be: None, ghostscript or xpdf
                                         # Experimental: may produce smaller files.
                                         # xpdf intended for production of publication quality

```

```

# but requires ghostscript, xpdf and ps2eps

#ps.distiller.res : 6000      # dpi

#ps.fonttype : 3            # Output Type 3 (Type3) or Type 42 (TrueType)

# pdf backend params

#pdf.compression : 6 # integer from 0 to 9

# 0 disables compression (good for debugging)

#pdf.fonttype : 3          # Output Type 3 (Type3) or Type 42 (TrueType)

# svg backend params

#svg.image_inline : True    # write raster image data directly into the svg file

#svg.fonttype : 'path'       # How to handle SVG fonts:

#     'none': Assume fonts are installed on the machine where the SVG will be viewed.

#     'path': Embed characters as paths -- supported by most SVG renderers

#     'svgfont': Embed characters as SVG fonts -- supported only by Chrome,

#                 Opera and Safari

#svg.hashsalt : None        # if not None, use this string as hash salt

# instead of uuid4

# docstring params

#docstring.hardcopy = False # set this when you want to generate hardcopy docstring

# Set the verbose flags. This controls how much information

# matplotlib gives you at runtime and where it goes. The verbosity
```

```

# levels are: silent, helpful, debug, debug-annoying. Any level is

# inclusive of all the levels below it. If your setting is "debug",

# you'll get all the debug and helpful messages. When submitting

# problems to the mailing-list, please set verbose to "helpful" or "debug"

# and paste the output into your report.

#

# The "fileo" gives the destination for any calls to verbose.report.

# These objects can a filename, or a filehandle like sys.stdout.

#

# You can override the rc default verbosity from the command line by

# giving the flags --verbose-LEVEL where LEVEL is one of the legal

# levels, e.g., --verbose-helpful.

#

# You can access the verbose instance in your code

#     from matplotlib import verbose.

#verbose.level : silent      # one of silent, helpful, debug, debug-annoying

#verbose.fileo : sys.stdout # a log filename, sys.stdout or sys.stderr

# Event keys to interact with figures/plots via keyboard.

# Customize these settings according to your needs.

# Leave the field(s) empty if you don't need a key-map. (i.e., fullscreen : '')

#keymap.fullscreen : f, ctrl+f      # toggling

#keymap.home : h, r, home          # home or reset mnemonic

```

```

#keymap.back : left, c, backspace      # forward / backward keys to enable

#keymap.forward : right, v            #   left handed quick navigation

#keymap.pan : p                      # pan mnemonic

#keymap.zoom : o                    # zoom mnemonic

#keymap.save : s                  # saving current figure

#keymap.quit : ctrl+w, cmd+w       # close the current figure

#keymap.grid : g                  # switching on/off a grid in current axes

#keymap.yscale : l                # toggle scaling of y-axes ('log'/'linear')

#keymap.xscale : L, k              # toggle scaling of x-axes ('log'/'linear')

#keymap.all_axes : a             # enable all axes

# Control location of examples data files

#examples.directory : ''    # directory to look in for custom installation

###ANIMATION settings

#animation.html : 'none'          # How to display the animation as HTML in

                                  # the IPython notebook. 'html5' uses

                                  # HTML5 video tag.

#animation.writer : ffmpeg        # MovieWriter 'backend' to use

#animation.codec : h264           # Codec to use for writing movie

#animation.bitrate: -1           # Controls size/quality tradeoff for movie.

                                  # -1 implies let utility auto-determine

#animation.frame_format: 'png'    # Controls frame format used by temp files

#animation.ffmpeg_path: 'ffmpeg'  # Path to ffmpeg binary. Without full path

```

```

# $PATH is searched

#animation.ffmpeg_args: ''          # Additional arguments to pass to ffmpeg

#animation.avconv_path: 'avconv'    # Path to avconv binary. Without full path

# $PATH is searched

#animation.avconv_args: ''          # Additional arguments to pass to avconv

#animation.mencoder_path: 'mencoder'

# Path to mencoder binary. Without full path

# $PATH is searched

#animation.mencoder_args: ''          # Additional arguments to pass to mencoder

#animation.convert_path: 'convert' # Path to ImageMagick's convert binary.

# On Windows use the full path since convert

# is also the name of a system tool.

```

1.1 tick format (eliminating the automatic offset)

```

In [6]: import matplotlib.pyplot as plt
        import matplotlib.ticker as mtick
        import numpy as np

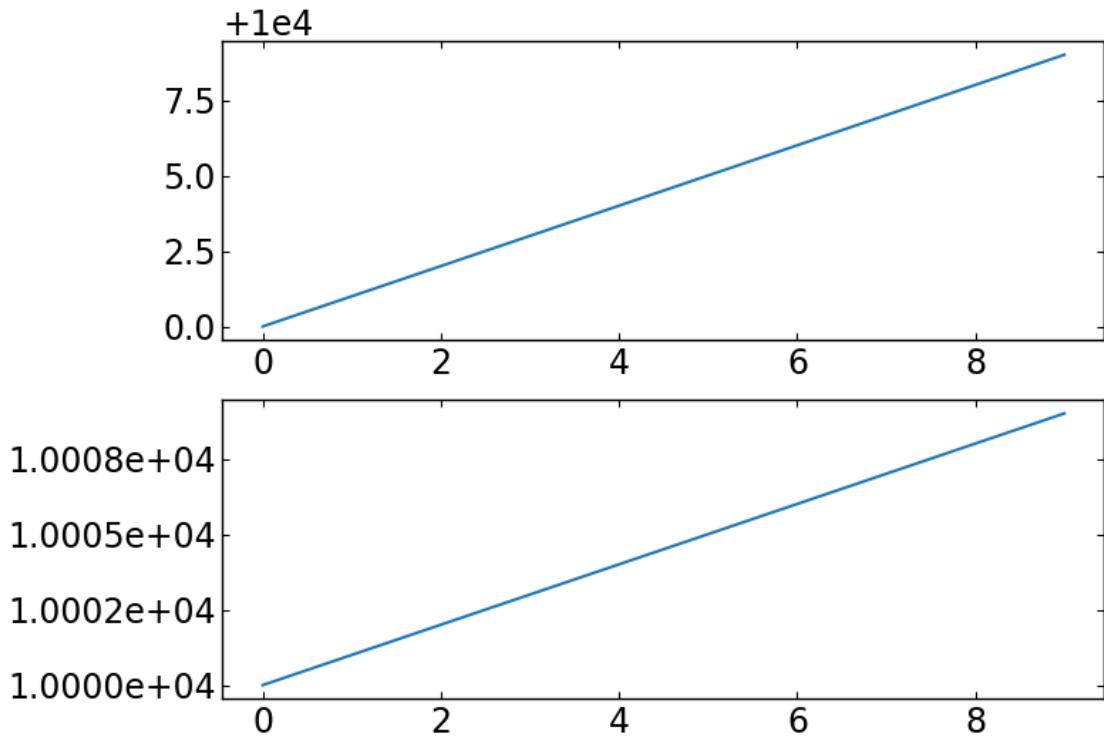
%matplotlib inline

fig,(ax1,ax2)=plt.subplots(2)

ax1.plot(np.arange(10000,10010,1))

ax2.plot(np.arange(10000,10010,1))
ax2.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.4e'))

```



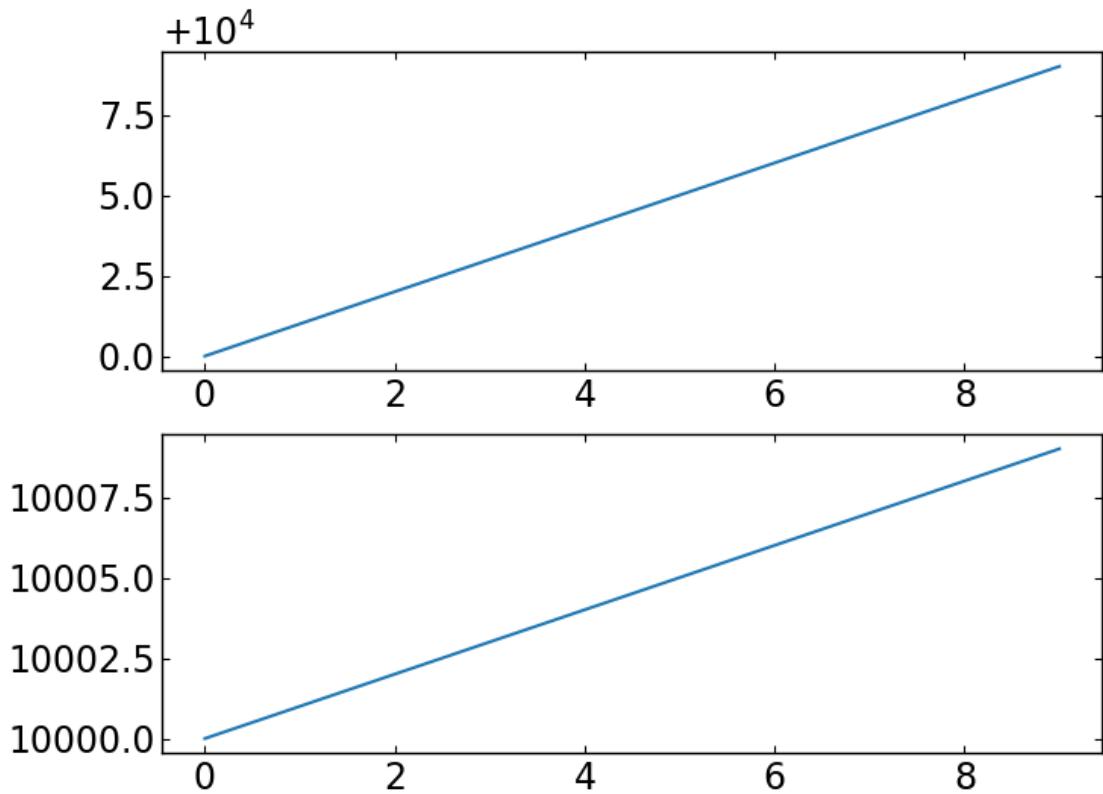
```
In [7]: import matplotlib.pyplot as plt
       import matplotlib.ticker as mtick
       import numpy as np

%matplotlib inline

fig,(ax1,ax2)=plt.subplots(2)

ax1.plot(np.arange(10000,10010,1))
formatter1 = mtick.ScalarFormatter(useMathText=True)
ax1.yaxis.set_major_formatter(formatter1)

ax2.plot(np.arange(10000,10010,1))
formatter = mtick.ScalarFormatter(useOffset=False)
ax2.yaxis.set_major_formatter(formatter)
```



```
In [8]: import matplotlib.pyplot as plt
        import matplotlib.ticker as mtick
        import numpy as np

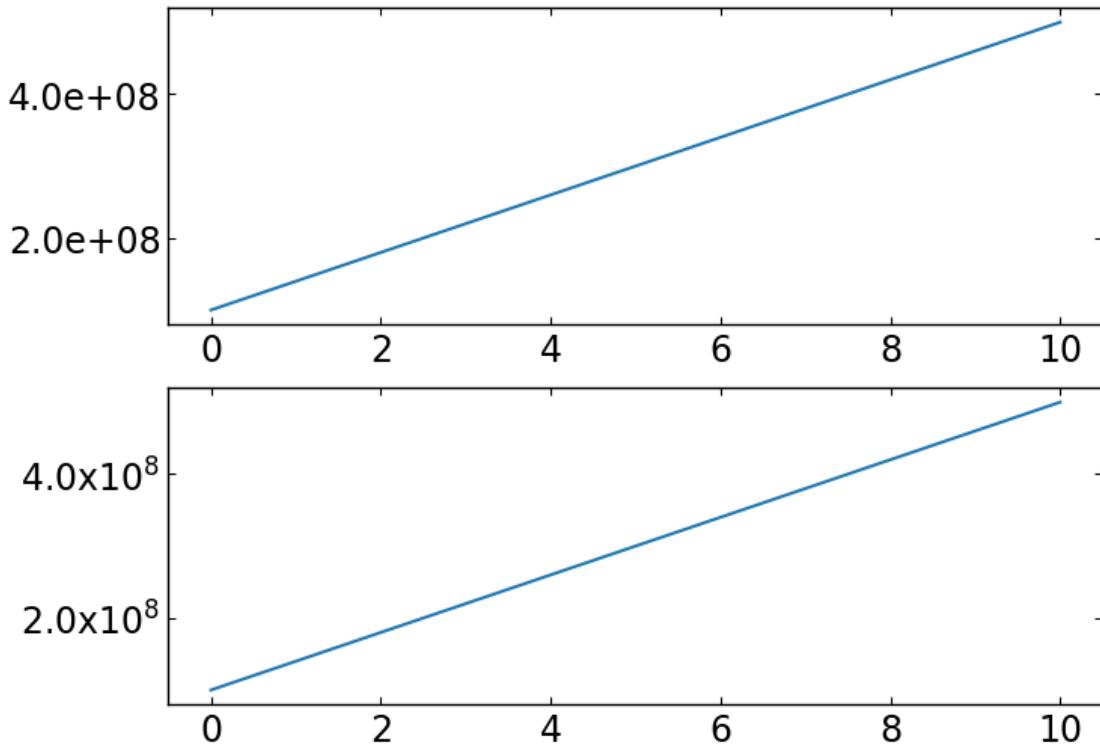
%matplotlib inline

fig,(ax1,ax2)=plt.subplots(2)

x = np.linspace(0,10,100)
y = np.linspace(1e8,5e8,100)

ax1.plot(x,y)
ax1.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.1e'))

# y-value is devided by 1e8 for tick formatter
ax2.plot(x,y/1e8)
ax2.yaxis.set_major_formatter(mtick.FormatStrFormatter(r'%.1fx$10^{\{8\}}$'))
```



1.2 Func formatter

```
In [9]: import matplotlib.ticker as mtick
        import matplotlib.pyplot as plt
        import numpy as np

%matplotlib inline

def format_tick(x,pos=None):
    xabs = np.abs(x)

    if x>0:
        xstr = r'$10^{%i}$'%(xabs)
    elif x<0:
        xstr = r'$-10^{%i}$'%(xabs)
    else:
        xstr = '0'

    return xstr

x = np.arange(10)-5
```

```

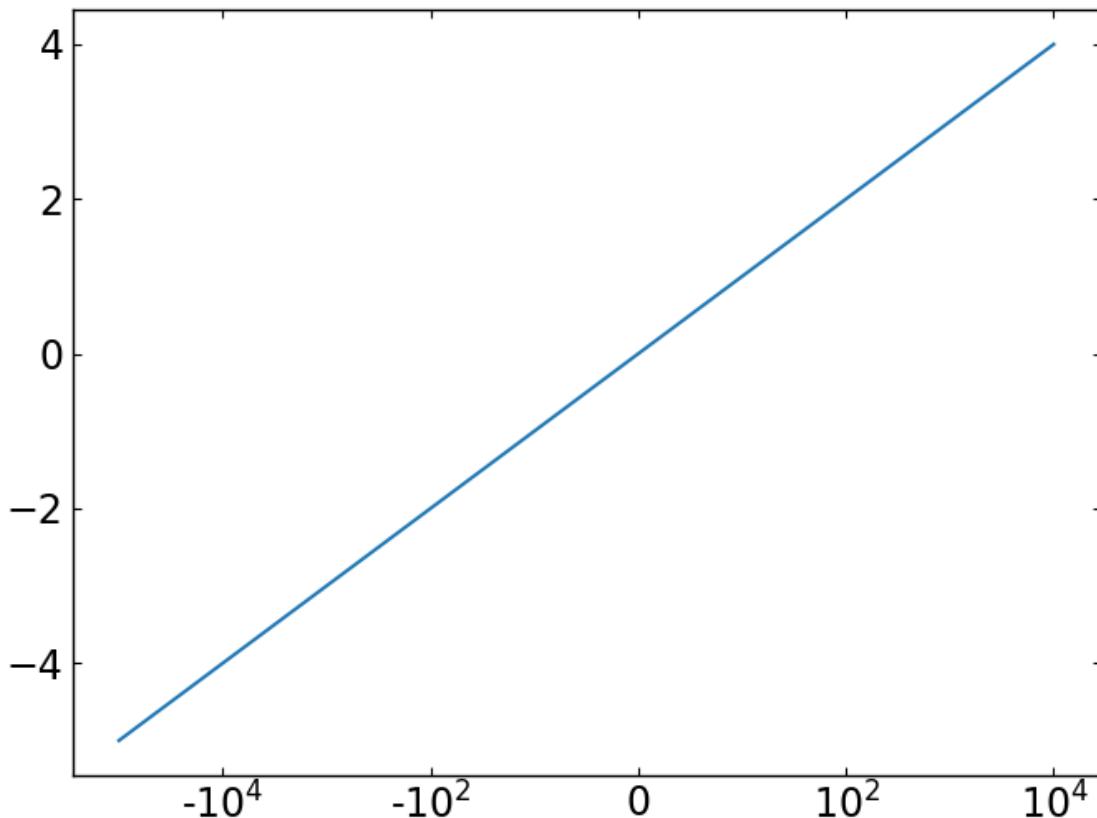
print x

fig,ax = plt.subplots()

ax.plot(x,x)
ax.xaxis.set_major_formatter(mtick.FuncFormatter(format_tick))

[-5 -4 -3 -2 -1  0  1  2  3  4]

```



```

In [10]: import matplotlib.ticker as mtick
        import matplotlib.pyplot as plt
        import numpy as np

%matplotlib inline

def format_tick(x,pos=None):
    xstr_e = '%e' %x
    xsplt = xstr_e.split('e')
    xfact = float(xsplt[0])
    xexp = int(xsplt[1])

```

```

if x != 0:
    xstr = r'%.1fx$10^{%i}$'%(xfact,xexp)
else:
    xstr = '0'

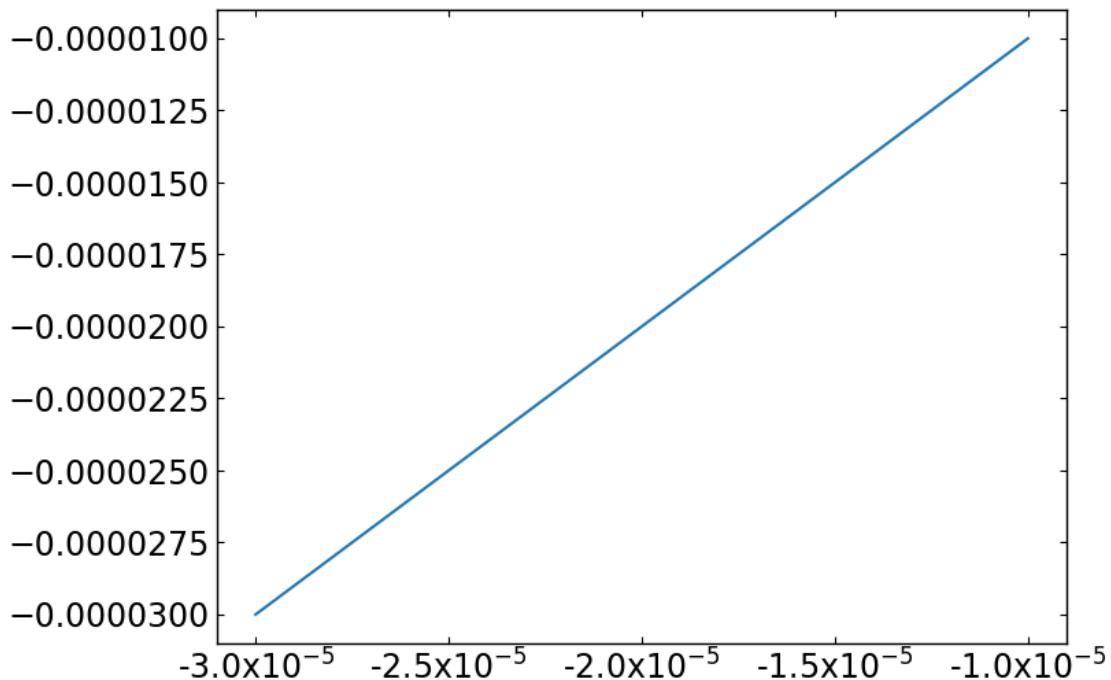
return xstr

x = np.linspace(-1e-5,-3e-5,1000)

fig,ax = plt.subplots()

ax.plot(x,x)
ax.xaxis.set_major_formatter(mtick.FuncFormatter(format_tick))

```



1.3 Major ticks & Minor ticks

```

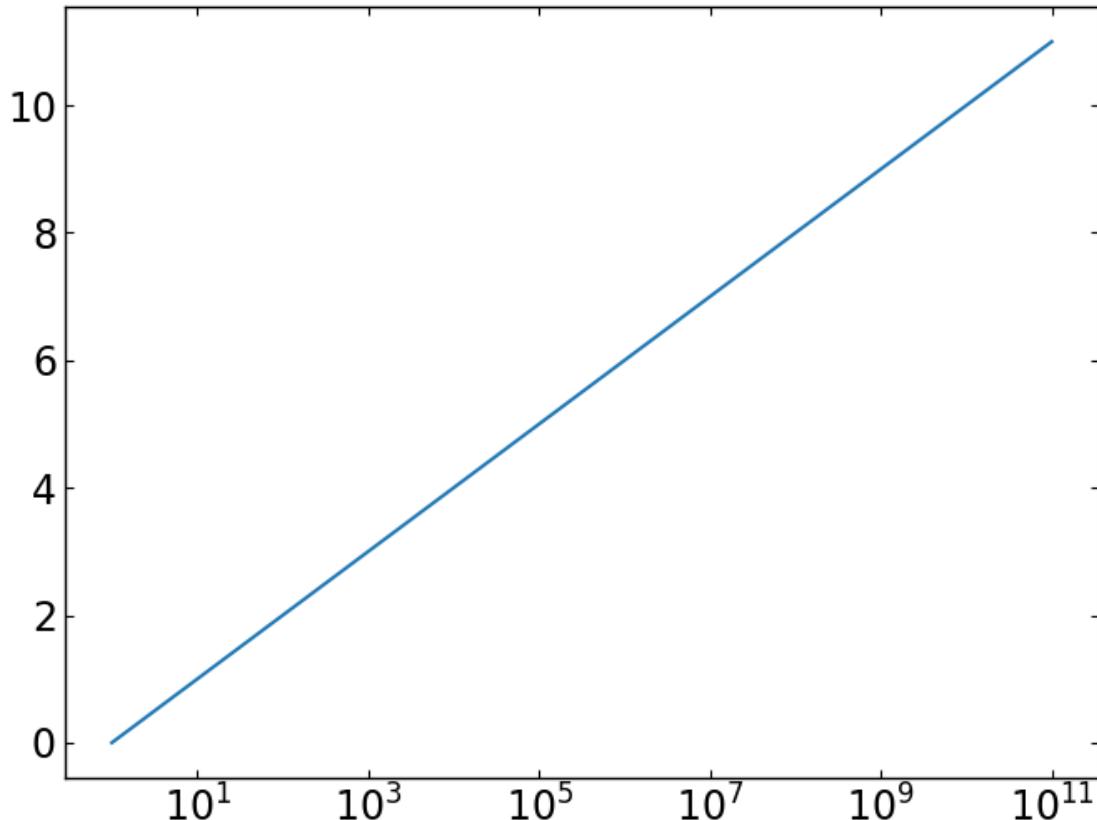
In [11]: import matplotlib.pyplot as plt
        import matplotlib.ticker
        import numpy as np
        %matplotlib inline

y = np.arange(12)
x = 10.0**y

```

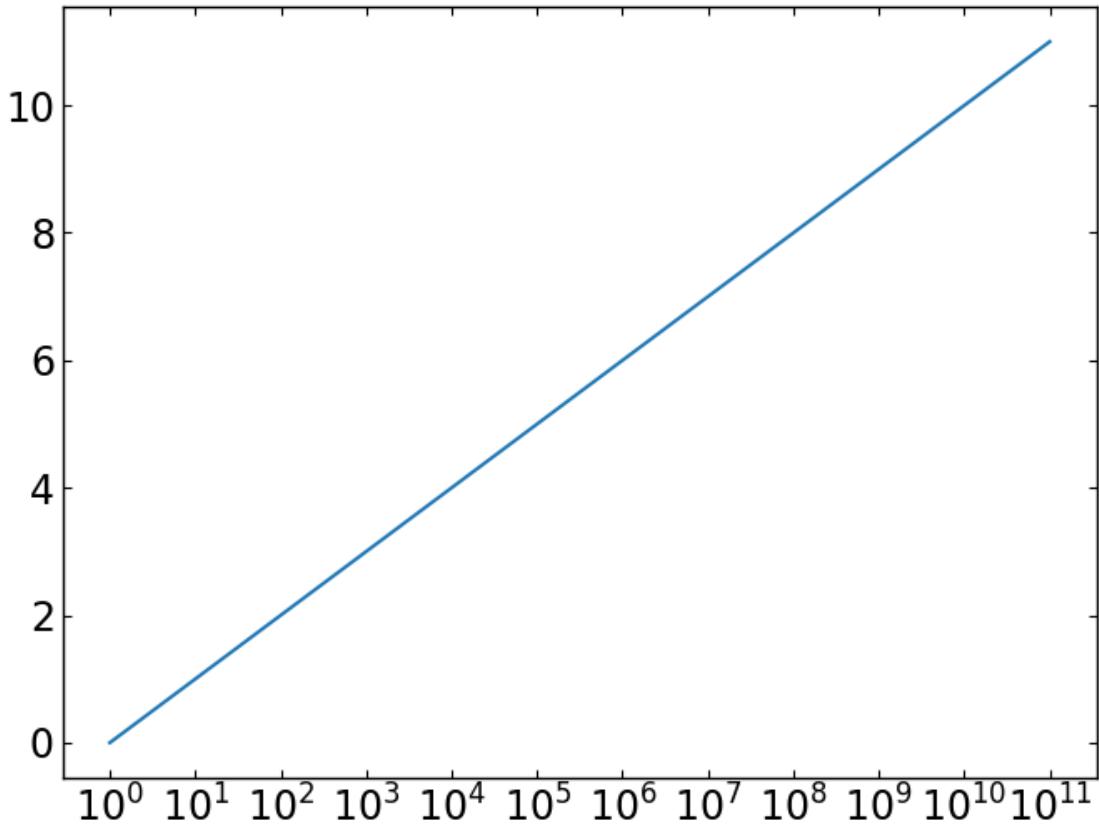
```
fig, ax=plt.subplots()  
ax.semilogx(x,y)
```

Out[11]: [`<matplotlib.lines.Line2D at 0x7f4fc0120e90>`]



In [12]: *""" Log scale; modify number of major ticks (1st method) """*

```
import matplotlib.pyplot as plt  
import matplotlib.ticker  
import numpy as np  
%matplotlib inline  
  
y = np.arange(12)  
x = 10.0**y  
  
fig, ax=plt.subplots()  
ax.semilogx(x,y)  
  
ax.locator_params(axis='x',numticks=12)
```



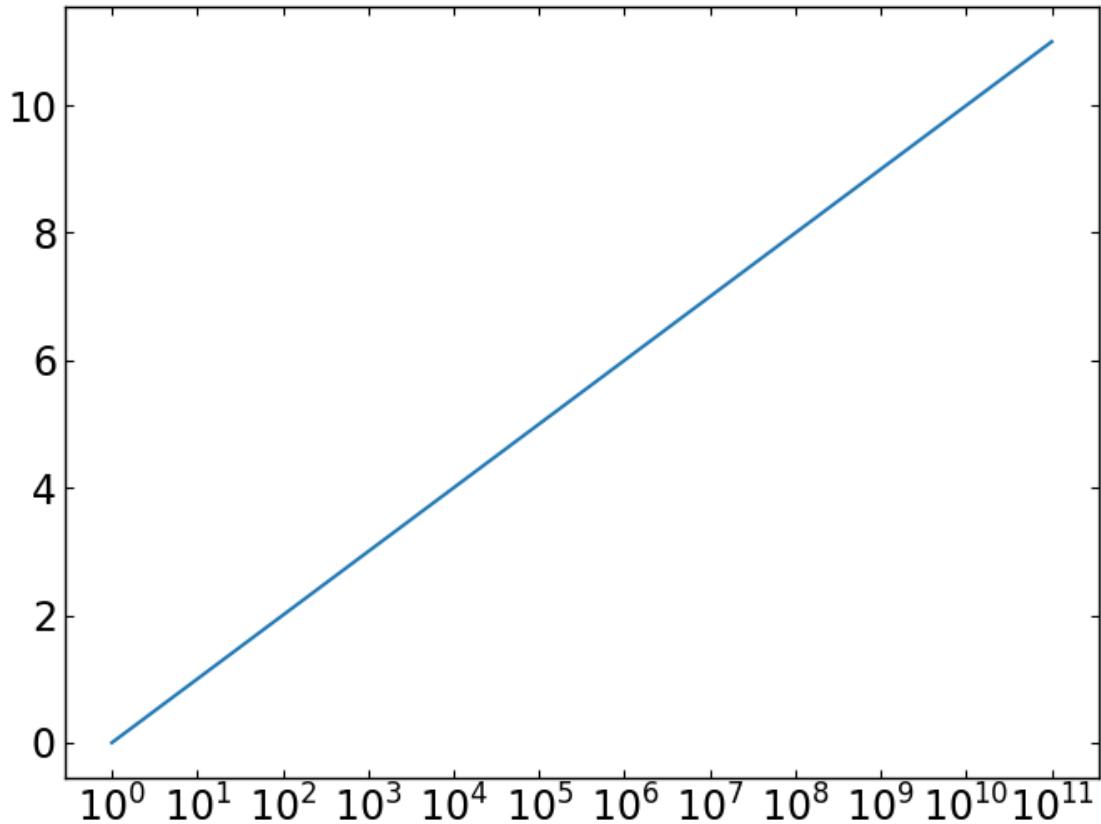
In [13]: *""" Log scale; modify number of major ticks (2nd method) """*

```
import matplotlib.pyplot as plt
import matplotlib.ticker
import numpy as np
%matplotlib inline

y = np.arange(12)
x = 10.0**y

fig, ax=plt.subplots()
ax.semilogx(x,y)

locmaj = matplotlib.ticker.LogLocator(base=10.0, subs=(0.1,1.0, ))
ax.xaxis.set_major_locator(locmaj)
```



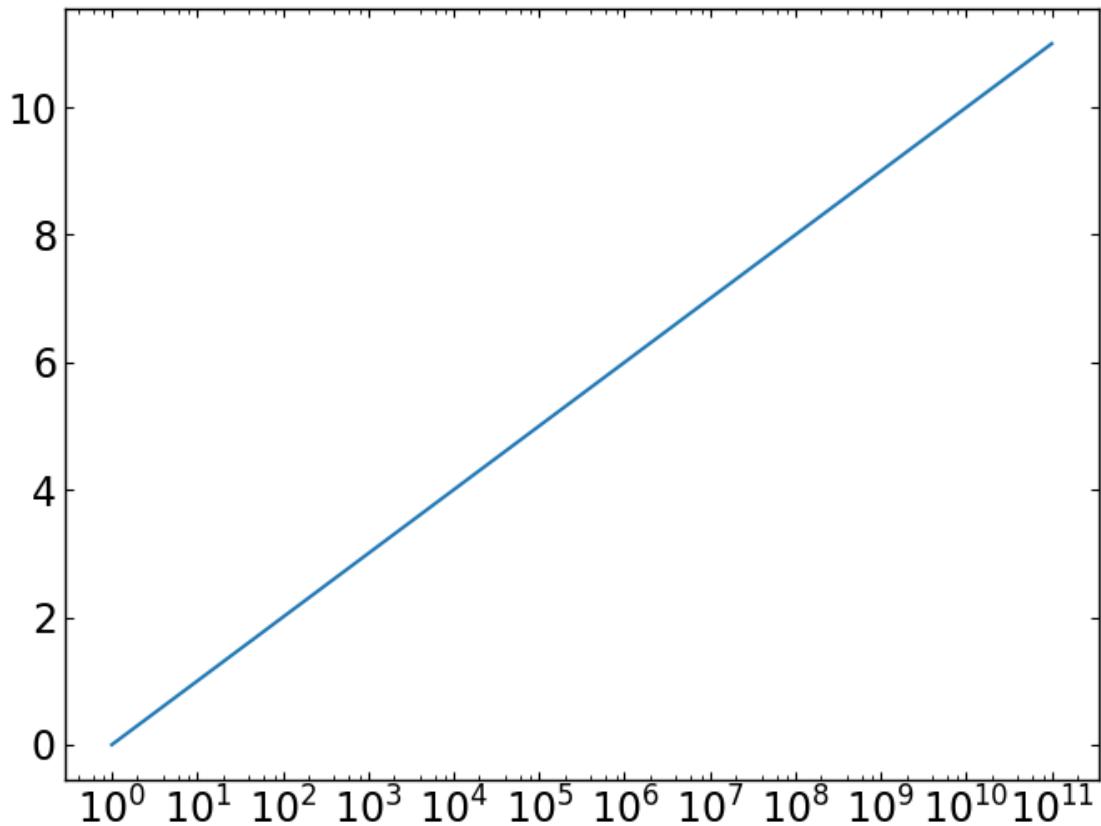
```
In [14]: """ Log scale; modify number of minor ticks """
    import matplotlib.pyplot as plt
    import matplotlib.ticker
    import numpy as np
    %matplotlib inline

    y = np.arange(12)
    x = 10.0**y

    fig, ax=plt.subplots()
    ax.semilogx(x,y)

    locmaj = matplotlib.ticker.LogLocator(base=10.0, subs=(0.1,1.0, ))
    ax.xaxis.set_major_locator(locmaj)

    # sub should be over two decades !!!
    locmin = matplotlib.ticker.LogLocator(base=10.0, subs=(0.1,0.2,0.4,0.6,0.8,1,2,4,6,8,16))
    ax.xaxis.set_minor_locator(locmin)
    ax.xaxis.set_minor_formatter(matplotlib.ticker.NullFormatter())
```



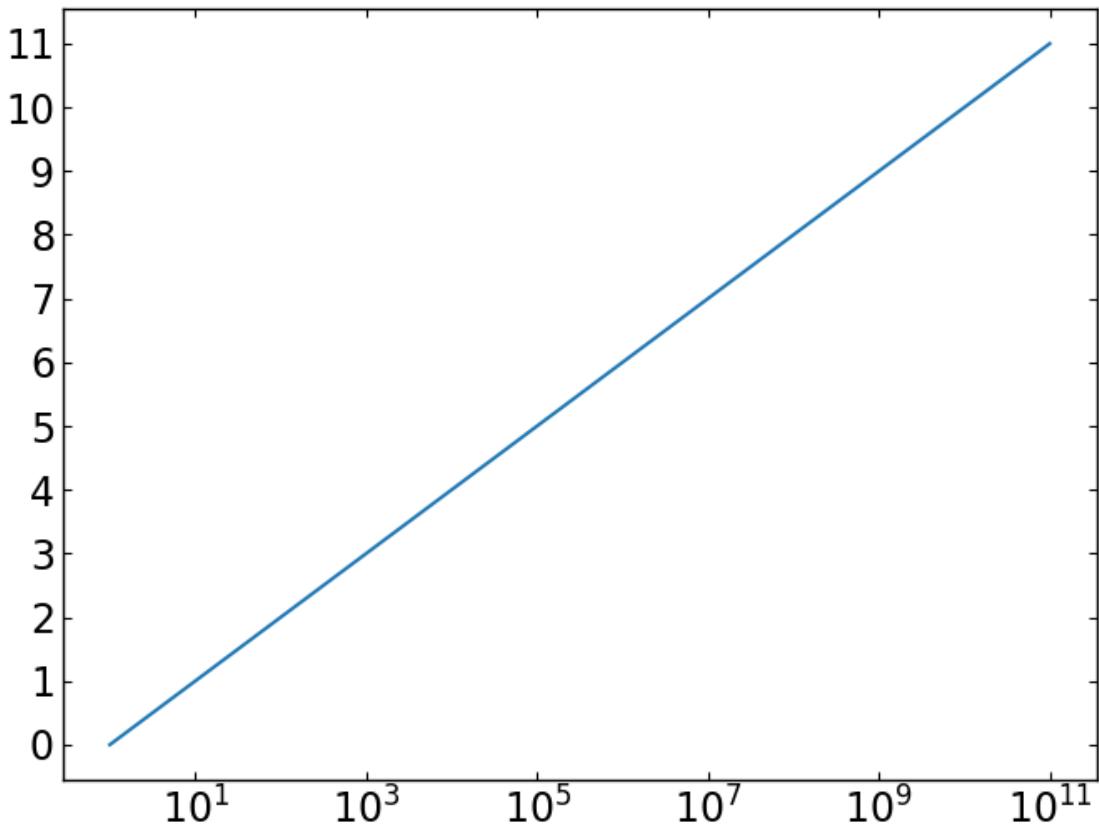
In [15]: """ linear scale; modify number of major ticks """

```
import matplotlib.pyplot as plt
import matplotlib.ticker
import numpy as np
%matplotlib inline

y = np.arange(12)
x = 10.0**y

fig, ax=plt.subplots()
ax.semilogx(x,y)

ax.locator_params(axis='y',nbins=15)
```



In [16]: *"linear scale; specify the major ticks"*

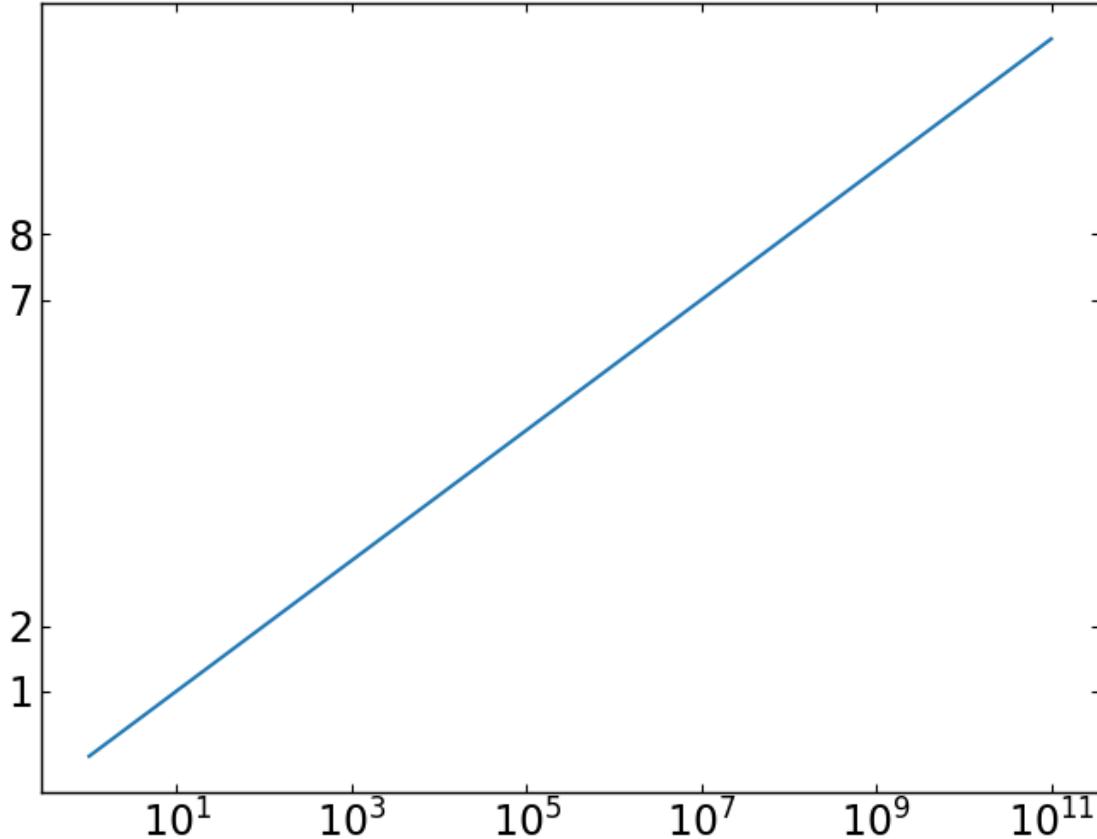
```
import matplotlib.pyplot as plt
import matplotlib.ticker
import numpy as np
%matplotlib inline

y = np.arange(12)
x = 10.0**y

fig, ax=plt.subplots()
ax.semilogx(x,y)

ax.set_yticks((1,2,7,8))
```

Out[16]: [`<matplotlib.axis.YTick at 0x7f4fb30f410>`,
`<matplotlib.axis.YTick at 0x7f4fc0177890>`,
`<matplotlib.axis.YTick at 0x7f4fb40de10>`,
`<matplotlib.axis.YTick at 0x7f4fc802a790>`]



In [17]: """ Special Case: when axis has a data of exponent """

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

%matplotlib inline

x = np.linspace(1,5,10) # represent 10 ~ 10^5
y = x*x

fig, ax = plt.subplots()

ax.plot(x,y)

# recover the format of 10^exponent
ax.xaxis.set_major_formatter(mtick.FormatStrFormatter(r'$10^{%i}$'))

## set minor-tick configuration
xr = ax.get_xlim()
```

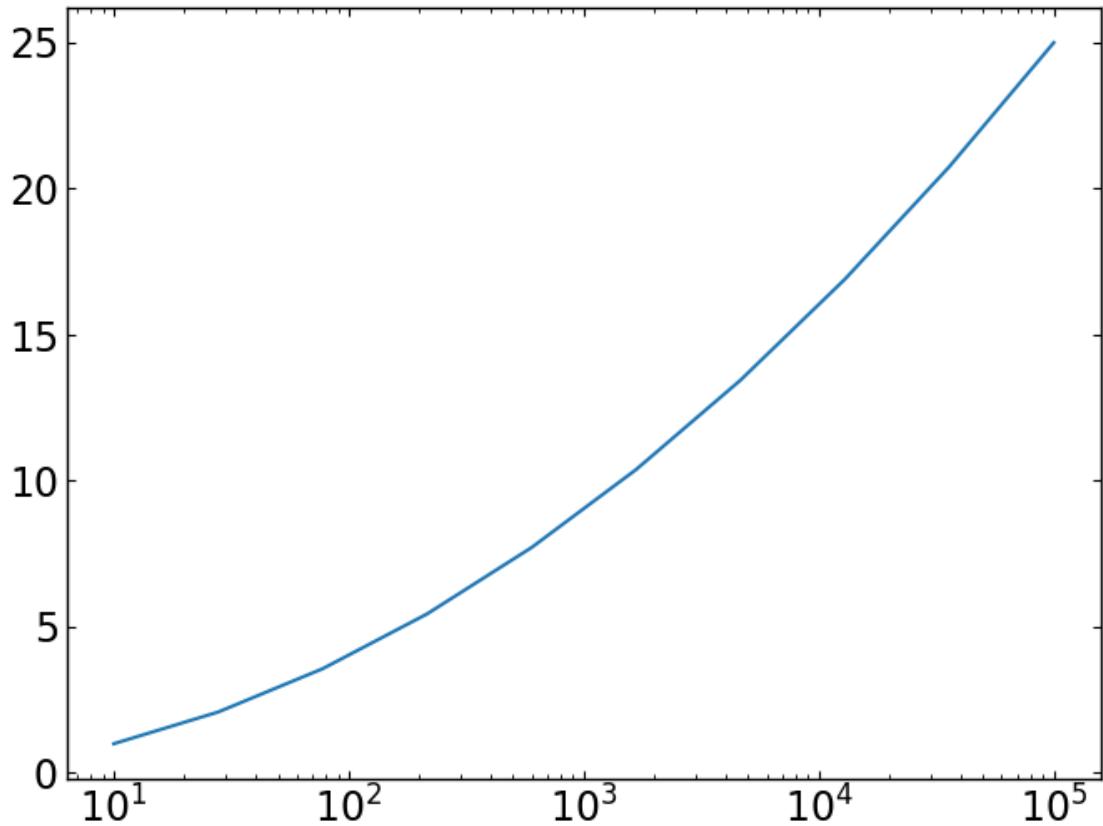
```

xmin = int(xr[0])-1; xmax = int(xr[1])+2
majortk = range(xmin,xmax); lmj = len(majortk)
minortk = np.asarray([np.log10(i) for i in range(2,10)]); lmi = len(minortk)

minortks = np.zeros(lmj*lmi)
for i, imj in enumerate(majortk):
    minortks[i*lmi:i*lmi+lmi] = imj + minortk[:]

locminor = mtick.FixedLocator(minortks)
ax.xaxis.set_minor_locator(locminor)

```



matplotlib3

January 30, 2019

```
In [1]: %matplotlib inline
```

0.1 Styles

```
In [2]: """
```

```
    Reference for line-styles included with Matplotlib.  
    """
```

```
import numpy as np  
import matplotlib.pyplot as plt  
  
color = 'cornflowerblue'  
points = np.ones(5)      # Draw 5 points for each line  
text_style = dict(horizontalalignment='right', verticalalignment='center',  
                  fontsize=12, fontdict={'family': 'monospace'})  
  
def format_axes(ax):  
    ax.margins(0.2)  
    ax.set_axis_off()  
  
def nice_repr(text):  
    return repr(text).lstrip('u')  
  
# Plot all line styles.  
f, ax = plt.subplots()  
  
linestyles = ['-', '--', '-.', ':']  
for y, linestyle in enumerate(linestyles):  
    ax.text(-0.5,y,nice_repr(linestyle), **text_style)  
    #ax.text(-0.5,y,linestyle, **text_style)  
    ax.plot(y*points, linestyle=linestyle, color=color, linewidth=3)  
    format_axes(ax)  
    ax.set_title('line style')  
  
plt.show()
```

line style



In [3]: """

Reference for marker fill-styles included with Matplotlib.
"""

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

points = np.ones(5)    # Draw 3 points for each line
text_style = dict(horizontalalignment='right', verticalalignment='center',
                  fontsize=12, fontdict={'family': 'monospace'})
marker_style = dict(color='cornflowerblue', linestyle=':', marker='o',
                     markersize=15, markerfacecoloralt='gray')

def format_axes(ax):
    ax.margins(0.2)
    ax.set_axis_off()

def nice_repr(text):
    return repr(text).lstrip('u')
```

```

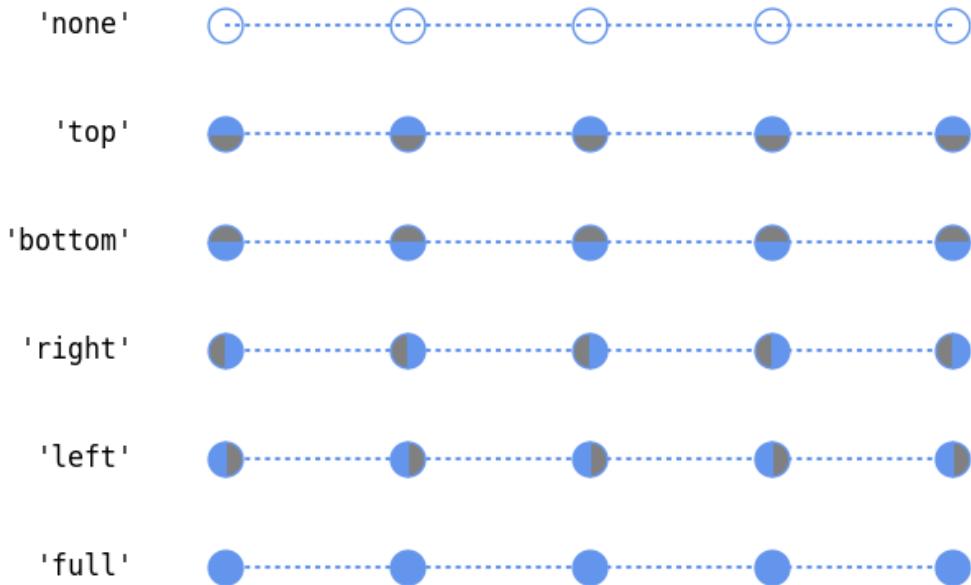
fig, ax = plt.subplots()

# Plot all fill styles.
for y, fill_style in enumerate(Line2D.fillStyles):
    #print y,y*points
    ax.text(-0.5, y, nice_repr(fill_style), **text_style)
    ax.plot(y*points, fillstyle=fill_style, **marker_style)
format_axes(ax)
ax.set_title('fill style')

plt.show()

```

fill style



In [2]: """

Reference for filled- and unfilled-marker types included with Matplotlib.
"""

```

from six import iteritems
import numpy as np
import matplotlib.pyplot as plt

```

```

from matplotlib.lines import Line2D

points = np.ones(3)      # Draw 3 points for each line
text_style = dict(horizontalalignment='right', verticalalignment='center',
                  fontsize=12, fontdict={'family': 'monospace'})
marker_style = dict(linestyle=':', color='cornflowerblue', markersize=10)

def format_axes(ax):
    ax.margins(0.2)
    ax.set_axis_off()

def nice_repr(text):
    return repr(text).lstrip('u')

def split_list(a_list):
    i_half = len(a_list) // 2
    return (a_list[:i_half], a_list[i_half:])

# Plot all un-filled markers
# ----

fig, axes = plt.subplots(ncols=2)

# Filter out filled markers and marker settings that do nothing.
# We use iteritems from six to make sure that we get an iterator
# in both python 2 and 3

unfilled_markers = [m for m, func in iteritems(Line2D.markers)
                    if func != 'nothing' and m not in Line2D.filled_markers]

# Reverse-sort for pretty. We use our own sort key which is essentially
# a python3 compatible reimplementation of python2 sort.

unfilled_markers = sorted(unfilled_markers,
                          key=lambda x: (str(type(x)), str(x))[:-1])

for ax, markers in zip(axes, split_list(unfilled_markers)):
    for y, marker in enumerate(markers):
        ax.text(-0.5, y, nice_repr(marker), **text_style)
        ax.plot(y*points, marker=marker, **marker_style)
    format_axes(ax)
fig.suptitle('unfilled markers', fontsize=14)

# Plot all filled markers.
# ----

fig, axes = plt.subplots(ncols=2)

```

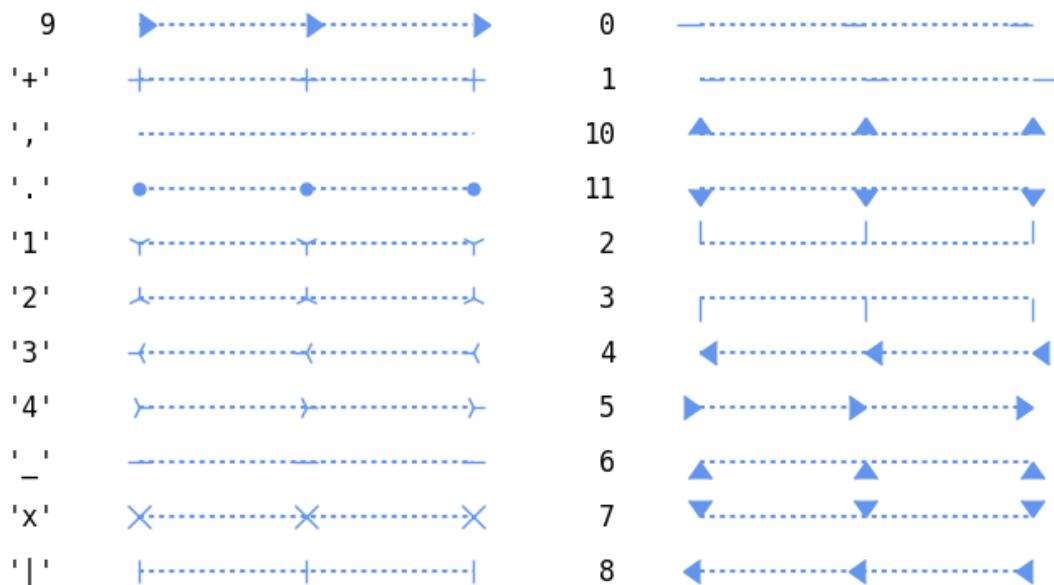
```

for ax, markers in zip(axes, split_list(Line2D.filled_markers)):
    for y, marker in enumerate(markers):
        ax.text(-0.5, y, nice_repr(marker), **text_style)
        ax.plot(y*points, marker=marker, **marker_style)
    format_axes(ax)
fig.suptitle('filled markers', fontsize=14)

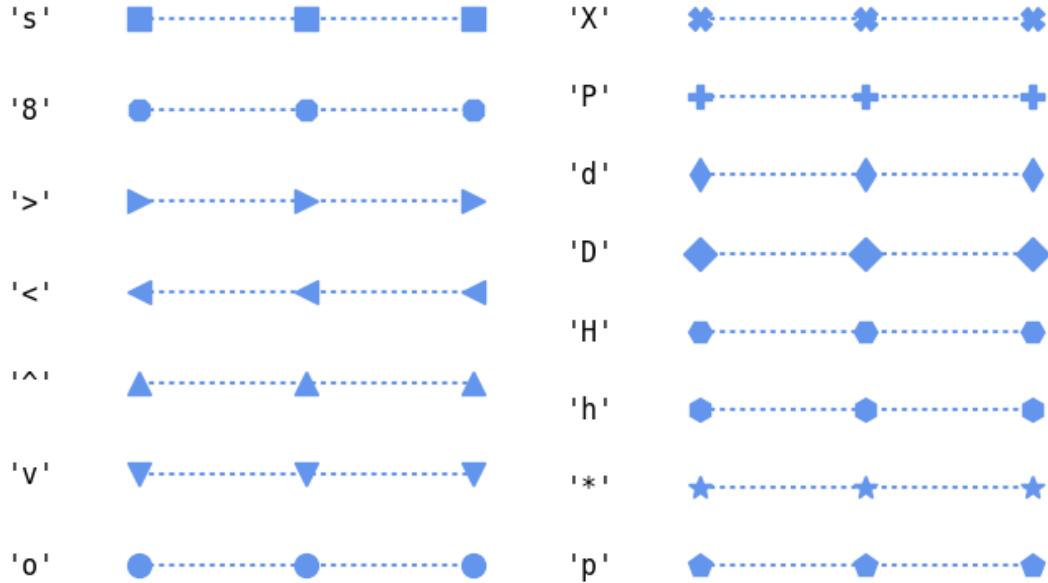
plt.show()

```

unfilled markers



filled markers



In [13]: *""" Marker w/ different alpha between face and edge (png,pdf ...) """*

```
import matplotlib.pyplot as plt
import matplotlib.colors as cls
import numpy as np
import os

%matplotlib inline

x = np.arange(10)
y1 = x**2
y2 = x**2+10
y3 = x**2+20
y4 = x**2+30
y5 = x**2+40

clr = 'black'
# transform the color to tuple (r,g,b,a) --> "a" represents alpha
```

```

clralp1 = cls.to_rgba(clr,0.1)
clralp2 = cls.to_rgba(clr,0.3)
clralp3 = cls.to_rgba(clr,0.5)
clralp4 = cls.to_rgba(clr,0.7)
clralp5 = cls.to_rgba(clr,0.9)

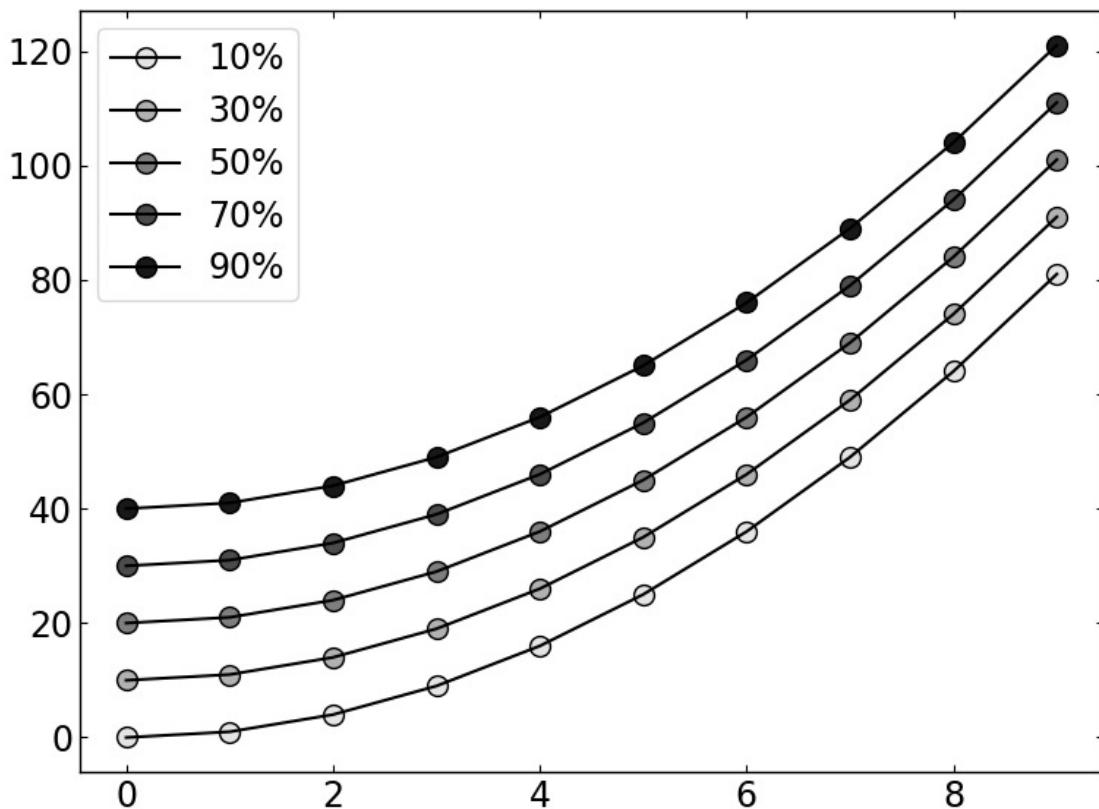
fig,ax = plt.subplots()
ax.plot(x,y1,marker='o',color=clr,ms=10,mfc=clralp1,mec=clr,label='10%')
ax.plot(x,y2,marker='o',color=clr,ms=10,mfc=clralp2,mec=clr,label='30%')
ax.plot(x,y3,marker='o',color=clr,ms=10,mfc=clralp3,mec=clr,label='50%')
ax.plot(x,y4,marker='o',color=clr,ms=10,mfc=clralp4,mec=clr,label='70%')
ax.plot(x,y5,marker='o',color=clr,ms=10,mfc=clralp5,mec=clr,label='90%')

ax.legend(loc='best')

homedir = os.environ['HOME']
figdir = homedir+='/CodeExr/Python/figures/'

fig.tight_layout()
fig.savefig(figdir+'markeralpha.png')

```



In [23]: """ Marker w/ different alpha between face and edge (eps ...)

Since the post-script files cannot deal with the transparency, we need to draw the line (non-transparent) and symbol (transparent) separately. For the transparency of the symbol face color, we use the `mimic_alpha.colorAlpha_` for generate (r,g,b) tuple, at which the alpha effect is applied.

Since one data-set consists of two components (line & symbol), the legend should

```

import matplotlib.pyplot as plt
import matplotlib.colors as cls
from mimic_alpha import colorAlpha_to_rgb as calp
import numpy as np
import os

%matplotlib inline

x = np.arange(10)
y1 = x**2
y2 = x**2+10
y3 = x**2+20
y4 = x**2+30
y5 = x**2+40

clr = 'black'
# transform the color to tuple (r,g,b,a) --> a represents alpha
clralp1 = calp(clr,0.1)[0]
clralp2 = calp(clr,0.3)[0]
clralp3 = calp(clr,0.5)[0]
clralp4 = calp(clr,0.7)[0]
clralp5 = calp(clr,0.9)[0]

fig,ax = plt.subplots(figsize=(8,6))

mk1, = ax.plot(x,y1,'o',color=clr,ms=10,mfc=clralp1)
ln1, = ax.plot(x,y1,'o-',color=clr,ms=10,fillstyle='none',mec=clr)
mk2, = ax.plot(x,y2,'o',color=clr,ms=10,mfc=clralp2)
ln2, = ax.plot(x,y2,'o-',color=clr,ms=10,fillstyle='none',mec=clr)
mk3, = ax.plot(x,y3,'o',color=clr,ms=10,mfc=clralp3)
ln3, = ax.plot(x,y3,'o-',color=clr,ms=10,fillstyle='none',mec=clr)
mk4, = ax.plot(x,y4,'o',color=clr,ms=10,mfc=clralp4)
ln4, = ax.plot(x,y4,'o-',color=clr,ms=10,fillstyle='none',mec=clr)
mk5, = ax.plot(x,y5,'o',color=clr,ms=10,mfc=clralp5)
ln5, = ax.plot(x,y5,'o-',color=clr,ms=10,fillstyle='none',mec=clr)

ax.legend([(mk1,ln1),(mk2,ln2),(mk3,ln3),(mk4,ln4),(mk5,ln5)] \
 ,["10%","30%","50%","70%","90%"],loc='best')

homedir = os.environ['HOME']

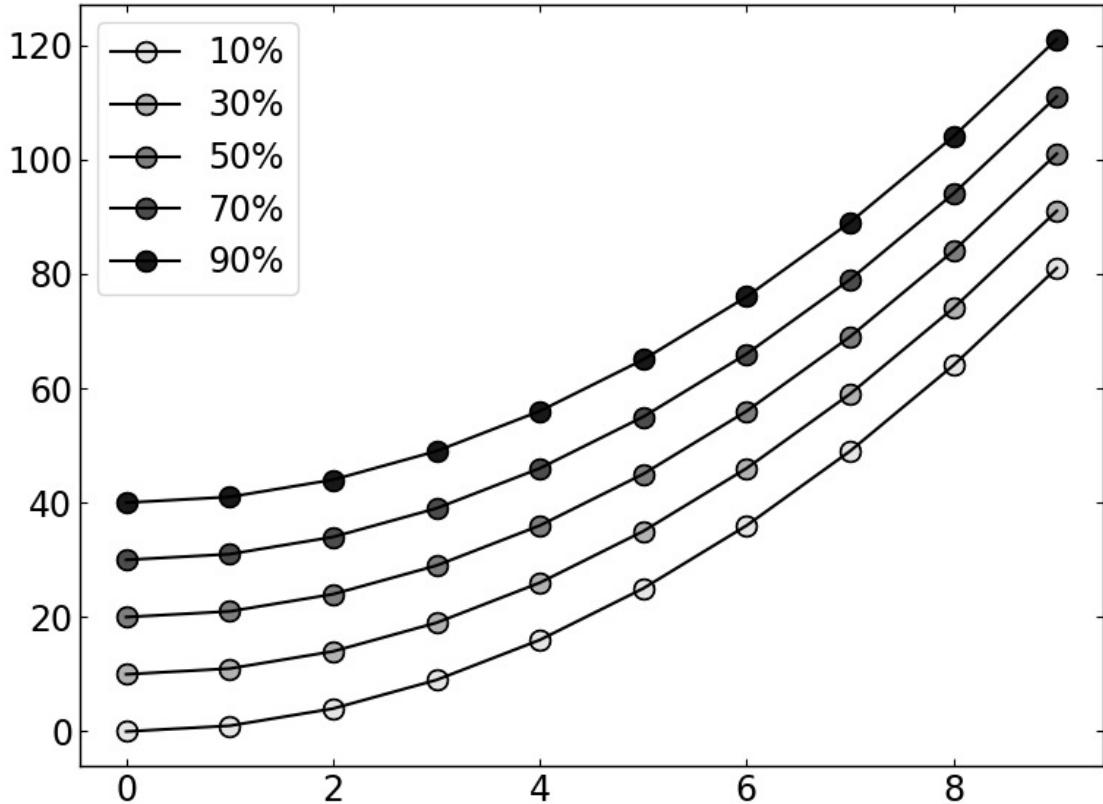
```

```

figdir = homedir+'CodeExr/Python/figures/'

fig.tight_layout()
fig.savefig(figdir+'markeralpha.eps')

```



In [4]: *""" Reference of Colors """*

```

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(0,100,100)
y0,y1,y2,y3,y4,y5,y6,y7,y8,y9 = 2*x,1.8*x,1.6*x,1.4*x,1.2*x,x,0.8*x,0.6*x,0.4*x,0.2*x

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)

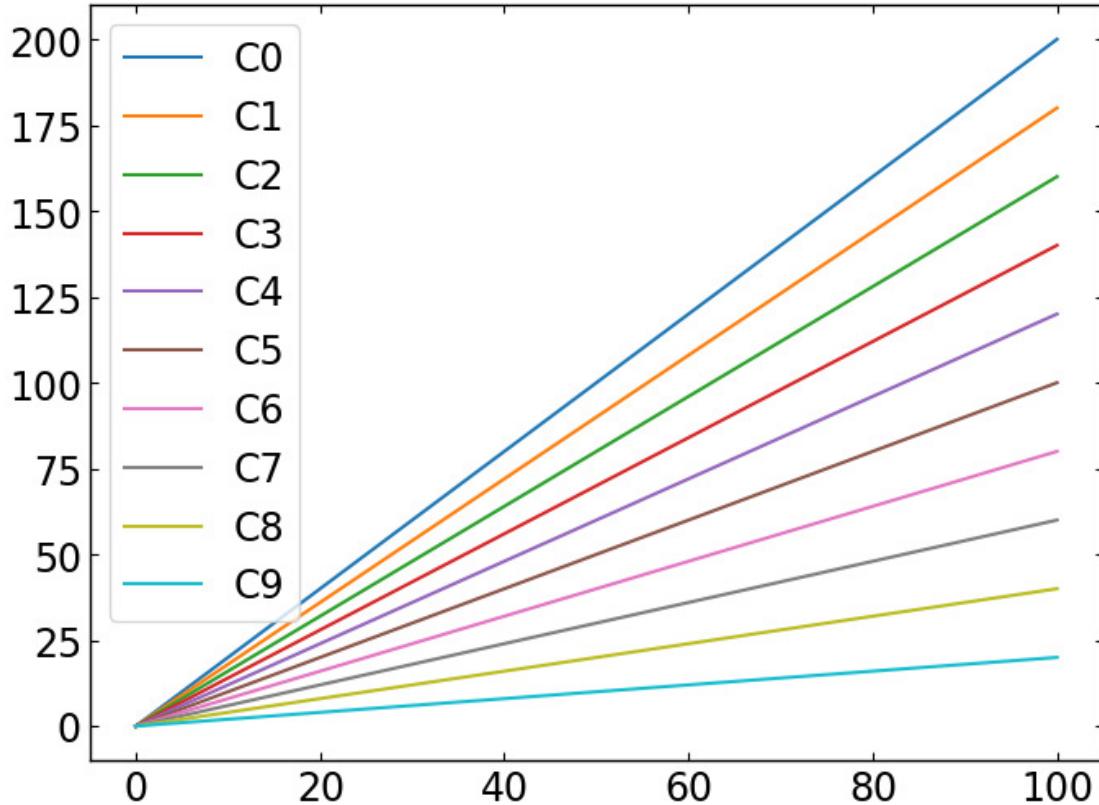
ax.plot(x,y0,label='C0')
ax.plot(x,y1,label='C1')
ax.plot(x,y2,label='C2')
ax.plot(x,y3,label='C3')

```

```
ax.plot(x,y4,label='C4')
ax.plot(x,y5,label='C5')
ax.plot(x,y6,label='C6')
ax.plot(x,y7,label='C7')
ax.plot(x,y8,label='C8')
ax.plot(x,y9,label='C9')

ax.legend(loc='best')
```

Out [4]: <matplotlib.legend.Legend at 0x7f8e04cd8bd0>



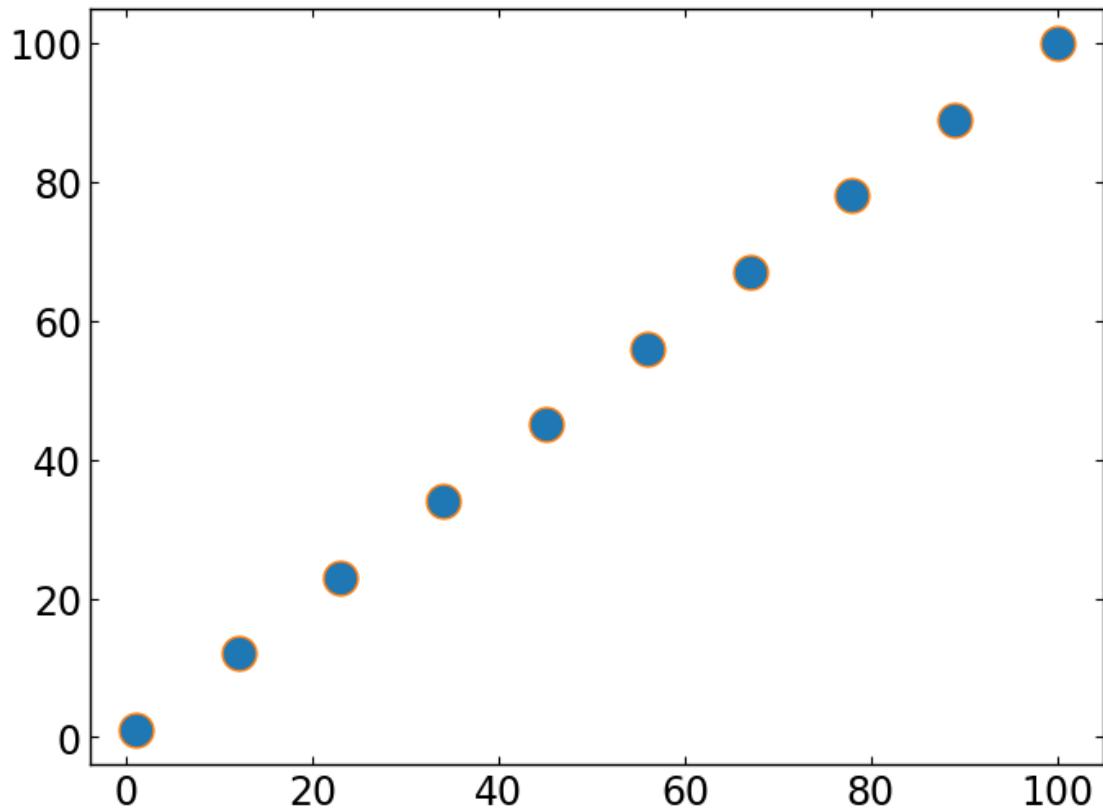
```
In [47]: import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(1,100,10)
y = x

fig,ax = plt.subplots(figsize=(8,6))

ax.plot(x,y,'o',markersize=15,fillstyle='full',zorder=0,markeredgecolor='C1')
```

Out [47]: [<matplotlib.lines.Line2D at 0x7fade1811610>]



0.2 Play with ERRORS

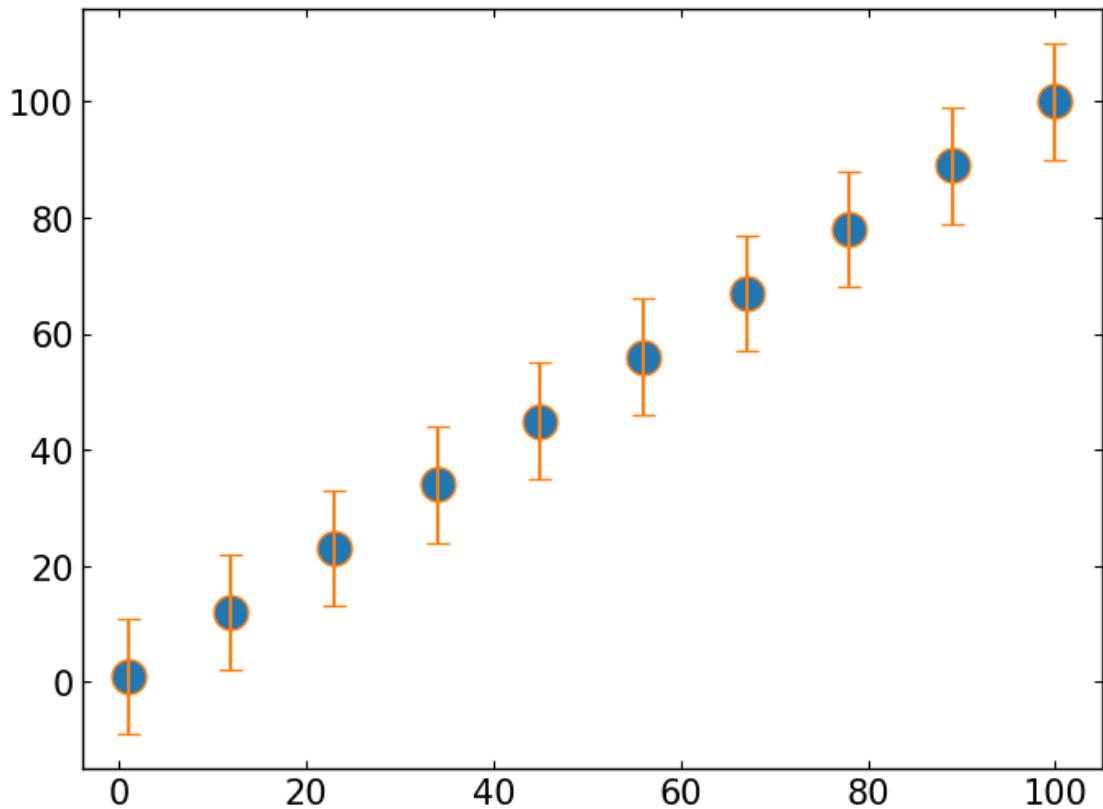
```
In [25]: import numpy as np
         import matplotlib.pyplot as plt

         x = np.linspace(1,100,10)
         y = x

         fig,ax = plt.subplots()

         ax.errorbar(x,y,yerr=10,fmt='o',markersize=15,fillstyle='full',markeredgecolor='C1' \
                     ,ecolor='C1',capsize=5,barsabove=True)
```

```
Out[25]: <Container object of 3 artists>
```



```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        # example data
x = np.arange(0.1, 4, 0.1)
y = np.exp(-x)

        # example variable error bar values
yerr = 0.1 + 0.1 * np.sqrt(x)

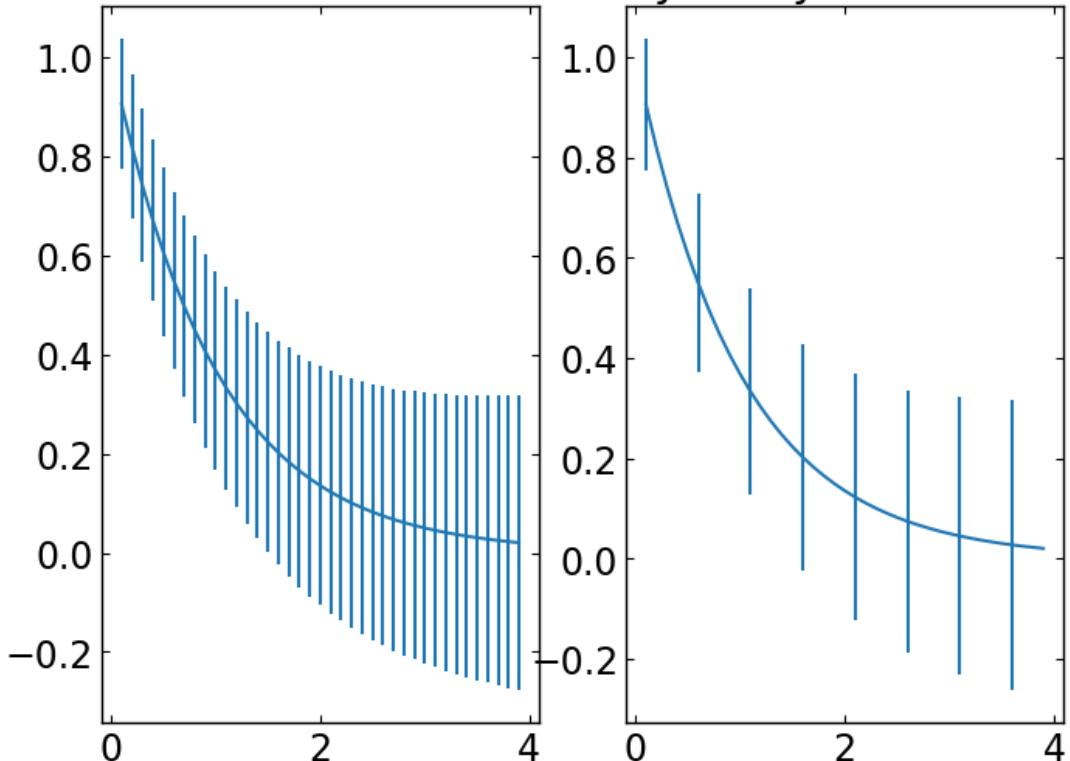
        # Now switch to a more OO interface to exercise more features.
fig, axs = plt.subplots(nrows=1, ncols=2, sharex=True)
ax = axs[0]
ax.errorbar(x, y, yerr=yerr)
ax.set_title('all errorbars')

ax = axs[1]
ax.errorbar(x, y, yerr=yerr, errorevery=5)
ax.set_title('only every 5th errorbar')
```

```
fig.suptitle('Errorbar subsampling for better appearance')
```

```
plt.show()
```

Errorbar subsampling for better appearance
all errorbars only every 5th errorbar



0.2.1 Draw the 2D image in polar coordinate

```
In [6]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib as mpl

%matplotlib inline

Nr, Nth = 500, 500
r = np.linspace(1.,100.,Nr)
th = np.linspace(0.,2.*np.pi,Nth)

d = np.zeros((Nr,Nth), dtype=float)
for i,ir in enumerate(r):
    d[i,:] = 1./ir**2.
```

```

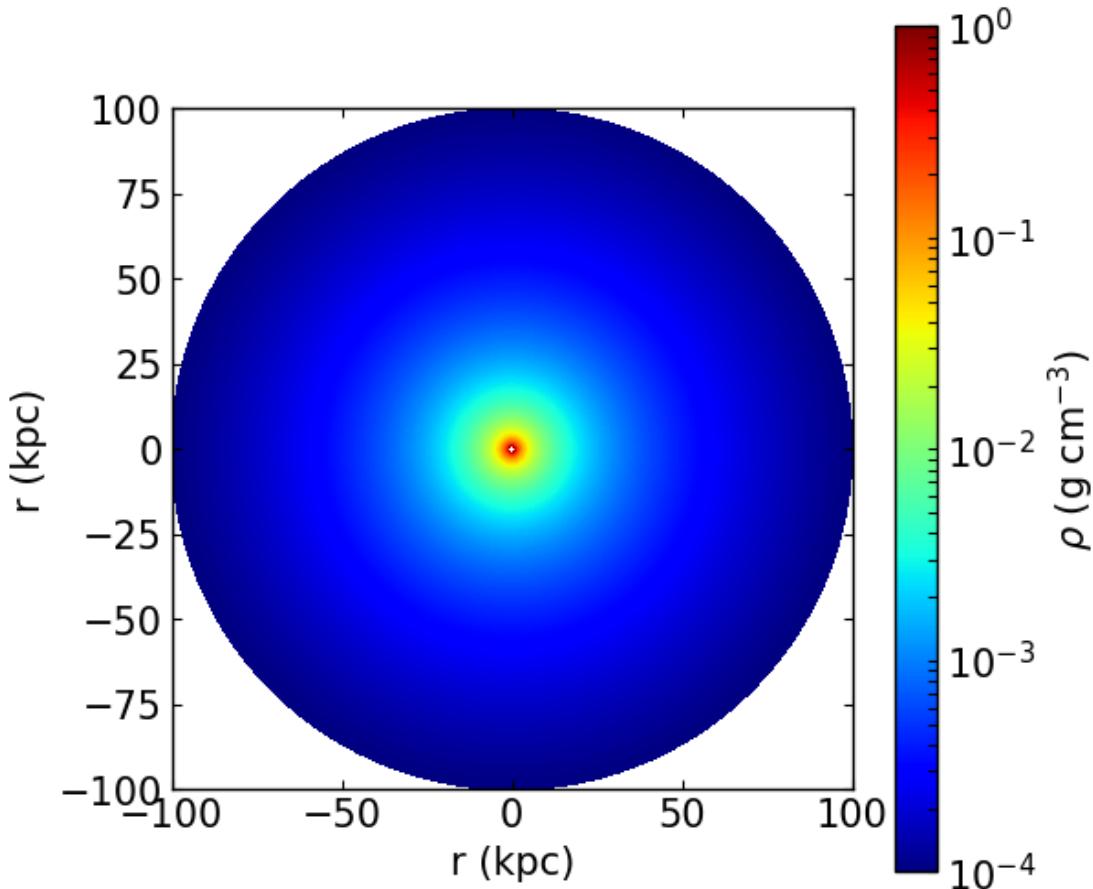
r2d, th2d = np.meshgrid(r,th,indexing='ij')
x2d, y2d = r2d*np.sin(th2d), r2d*np.cos(th2d)

fig = plt.figure(figsize=(6,6))
#ax1 = fig.add_subplot(111, axisbg='white')      # axisbg is not working for version >2.0
ax1 = fig.add_subplot(111)
ax1.set_aspect('equal')

norm = mpl.colors.LogNorm(vmin=d.min(),vmax=d.max())
p = ax1.pcolormesh(x2d,y2d, d, norm=norm, cmap='jet')
ax1.set_xlabel('r (kpc)')
ax1.set_ylabel('r (kpc)')

# colorbar
clb = fig.colorbar(p,ax=ax1,label=r'$\rho$ (g cm$^{-3}$)')
#plt.locator_params(nbins=2)

```



In [7]: import numpy as np

```

import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.axes_grid1 import make_axes_locatable      # adjusted colorbar

%matplotlib inline

Nr, Nth = 500, 500
r = np.linspace(1.,100.,Nr)
th = np.linspace(0.,2.*np.pi,Nth)

d = np.zeros((Nr,Nth), dtype=float)
for i,ir in enumerate(r):
    d[i,:] = 1./ir**2.

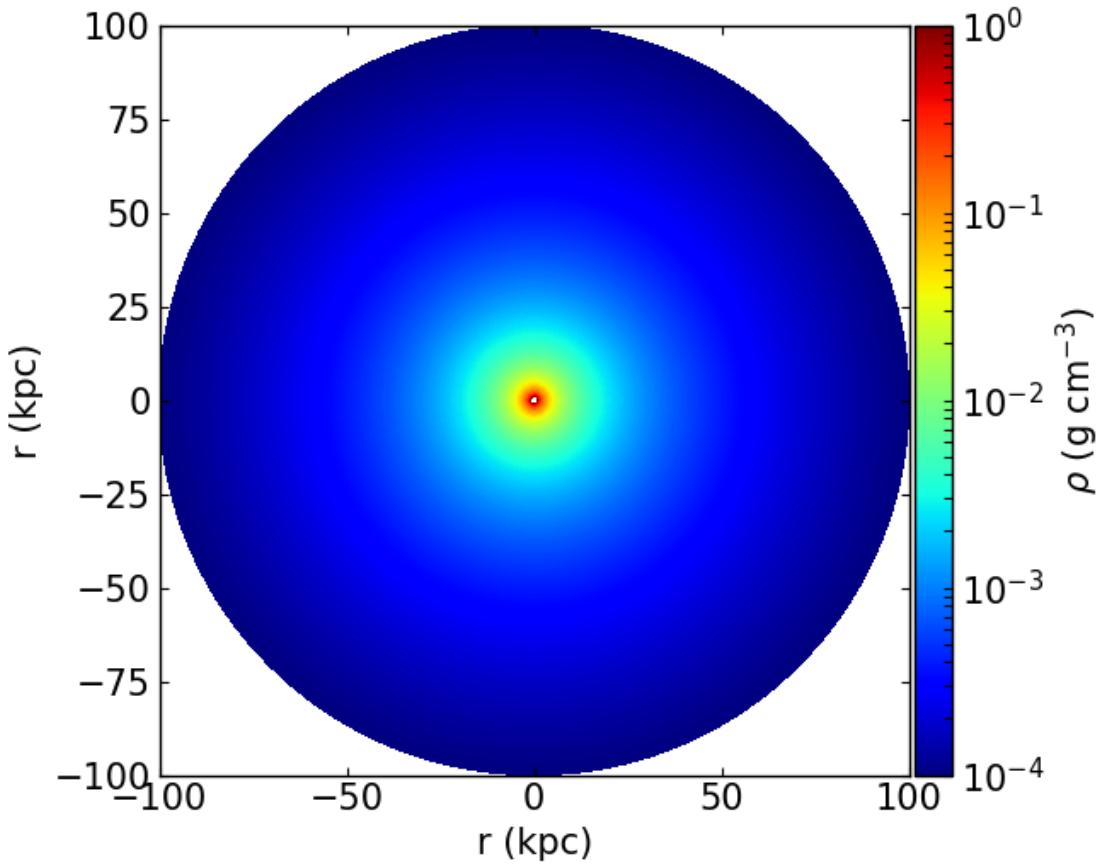
r2d, th2d = np.meshgrid(r,th,indexing='ij')
x2d, y2d = r2d*np.sin(th2d), r2d*np.cos(th2d)

fig = plt.figure(figsize=(6,6))
#ax1 = fig.add_subplot(111, axisbg='white')
ax1 = fig.add_subplot(111)
ax1.set_aspect('equal')

norm = mpl.colors.LogNorm(vmin=d.min(),vmax=d.max())
p = ax1.pcolormesh(x2d,y2d, d, norm=norm, cmap='jet')
ax1.set_xlabel('r (kpc)')
ax1.set_ylabel('r (kpc)')

divider = make_axes_locatable(ax1)
cax1 = divider.append_axes('right', size='5%', pad=0.03)
# colorbar
clb = fig.colorbar(p,ax=ax1,cax=cax1,label=r'$\rho$ (g cm$^{-3}$)')

```



```
In [8]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.axes_grid1 import make_axes_locatable      # adjusted colorbar

%matplotlib inline

Nr, Nth = 500, 500
r = np.linspace(1.,100.,Nr)
th = np.linspace(0.,2.*np.pi,Nth)

d = np.zeros((Nr,Nth), dtype=float)
for i,ir in enumerate(r):
    d[i,:] = 1./ir**2.

r2d, th2d = np.meshgrid(r,th,indexing='ij')
x2d, y2d = r2d*np.sin(th2d), r2d*np.cos(th2d)

fig = plt.figure(figsize=(6,6))
#ax1 = fig.add_subplot(111, axisbg='white')
```

```

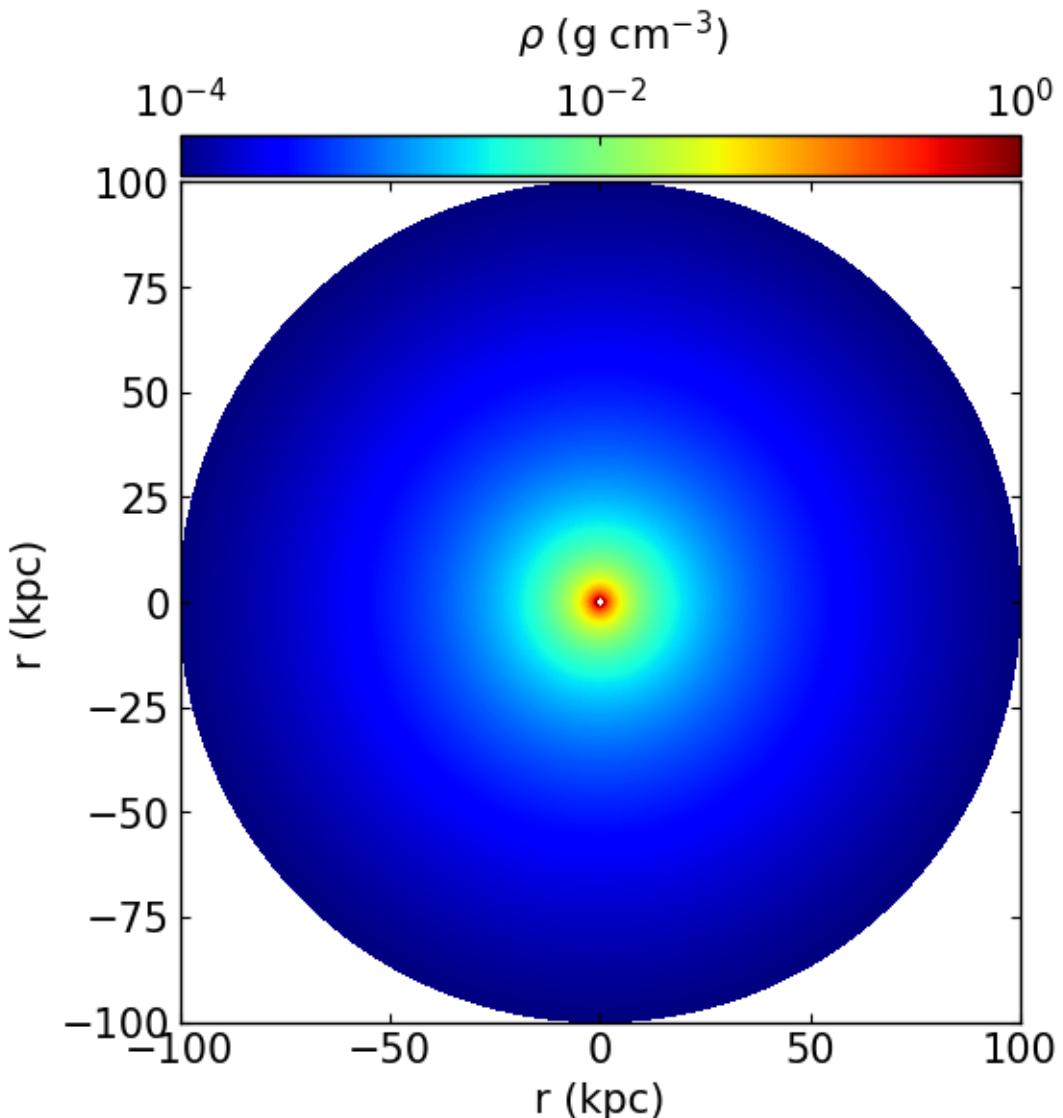
ax1 = fig.add_subplot(111)
ax1.set_aspect('equal')

norm = mpl.colors.LogNorm(vmin=d.min(),vmax=d.max())
p = ax1.pcolormesh(x2d,y2d, d, norm=norm, cmap='jet')
ax1.set_xlabel('r (kpc)')
ax1.set_ylabel('r (kpc)')

# colorbar
divider = make_axes_locatable(ax1)
cax1 = divider.append_axes('top', size='5%', pad=0.03)
clb = fig.colorbar(p,ax=ax1,cax=cax1 \
                    ,orientation='horizontal')
cax1.xaxis.set_ticks_position('top')
cax1.annotate(r'$\rho$ (g cm$^{-3}$)',(0.4,3.),xycoords='axes fraction')

# set the number of ticks
clb.locator = mpl.ticker.LogLocator(numticks=3)
# if the scale is not log, but normal, it should be
# clb.locator = mpl.ticker.MaxNLocator(nbins=3)
clb.update_ticks()

```



In [9]: """ Shared axis & colorbar with multiple plots """

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

%matplotlib inline

Nr, Nth = 500, 500
r = np.linspace(1.,100.,Nr)
th = np.linspace(0.,2.*np.pi,Nth)
```

```

d = np.zeros((Nr,Nth), dtype=float)
for i,ir in enumerate(r):
    d[i,:] = 1./ir**2.

r2d, th2d = np.meshgrid(r,th,indexing='ij')
x2d, y2d = r2d*np.sin(th2d), r2d*np.cos(th2d)

fig = plt.figure(figsize=(12,6))

# first plot
#ax1 = fig.add_subplot(121, axisbg='white')
ax1 = fig.add_subplot(121)
ax1.set_aspect('equal')

norm = mpl.colors.LogNorm(vmin=d.min(),vmax=d.max())
p = ax1.pcolormesh(x2d,y2d, d, norm=norm, cmap='jet')
ax1.set_xlabel('r (kpc)')
ax1.set_ylabel('r (kpc)')

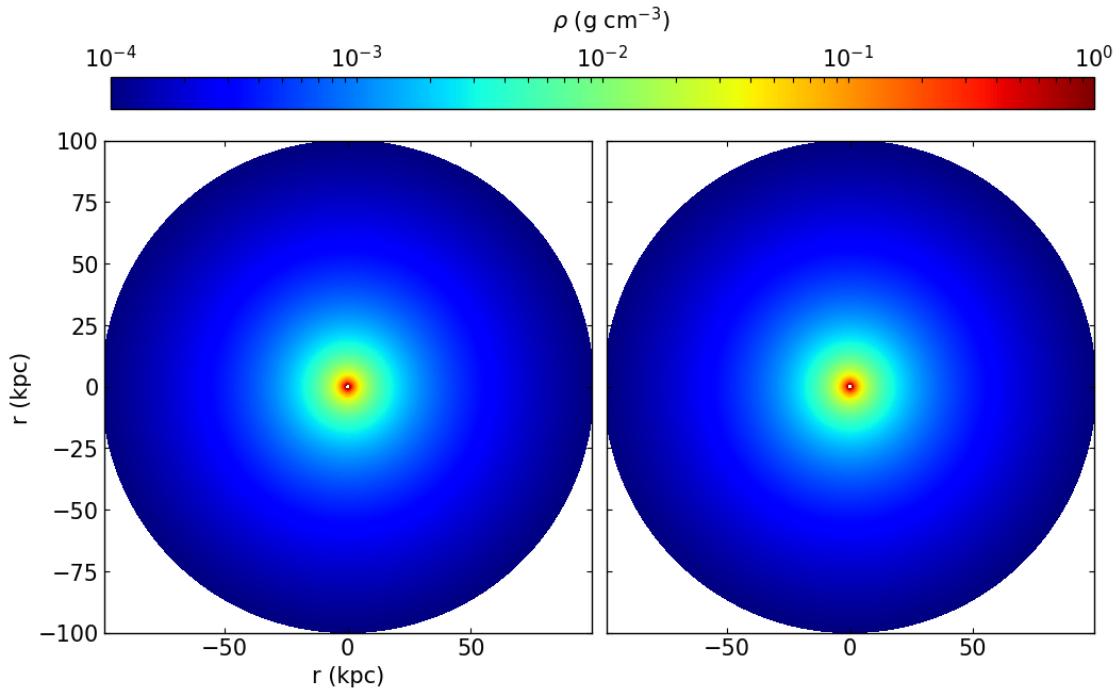
# second plot
#ax2 = fig.add_subplot(122, axisbg='white', sharey=ax1)
ax2 = fig.add_subplot(122,sharey=ax1)
ax2.set_aspect('equal')
p = ax2.pcolormesh(x2d,y2d, d, norm=norm, cmap='jet')
plt.setp(ax2.get_yticklabels(), visible=False)
plt.subplots_adjust(wspace=0.03)

# colorbar
# axis for color bar
ax3 = fig.add_axes([0.13,0.93,0.77,0.05]) # [x,y, x-width, y-width]

norm = mpl.colors.LogNorm(vmin=d.min(), vmax=d.max())
clb = mpl.colorbar.ColorbarBase(ax3,orientation='horizontal' \
                                ,norm=norm,cmap='jet')
ax3.xaxis.set_ticks_position("top")
ax3.annotate(r'$\rho$ (g cm$^{-3}$)',(0.45,2.5),xycoords='axes fraction')

```

Out[9]: <matplotlib.text.Annotation at 0x7f3953fe9450>



In [10]: *""" antialiased Problem (light borders between contour patches) """*

```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

%matplotlib inline

Nr, Nth = 500, 500
r = np.linspace(1.,100.,Nr)
th = np.linspace(0.,2.*np.pi,Nth)

d = np.zeros((Nr,Nth), dtype=float)
for i,ir in enumerate(r):
    d[i,:] = 1./ir**2.

r2d, th2d = np.meshgrid(r,th,indexing='ij')
x2d, y2d = r2d*np.sin(th2d), r2d*np.cos(th2d)

fig = plt.figure(figsize=(12,6))

nlevs=256
vmax=d.max() ; vmin=d.min()
levs = np.power(10.,np.arange(nlevs)/float(nlevs-1)*(np.log10(vmax)-np.log10(vmin)) \
                + np.log10(vmin))

```

```

# first plot
#ax1 = fig.add_subplot(121, axisbg='white')
ax1 = fig.add_subplot(121)
ax1.set_aspect('equal')

norm = mpl.colors.LogNorm(vmin=d.min(), vmax=d.max())
p = ax1.contourf(x2d,y2d, d, norm=norm, cmap='jet', antialiased=False, levels=levs)
ax1.set_xlabel('r (kpc)')
ax1.set_ylabel('r (kpc)')

# second plot
#ax2 = fig.add_subplot(122, axisbg='white', sharey=ax1)
ax2 = fig.add_subplot(122, sharey=ax1)
ax2.set_aspect('equal')

p = ax2.contourf(x2d,y2d, d, norm=norm, cmap='jet', antialiased=True, levels=levs)
plt.setp(ax2.get_yticklabels(), visible=False)
plt.subplots_adjust(wspace=0.03)

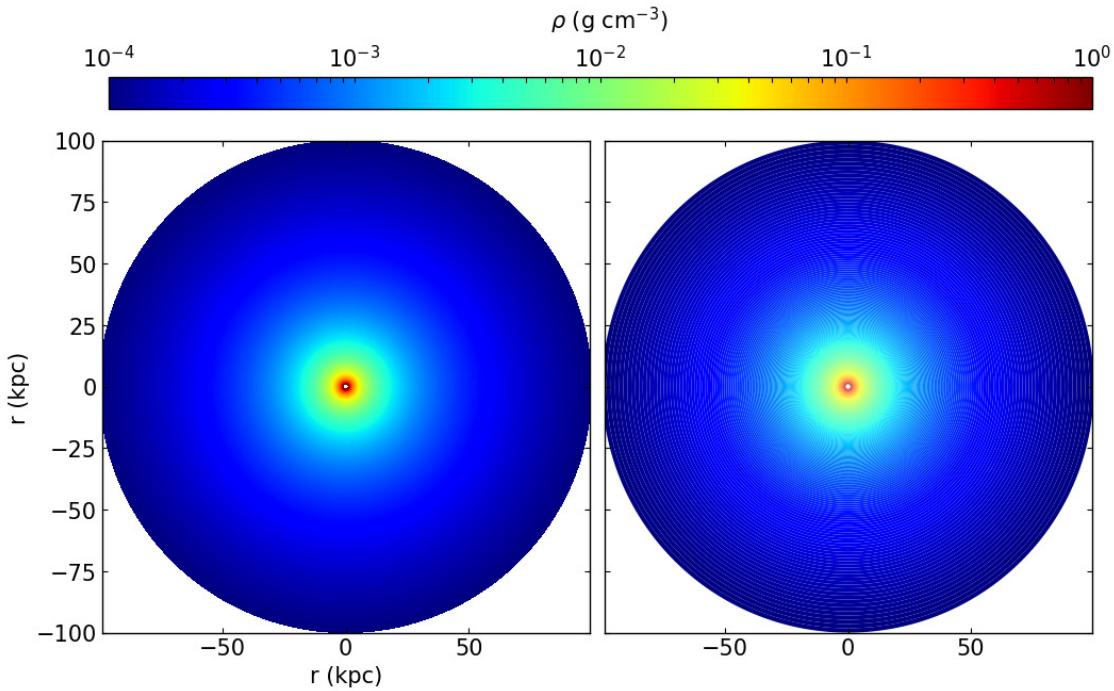
# colorbar

# axis for color bar
ax3 = fig.add_axes([0.13,0.93,0.77,0.05])    #[x, y, x-width, y-width]

norm = mpl.colors.LogNorm(vmin=d.min(), vmax=d.max())
clb = mpl.colorbar.ColorbarBase(ax3,orientation='horizontal' \
                                ,norm=norm,cmap='jet')
ax3.xaxis.set_ticks_position("top")
ax3.annotate(r'$\rho$ (g cm$^{-3}$)',(0.45,2.5),xycoords='axes fraction')

```

Out[10]: <matplotlib.text.Annotation at 0x7f3953ffb310>



In [11]: """ antialiased Problem (light borders between contour patches) II
If we save the figure as postscript file (.eps), the light borders are still shown even without antialiased option.
Solution -> edgecolor of the contour should be set to 'face' """

```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline

Nr, Nth = 500, 500
r = np.linspace(1.,100.,Nr)
th = np.linspace(0.,2.*np.pi,Nth)

d = np.zeros((Nr,Nth), dtype=float)
for i,ir in enumerate(r):
    d[i,:] = 1./ir**2.

r2d, th2d = np.meshgrid(r,th,indexing='ij')
x2d, y2d = r2d*np.sin(th2d), r2d*np.cos(th2d)

fig = plt.figure(figsize=(12,6))

nlevs=256
vmax=d.max() ; vmin=d.min()

```

```

levs = np.power(10.,np.arange(nlevs)/float(nlevs-1)*(np.log10(vmax)-np.log10(vmin)) \
                + np.log10(vmin))

# first plot
#ax1 = fig.add_subplot(121, axisbg='white')
ax1 = fig.add_subplot(121)
ax1.set_aspect('equal')

# reverse of the color_map
norm = mpl.colors.LogNorm(vmin=d.min(),vmax=d.max())
p = ax1.contourf(x2d,y2d, d, norm=norm, cmap='jet_r', antialiased=False, levels=levs)
ax1.set_xlabel('r (kpc)')
ax1.set_ylabel('r (kpc)')

""" Important Part """
# To erase the antialiased line between counter patches
for c in p.collections:
    c.set_edgecolor("face")

# second plot
#ax2 = fig.add_subplot(122, axisbg='white', sharey=ax1)
ax2 = fig.add_subplot(122, sharey=ax1)
ax2.set_aspect('equal')

p = ax2.contourf(x2d,y2d, d, norm=norm, cmap='jet', antialiased=True, levels=levs)
plt.setp(ax2.get_yticklabels(), visible=False)
plt.subplots_adjust(wspace=0.03)

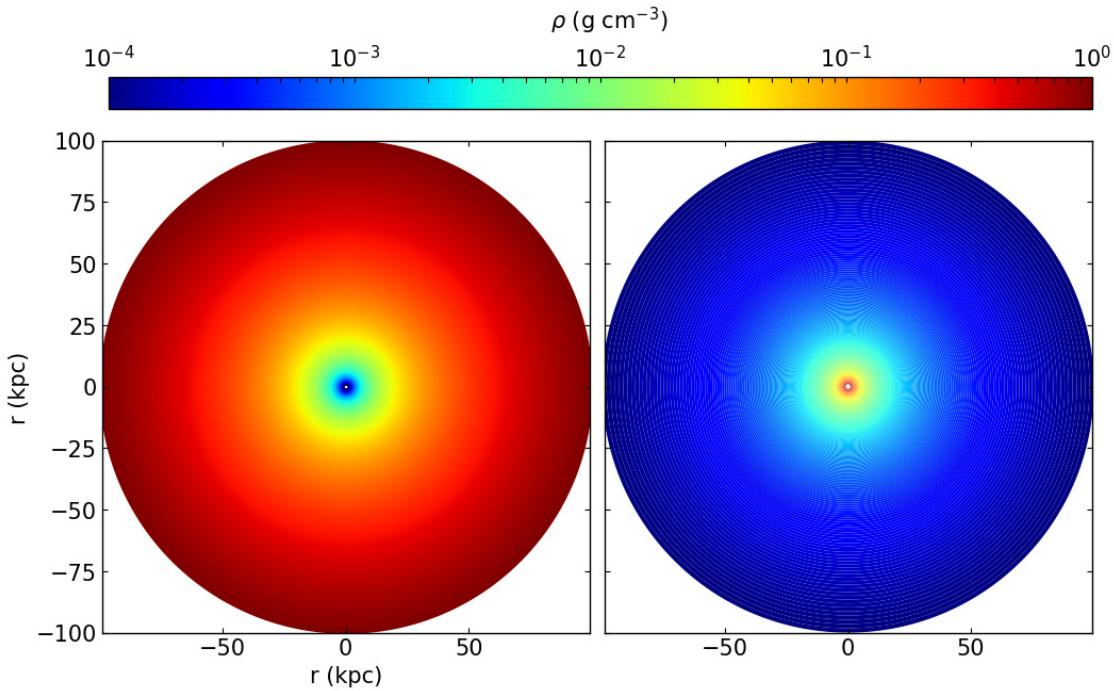
# colorbar

# axis for color bar
ax3 = fig.add_axes([0.13,0.93,0.77,0.05]) #[x,y, x-width, y-width]

norm = mpl.colors.LogNorm(vmin=d.min(), vmax=d.max())
clb = mpl.colorbar.ColorbarBase(ax3,orientation='horizontal' \
                                ,norm=norm,cmap='jet')
ax3.xaxis.set_ticks_position("top")
ax3.annotate(r'$\rho$ (g cm$^{-3}$)',(0.45,2.5),xycoords='axes fraction')

fig.savefig('../figures/test.eps')

```



In [12]: *""" Contour non-uniform grid data """*

```

from numpy.random import uniform, seed
from matplotlib.mlab import griddata
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline

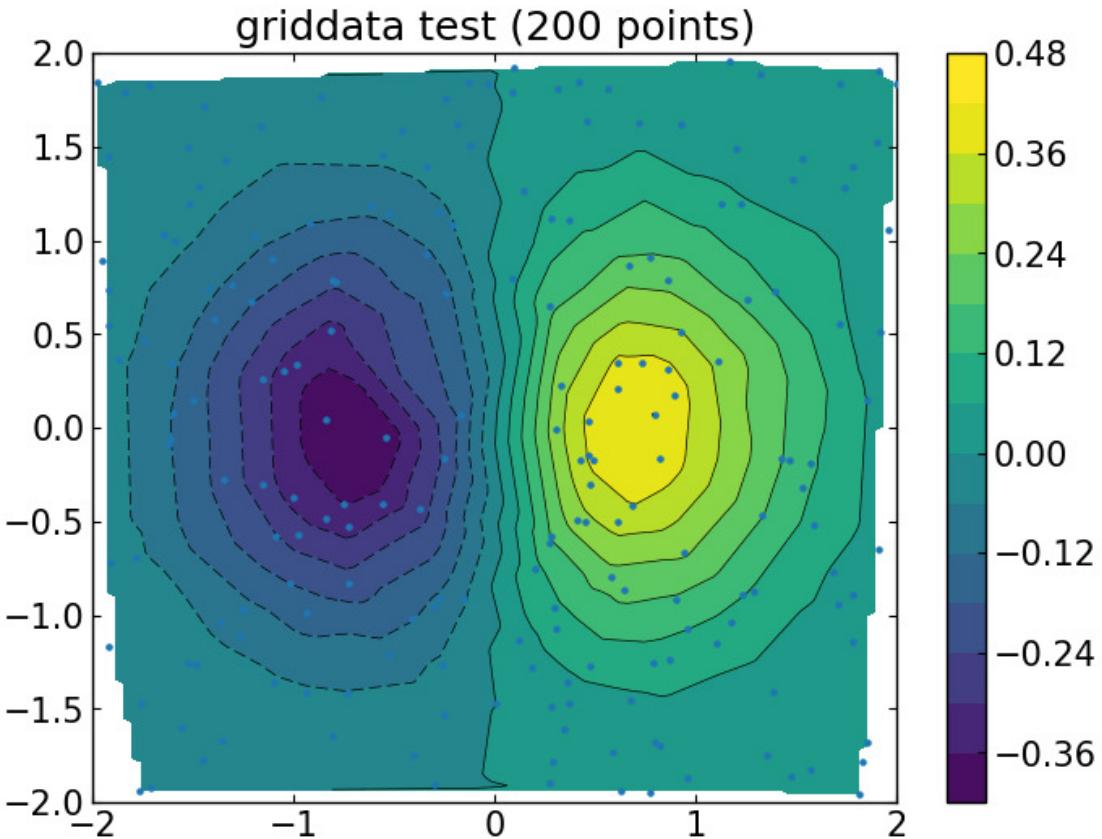
# make up data.
npts = int(raw_input('enter # of random points to plot:'))
seed(0)
npts = 200
x = uniform(-2, 2, npts)
y = uniform(-2, 2, npts)
z = x*np.exp(-x**2 - y**2)
# define grid.
xi = np.linspace(-2.1, 2.1, 100)
yi = np.linspace(-2.1, 2.1, 200)
# grid the data.
zi = griddata(x, y, z, xi, yi, interp='linear')
# contour the gridded data, plotting dots at the nonuniform data points.
CS = plt.contour(xi, yi, zi, 15, linewidths=0.5, colors='k')
CS = plt.contourf(xi, yi, zi, 15,
                  vmax=abs(zi).max(), vmin=-abs(zi).max())

```

```

plt.colorbar() # draw colorbar
# plot data points.
plt.scatter(x, y, marker='o', s=5, zorder=10)
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.title('griddata test (%d points)' % npts)
plt.show()

```



0.3 Draw vertical & horizontal lines in the figure

- Note that the position in axhline or axvline indicate the relative one for the window size. --> e.g.) xmin, xmax indicates the relative position for the window size
- To draw the vertical/horizontal lines with the portion of the window with data coordinate, it should be --> plt.plot((x1,x2),(y1,y2))

```

In [13]: import numpy as np
        import matplotlib.pyplot as plt
%matplotlib inline

t = np.arange(-1, 2, .01)

```

```

s = np.sin(2*np.pi*t)

plt.plot(t, s)
# draw a thick red hline at y=0 that spans the xrange
l = plt.axhline(linewidth=8, color='#d62728')

# draw a default hline at y=1 that spans the xrange
l = plt.axhline(y=1)

# draw a default vline at x=1 that spans the yrange
l = plt.axvline(x=1)

# draw a thick blue vline at x=0 that spans the upper quadrant of
# the yrange
l = plt.axvline(x=0, ymin=0.75, linewidth=8, color='#1f77b4')

# draw a default hline at y=.5 that spans the middle half of
# the axes
l = plt.axhline(y=.5, xmin=0.25, xmax=0.75)

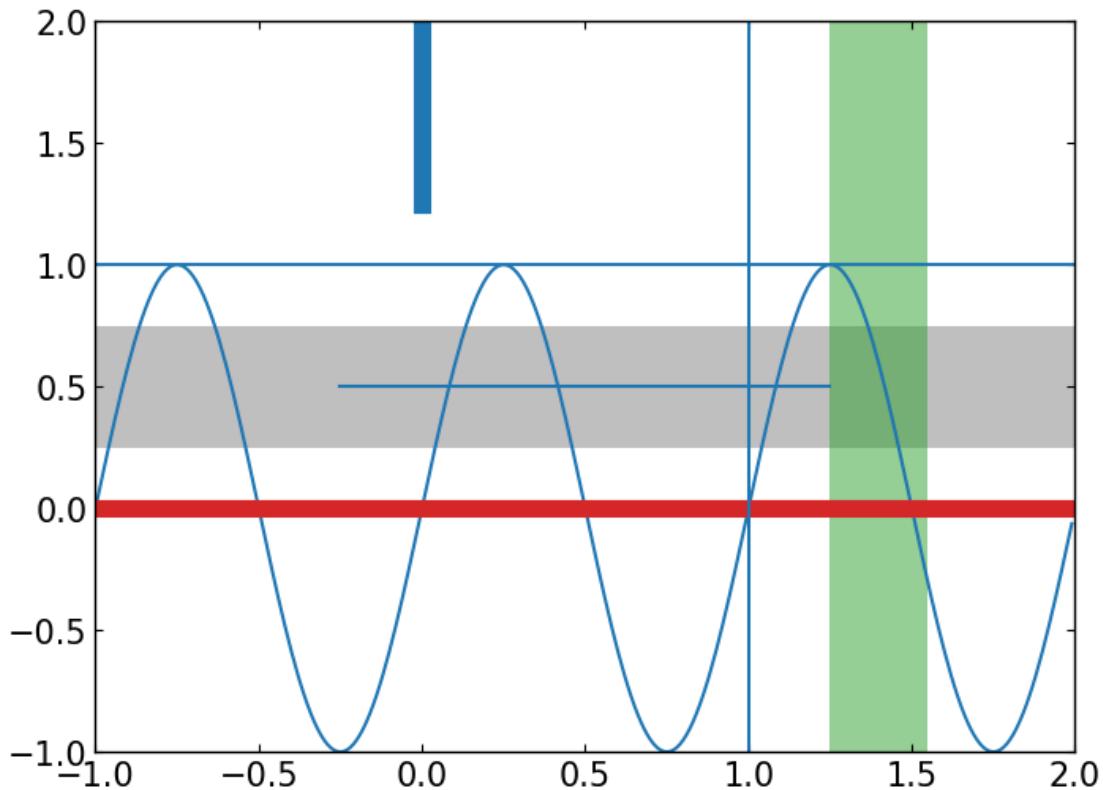
p = plt.axhspan(0.25, 0.75, facecolor='0.5', alpha=0.5)

p = plt.axvspan(1.25, 1.55, facecolor='#2ca02c', alpha=0.5)

plt.axis([-1, 2, -1, 2])

```

Out[13]: [-1, 2, -1, 2]



0.3.1 shared x & y label on multiple plots; managing multiple plots manually

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np

%matplotlib inline

xlim = [0,100]
ylim = [0,90000]

x = np.linspace(0,100,5000)
y1 = x**2/2.
y2 = x**2
y3 = x**2*5
y4 = x**2*10

# subplots configuration (position should be converted to normal scale)
pltx0=[30.,10.]; plty0=[20.,10.]    # margin left & right
pltxw=0.; pltyw=3.                     # x(horizontal), y(vertical) gap
pltxs=100.; pltys=100.                 # plot size
winxs=np.sum(pltx0)+2*pltxs+pltxw; winys=np.sum(plty0)+2*pltys+pltyw
```

```

fig = plt.figure(figsize=(9,9*winys/winx))

bbox_props = dict(boxstyle="round", fc="w", ec="0.5", alpha=0.9)

# first plot
ax1 = fig.add_subplot(221)
ax1.plot(x,y1)
ax1.set_xlim(xlim[0], xlim[1])
ax1.set_ylim(ylim[0], ylim[1])
ax1.text(10, 80000, "a", ha="center", va="center", size=20,
         bbox=bbox_props)
plt.setp(ax1.get_xticklabels(), visible=False)

# second plot
ax2 = fig.add_subplot(222, sharey=ax1)
ax2.plot(x,y2)
ax2.set_xlim(xlim[0], xlim[1])
ax2.set_ylim(ylim[0], ylim[1])
ax2.text(10, 80000, "b", ha="center", va="center", size=20,
         bbox=bbox_props)

plt.setp(ax2.get_xticklabels(), visible=False)
plt.setp(ax2.get_yticklabels(), visible=False)
#plt.subplots_adjust(wspace=plttxw/pltxs)

# third plot
ax3 = fig.add_subplot(223, sharex=ax1)
ax3.plot(x,y3)
ax3.set_xlim(xlim[0], xlim[1])
ax3.set_ylim(ylim[0], ylim[1])
ax3.text(10, 80000, "c", ha="center", va="center", size=20,
         bbox=bbox_props)

#plt.subplots_adjust(hspace=plttyw/pltys)

# forth plot
ax4 = fig.add_subplot(224, sharex=ax2, sharey=ax3)
ax4.plot(x,y4)
ax4.set_xlim(xlim[0], xlim[1])
ax4.set_ylim(ylim[0], ylim[1])
ax4.text(10, 80000, "d", ha="center", va="center", size=20,
         bbox=bbox_props)

plt.setp(ax4.get_yticklabels(), visible=False)
#plt.subplots_adjust(hspace=plttyw/pltys, wspace=plttxw/pltxs)

# set the figure label
plt.annotate('yaxis label',(0.01,0.6),xycoords='figure fraction',rotation='vertical')

```

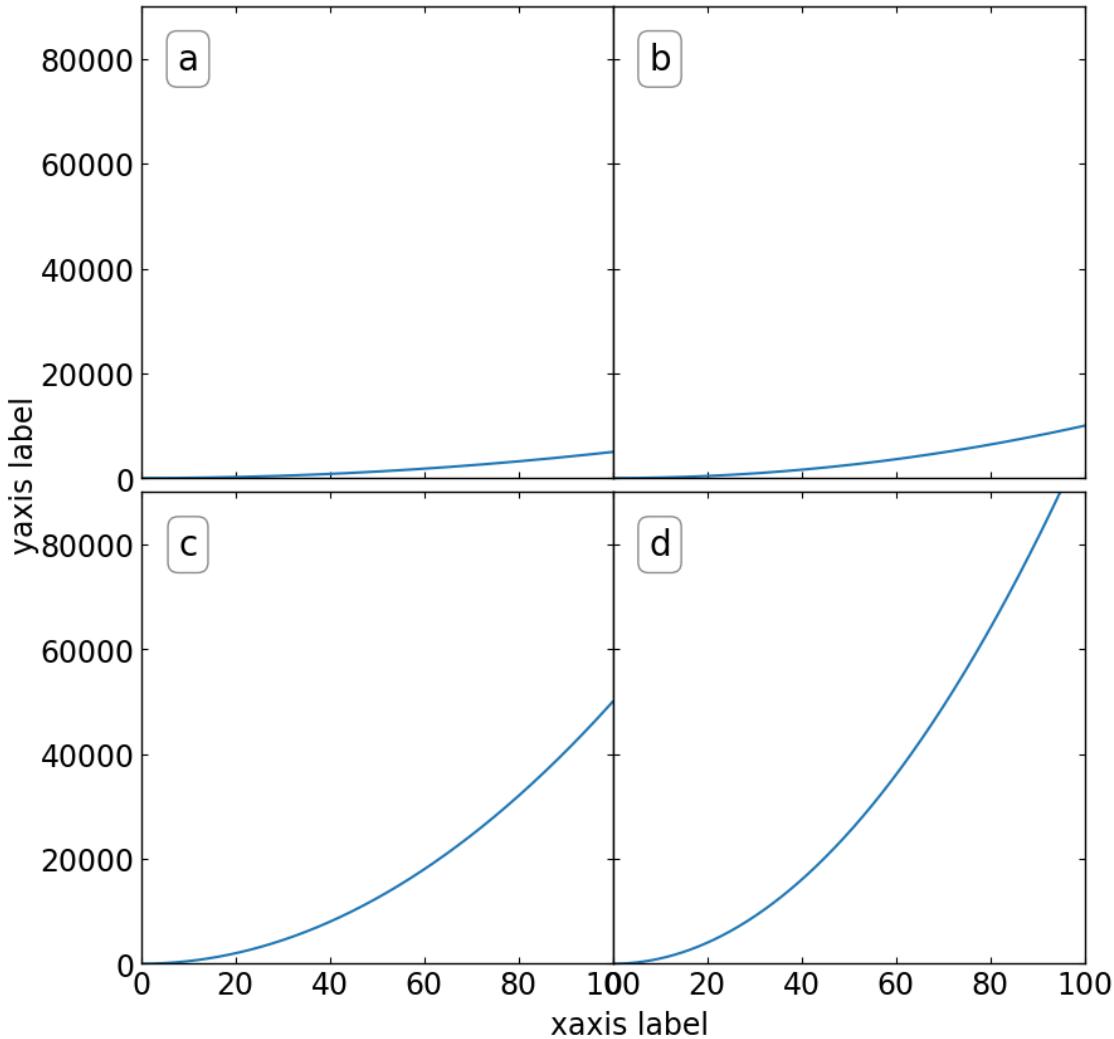
```

plt.annotate('xaxis label',(0.49,0.02),xycoords='figure fraction')

# adjust margin
fig.subplots_adjust(left=pltx0[0]/winxs,right=1-pltx0[1]/winxs,bottom=plty0[0]/winys,
                    hspace=pltyw/pltys, wspace=pltxw/pltxs)

# save the figure
#fig.savefig('test.eps')

```



In [3]: """ Another way of drawing multiple plots """

```

import matplotlib.pyplot as plt
import numpy as np

```

```

%matplotlib inline

xlim = [0,100]
ylim = [0,90000]

x = np.linspace(0,100,5000)
y1 = x**2/2.
y2 = x**2
y3 = x**2*5
y4 = x**2*10

# subplots configuration (position should be converted to normal scale)
pltx0=[30.,10.]; plty0=[20.,10.]    # margin left & right
pltxw=0.; pltyw=3.                  # x(horizontal), y(vertical) gap
pltxs1=100.; pltxs2=50.
pltys=100.                         # plot size
winxs=np.sum(pltx0)+pltxs1+pltxs2+pltxw; winys=np.sum(plty0)+2*pltys+pltyw

bbox_props = dict(boxstyle="round", fc="w", ec="0.5", alpha=0.9)

# definition for the axes
left1, width1 = pltx0[0]/winxs, pltxs1/winxs
left2, width2 = (pltx0[0]+pltxs1+pltxw)/winxs, pltxs2/winxs
bottom1, height1 = (plty0[0]+pltys+pltyw)/winys, pltys/winys
bottom2, height2 = plty0[0]/winys, pltys/winys

# axes positions
[ 1,1  2,1
 1,2  2,2 ]

plt.figure(figsize=(9,9*winys/winxs))

# first plot
ax1 = plt.axes([left1,bottom1,width1,height1])
ax1.plot(x,y1)
ax1.set_xlim(xlim[0], xlim[1])
ax1.set_ylim(ylim[0], ylim[1])
ax1.text(10, 80000, "a", ha="center", va="center", size=20,
         bbox=bbox_props)
plt.setp(ax1.get_xticklabels(), visible=False)

# second plot
ax2 = plt.axes([left2,bottom1,width2,height1])
ax2.plot(x,y2)
ax2.set_xlim(xlim[0], xlim[1])
ax2.set_ylim(ylim[0], ylim[1])
ax2.text(10, 80000, "b", ha="center", va="center", size=20,
         bbox=bbox_props)

```

```

plt.setp(ax2.get_xticklabels(), visible=False)
plt.setp(ax2.get_yticklabels(), visible=False)
# plt.subplots_adjust(wspace=pltxw/pltxs)

# third plot
ax3 = plt.axes([left1, bottom2, width1, height2])
ax3.plot(x, y3)
ax3.set_xlim(xlim[0], xlim[1])
ax3.set_ylim(ylim[0], ylim[1])
ax3.text(10, 80000, "c", ha="center", va="center", size=20,
         bbox=bbox_props)

# plt.subplots_adjust(hspace=pltwyw/pltys)

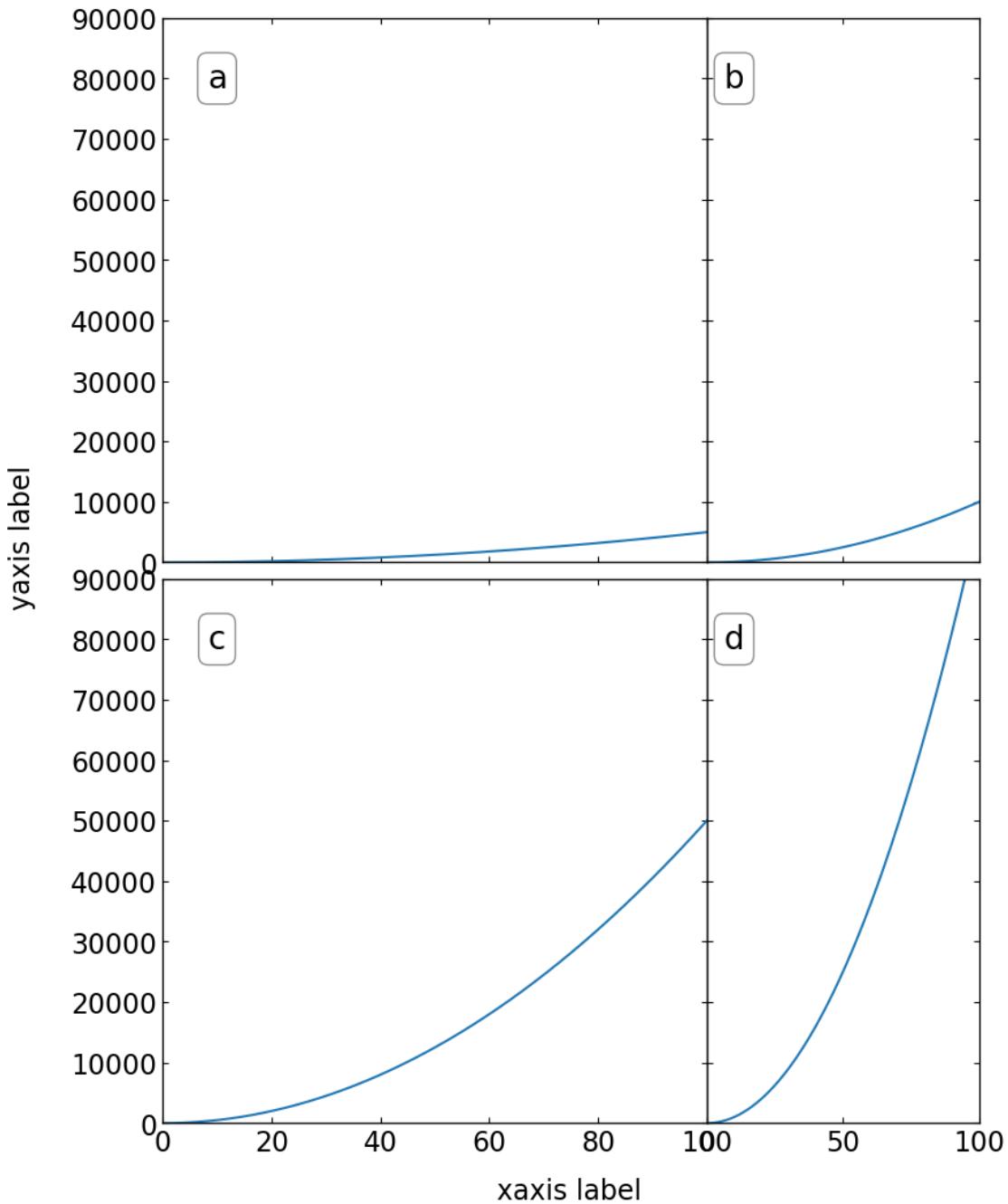
# forth plot
ax4 = plt.axes([left2, bottom2, width2, height2])
ax4.plot(x, y4)
ax4.set_xlim(xlim[0], xlim[1])
ax4.set_ylim(ylim[0], ylim[1])
ax4.text(10, 80000, "d", ha="center", va="center", size=20,
         bbox=bbox_props)

plt.setp(ax4.get_yticklabels(), visible=False)
# plt.subplots_adjust(hspace=pltwyw/pltys, wspace=pltxw/pltxs)

# set the figure label
plt.annotate('yaxis label', (0.01, 0.6), xycoords='figure fraction', rotation='vertical')
plt.annotate('xaxis label', (0.49, 0.02), xycoords='figure fraction')

```

Out[3]: <matplotlib.text.Annotation at 0x7ff90f592d50>



0.3.2 Set the drawing order

In [15]: *""" Drawing order (zorder) """*

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

```

x = np.linspace(0.,100.,100)
y1 = x*2.
y2 = x*2.

fig = plt.figure()
ax = fig.add_subplot(111)

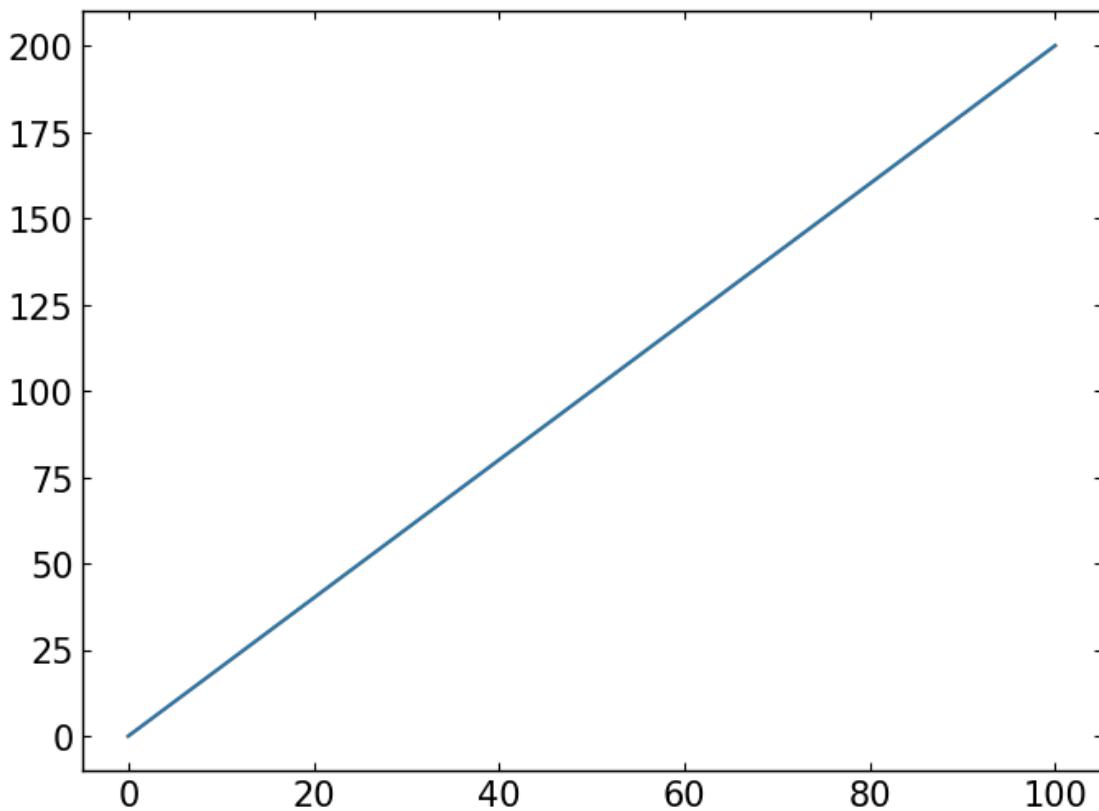
line1, = ax.plot(x,y1,color='C0',zorder=2)
line2, = ax.plot(x,y2,color='C1',zorder=1)

#line1, line2 = ax.plot(x,y1,x,y2)

line2
line1

```

Out[15]: <matplotlib.lines.Line2D at 0x7f395147ce10>



In [16]: """

The default drawing order for axes is patches, lines, text. This

order is determined by the zorder attribute. The following defaults are set

<i>Artist</i>	<i>Z-order</i>
<i>Patch / PatchCollection</i>	1
<i>Line2D / LineCollection</i>	2
<i>Text</i>	3

You can change the order for individual artists by setting the zorder. Any individual plot() call can set a value for the zorder of that particular item.

In the first subplot below, the lines are drawn above the patch collection from the scatter, which is the default.

In the subplot below, the order is reversed.

The second figure shows how to control the zorder of individual lines.
"""

```
import matplotlib.pyplot as plt
import numpy as np

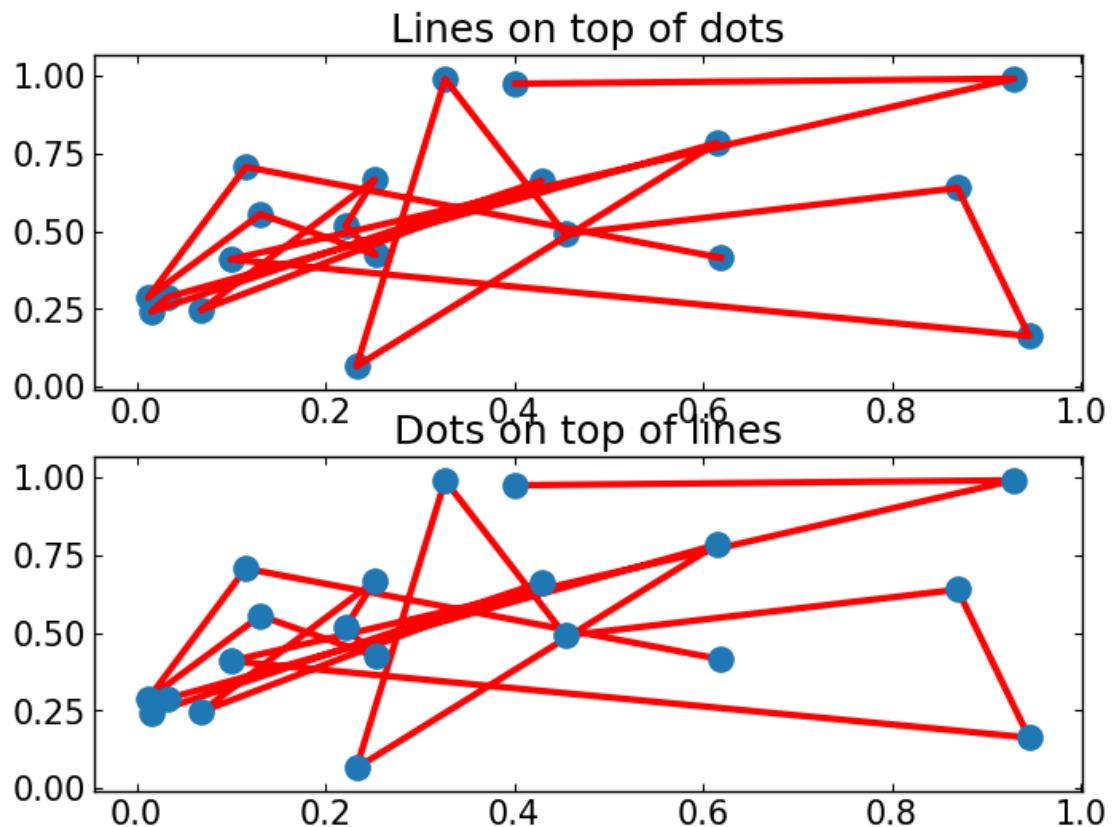
x = np.random.random(20)
y = np.random.random(20)

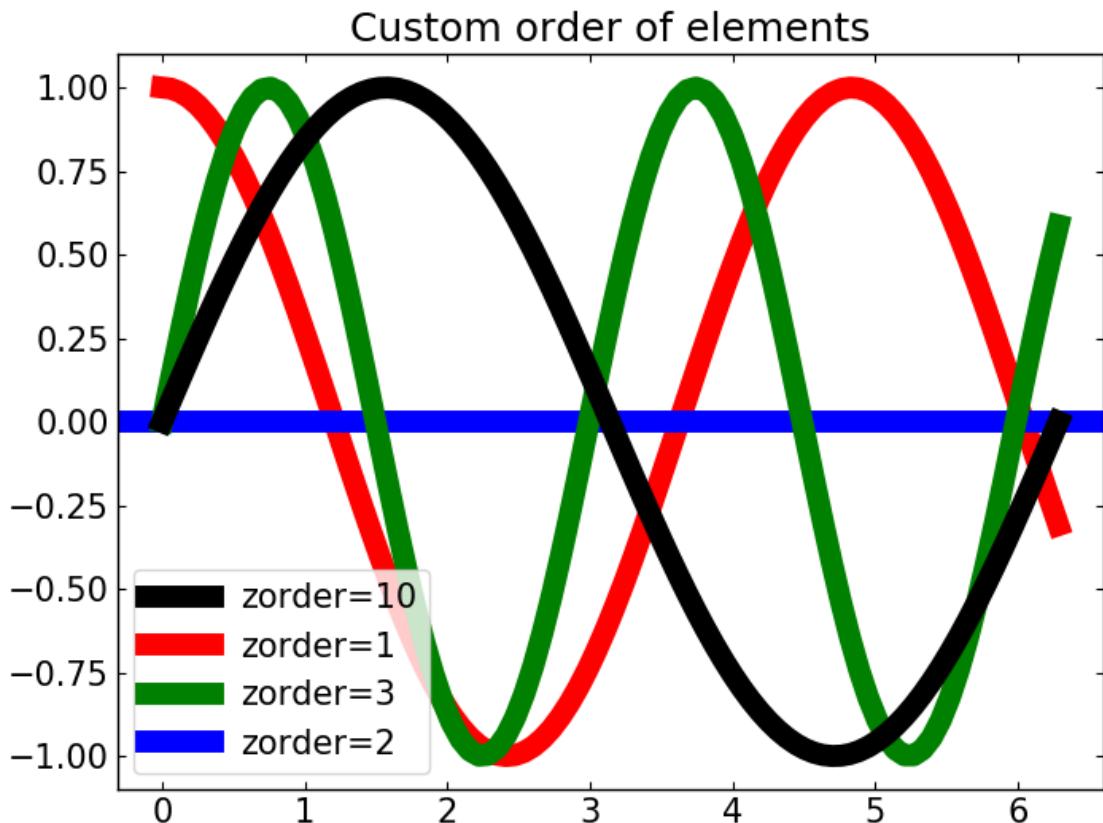
# Lines on top of scatter
plt.figure()
plt.subplot(211)
plt.plot(x, y, 'r', lw=3)
plt.scatter(x, y, s=120)
plt.title('Lines on top of dots')

# Scatter plot on top of lines
plt.subplot(212)
plt.plot(x, y, 'r', zorder=1, lw=3)
plt.scatter(x, y, s=120, zorder=2)
plt.title('Dots on top of lines')

# A new figure, with individually ordered items
x = np.linspace(0, 2*np.pi, 100)
plt.figure()
plt.plot(x, np.sin(x), linewidth=10, color='black', label='zorder=10', zorder=10) #
plt.plot(x, np.cos(1.3*x), linewidth=10, color='red', label='zorder=1', zorder=1) #
plt.plot(x, np.sin(2.1*x), linewidth=10, color='green', label='zorder=3', zorder=3)
plt.axhline(0, linewidth=10, color='blue', label='zorder=2', zorder=2)
plt.title('Custom order of elements')
l = plt.legend()
l.set_zorder(20) # put the legend on top
```

```
plt.show()
```





0.4 Histogram

In [31]: `"""`

```
=====
Demo of the histogram function's different ``histtype`` settings
=====
```

- * Histogram with step curve that has a color fill.*
- * Histogram with custom and unequal bin widths.*

Selecting different bin counts and sizes can significantly affect the shape of a histogram. The Astropy docs have a great section on how to select these parameters:

<http://docs.astropy.org/en/stable/visualization/histogram.html>
`"""`

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(0)
```

```

mu = 200
sigma = 25
x = np.random.normal(mu, sigma, size=100)

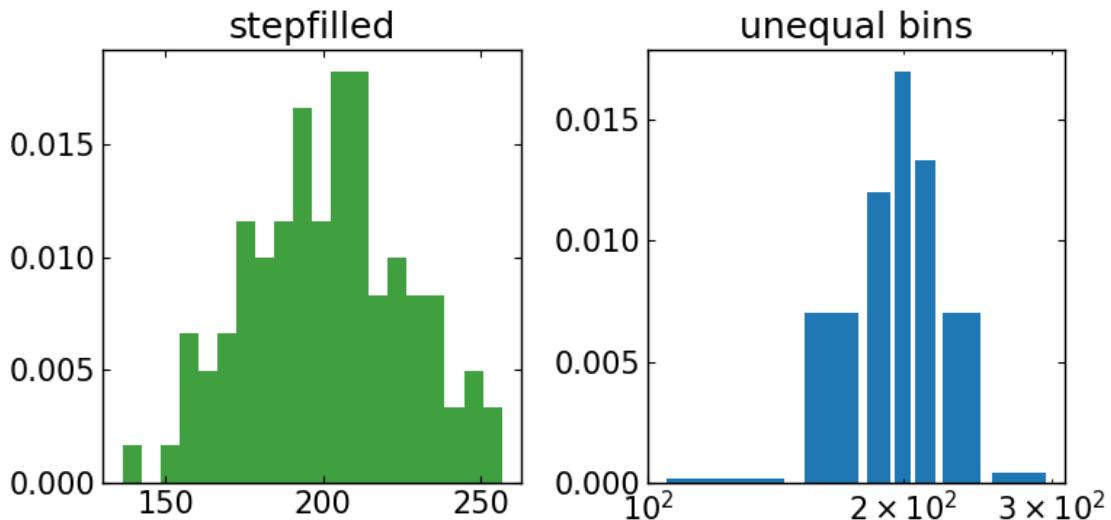
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(8, 4))

ax0.hist(x, 20, normed=1, histtype='stepfilled', facecolor='g', alpha=0.75)
ax0.set_title('stepfilled')

# Create a histogram by providing the bin edges (unequally spaced).
bins = [100, 150, 180, 195, 205, 220, 250, 300]
ax1.hist(x, bins, normed=1, histtype='bar', rwidth=0.8)
ax1.set_title('unequal bins')
plt.xscale('log')

fig.tight_layout()
plt.show()

```



0.5 Dual axes

In [32]: *"""*

```
=====
Plots with different scales
=====
```

Demonstrate how to do two plots on the same axes with different left and right scales.

*The trick is to use *two different axes* that share the same *x* axis. You can use separate `matplotlib.ticker` formatters and locators as desired since the two axes are independent.*

*Such axes are generated by calling the `Axes.twinx` method. Likewise, `Axes.twiny` is available to generate axes that share a *y* axis but have different top and bottom scales.*

The twinx and twiny methods are also exposed as pyplot functions.

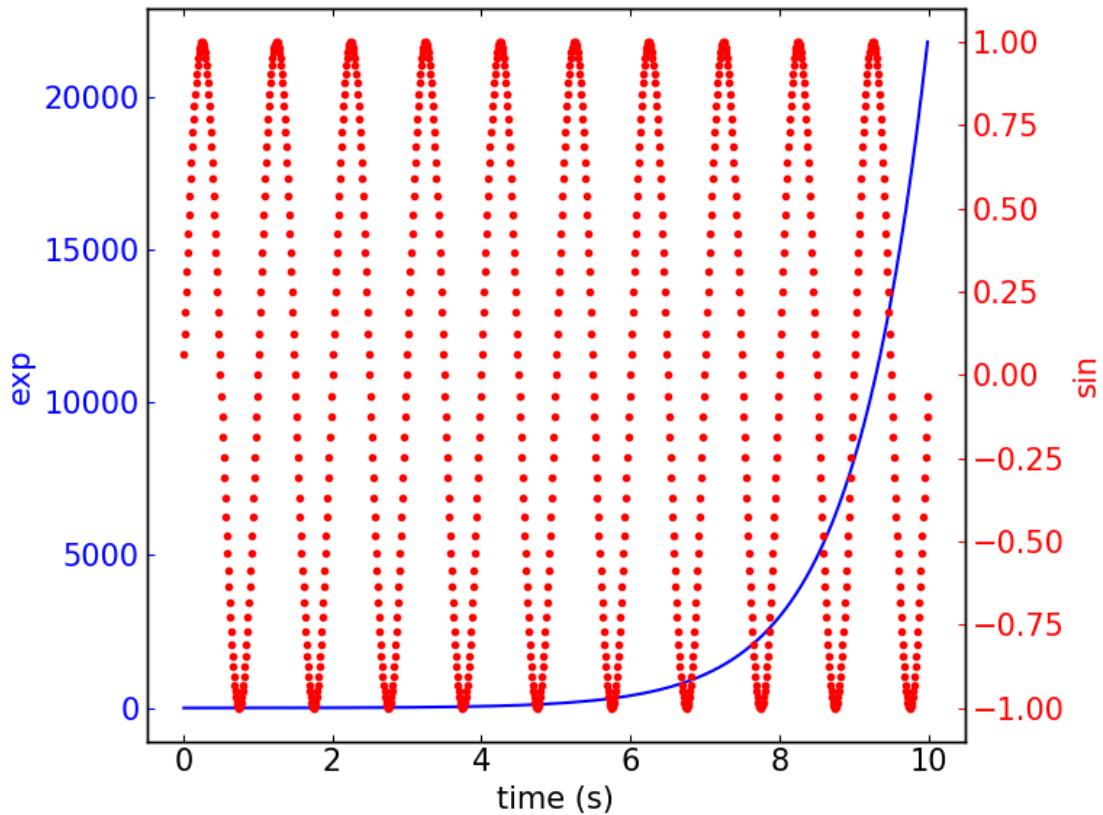
```
"""
```

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax1 = plt.subplots()
t = np.arange(0.01, 10.0, 0.01)
s1 = np.exp(t)
ax1.plot(t, s1, 'b-')
ax1.set_xlabel('time (s)')
# Make the y-axis label, ticks and tick labels match the line color.
ax1.set_ylabel('exp', color='b')
ax1.tick_params('y', colors='b')

ax2 = ax1.twinx()
s2 = np.sin(2 * np.pi * t)
ax2.plot(t, s2, 'r.')
ax2.set_ylabel('sin', color='r')
ax2.tick_params('y', colors='r')

fig.tight_layout()
plt.show()
```



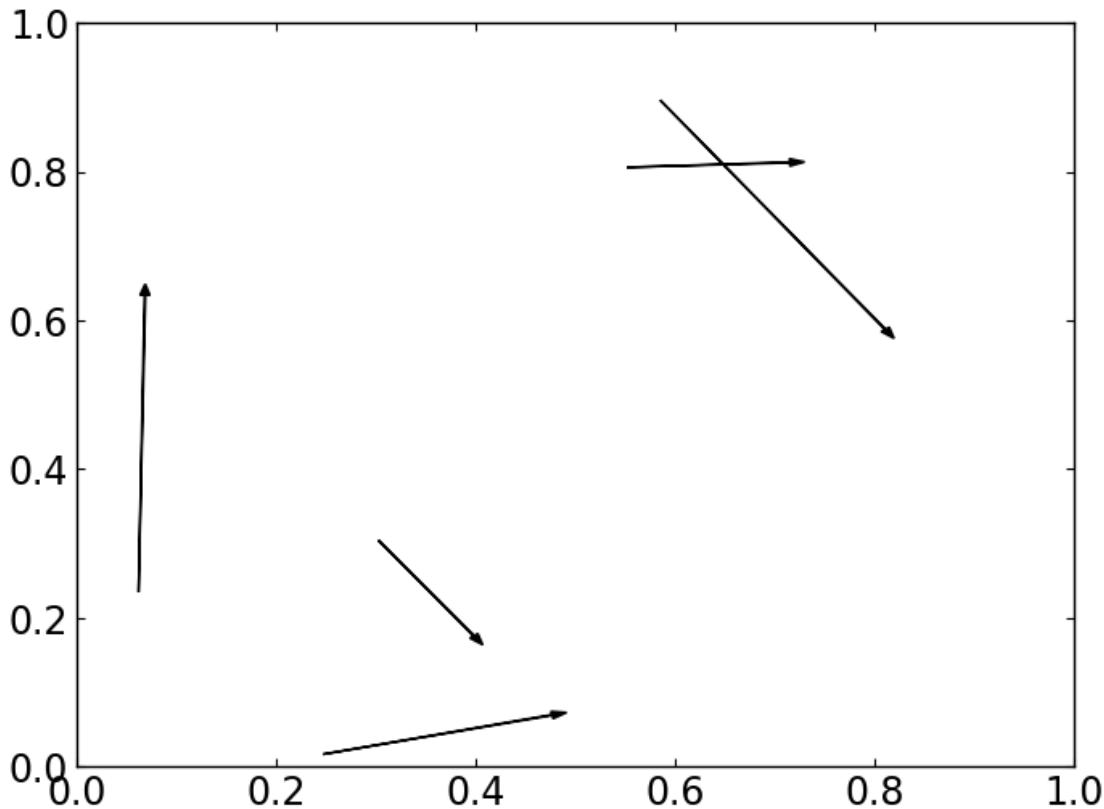
1 Arrows

```
In [27]: import numpy as np
        import matplotlib.pyplot as plt

%matplotlib inline

DATA = np.random.rand(5,5)

for idx in range(0,len(DATA[:,1])):
    colorVal = scalarMap.to_rgba(DATA[idx,4])
    plt.arrow(DATA[idx,0],  #x1
              DATA[idx,1],  #y1
              DATA[idx,2]-DATA[idx,0], #x2 - x1
              DATA[idx,3]-DATA[idx,1], #y2 - y1
              head_width=0.01,facecolor='k')
```

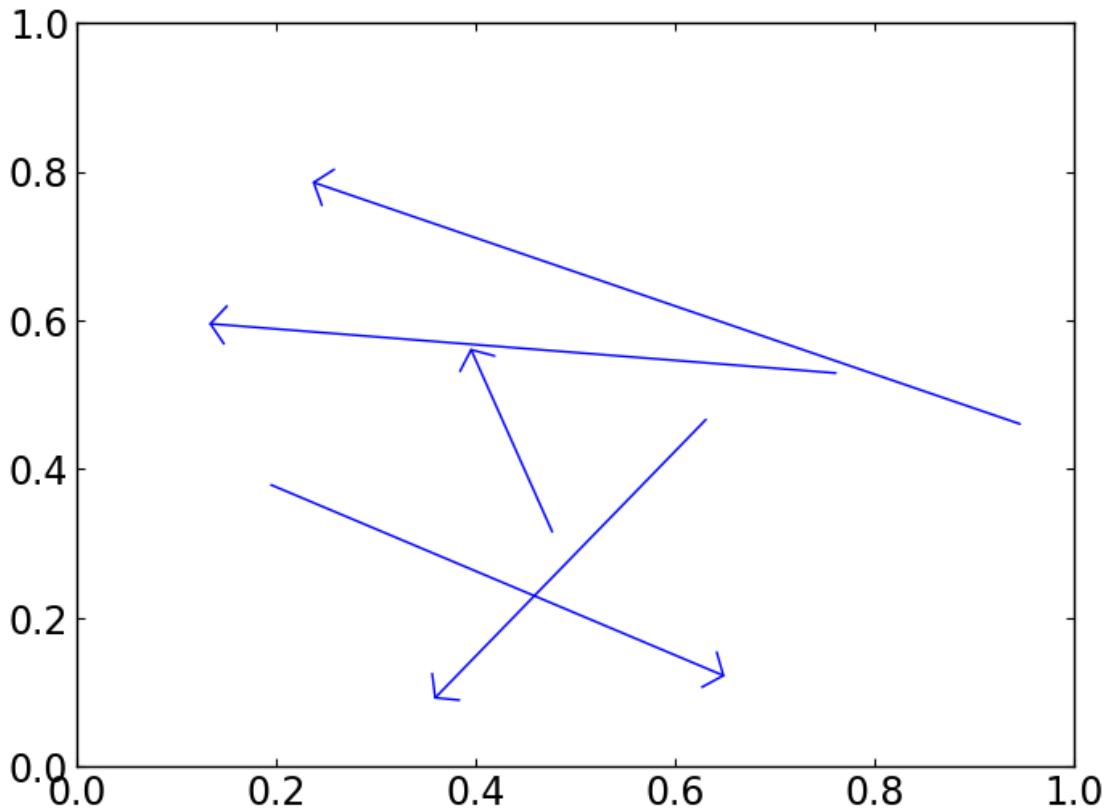


```
In [37]: import numpy as np
        import matplotlib.pyplot as plt

%matplotlib inline

DATA = np.random.rand(5,5)

for idx in range(0,len(DATA[:,1])):
    colorVal = scalarMap.to_rgba(DATA[idx,4])
    plt.annotate("", [DATA[idx,0], DATA[idx,1]],
                [DATA[idx,2], DATA[idx,3]],
                arrowprops=dict(arrowstyle='->',head_width=0.5,edgecolor='blue'))
```



```
In [12]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.colors as colors
        import matplotlib.cm as cmx
        import matplotlib as mpl

DATA = np.random.rand(5,5)

cmap = plt.cm.jet

cNorm  = colors.Normalize(vmin=np.min(DATA[:,4]), vmax=np.max(DATA[:,4]))

scalarMap = cmx.ScalarMappable(norm=cNorm,cmap=cmap)

fig = plt.figure()
ax  = fig.add_axes([0.1, 0.1, 0.7, 0.85]) # [left, bottom, width, height]
axc = fig.add_axes([0.85, 0.10, 0.05, 0.85])

for idx in range(0,len(DATA[:,1])):
    colorVal = scalarMap.to_rgba(DATA[idx,4])
    ax.arrow(DATA[idx,0],  # x1
```

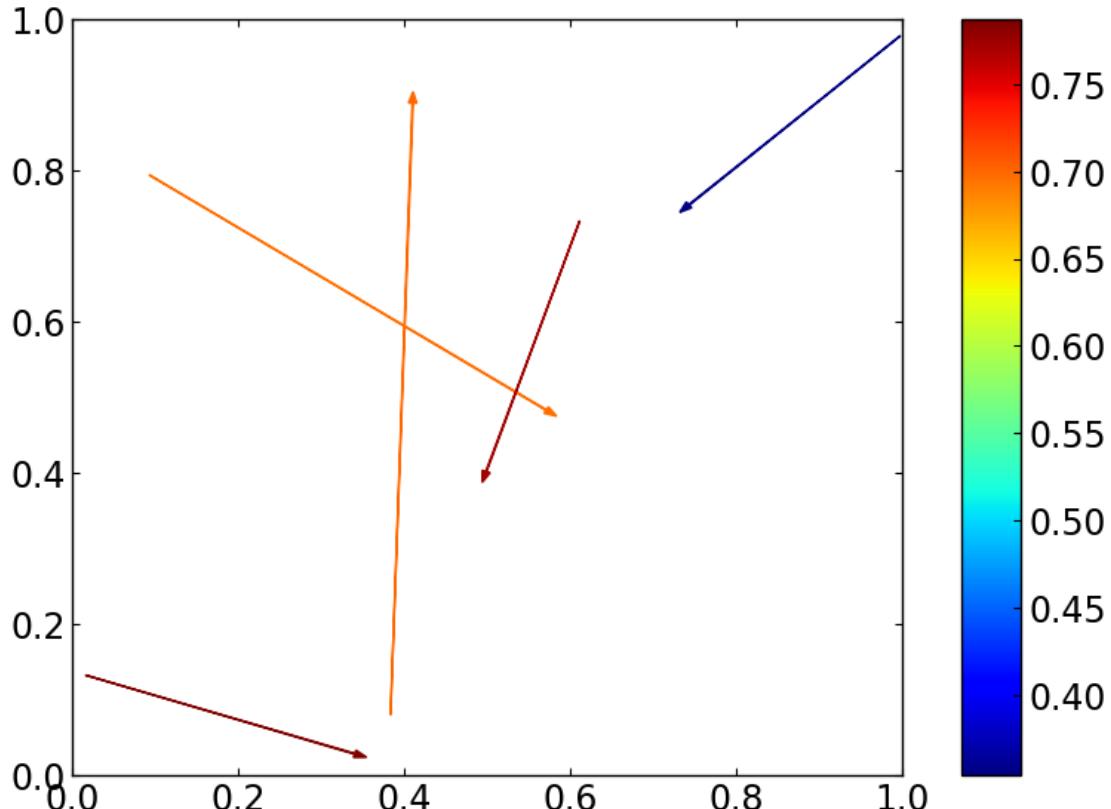
```

        DATA[idx,1], # y1
        DATA[idx,2]-DATA[idx,0], # x2 - x1
        DATA[idx,3]-DATA[idx,1], # y2 - y1
        color=colorVal, head_width=0.01)

cb1 = mpl.colorbar.ColorbarBase(axc, cmap=cmap,
                                norm=cNorm, orientation='vertical')

plt.show()

```



```

In [8]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import matplotlib.cm as cmx

%matplotlib inline

DATA = np.random.rand(5,5)

cmap = plt.cm.jet

```

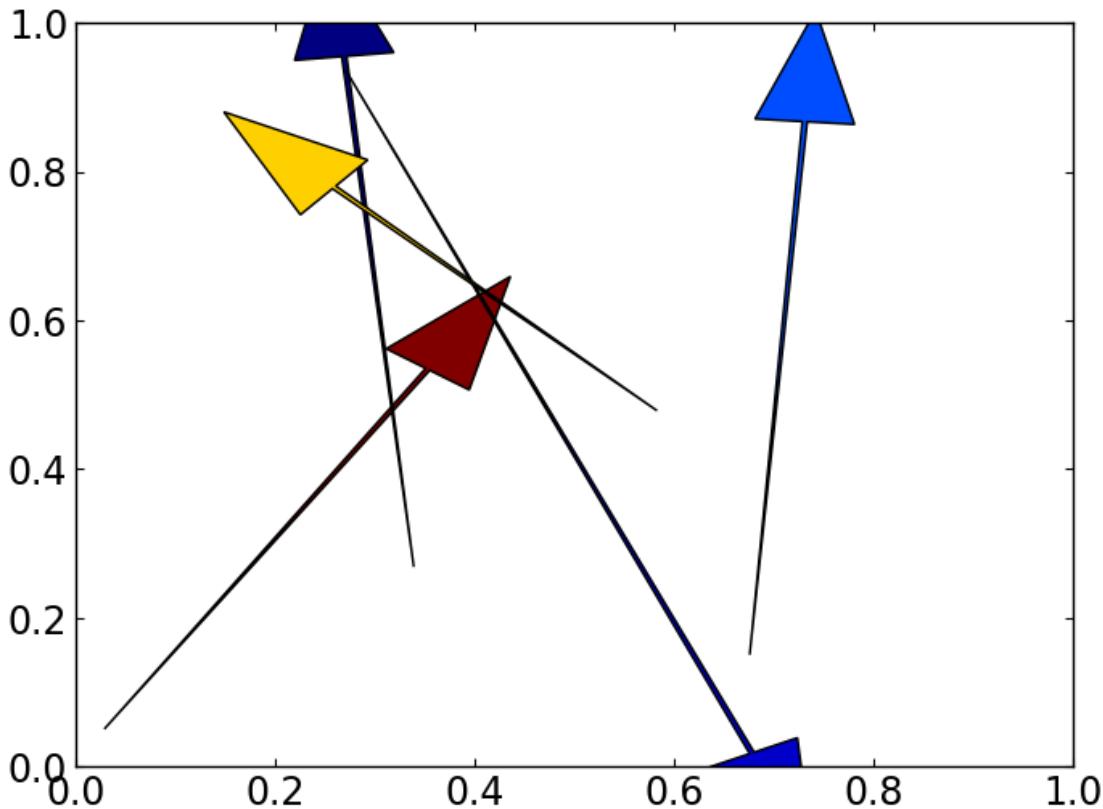
```

cNorm = colors.Normalize(vmin=np.min(DATA[:,4]), vmax=np.max(DATA[:,4]))

scalarMap = cmx.ScalarMappable(norm=cNorm,cmap=cmap)

for idx in range(0,len(DATA[:,1])):
    colorVal = scalarMap.to_rgba(DATA[idx,4])
    plt.arrow(DATA[idx,0], #x1
              DATA[idx,1], # y1
              DATA[idx,2]-DATA[idx,0], # x2 - x1
              DATA[idx,3]-DATA[idx,1], # y2 - y1
              facecolor=colorVal, edgecolor='k', width=0.005, head_width=0.1)

```



```

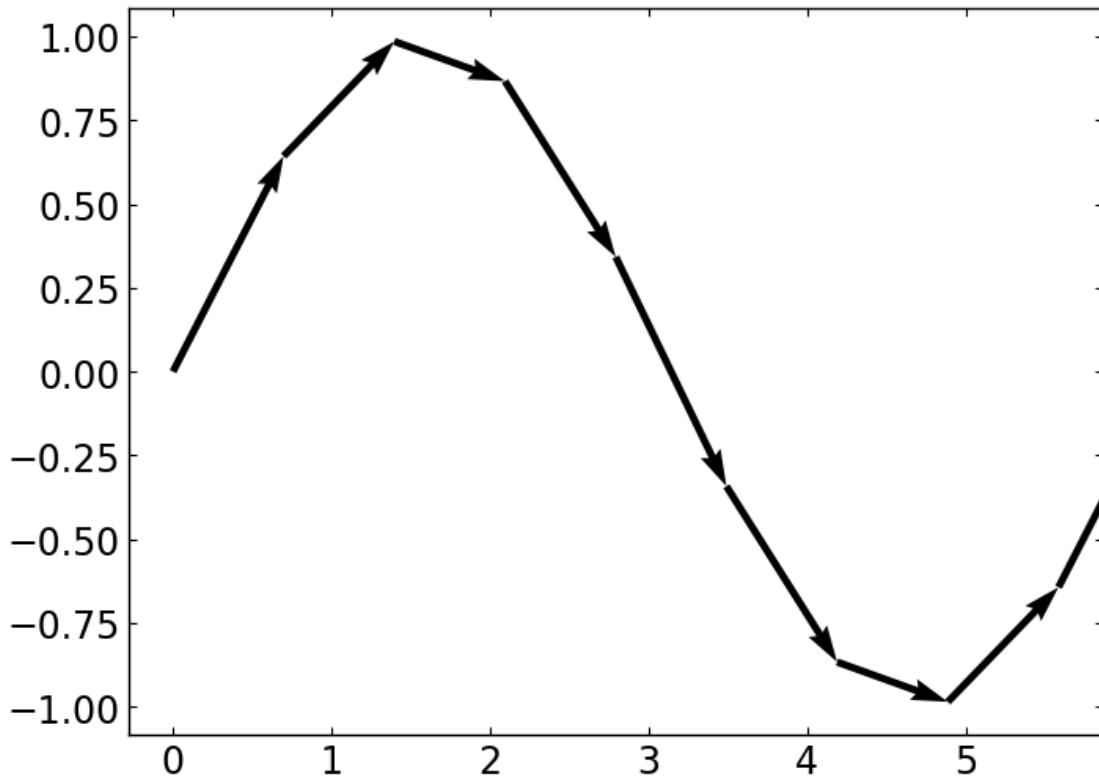
In [9]: """ 2-D vectors by arrows """
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 10)
y = np.sin(x)

plt.figure()
plt.quiver(x[:-1], y[:-1], x[1:]-x[:-1], y[1:]-y[:-1], scale_units='xy', angles='xy', s=100)

```

Out [9]: <matplotlib.quiver.Quiver at 0x7fade1d70a10>



```
In [16]: import matplotlib.patches as mpatches
         import matplotlib.pyplot as plt

%matplotlib inline

styles = mpatches.ArrowStyle.get_styles()

ncol = 2
nrow = (len(styles) + 1) // ncol
figheight = (nrow + 0.5)
fig1 = plt.figure(1, (4 * ncol / 1.5, figheight / 1.5))
fontsize = 0.2 * 70

ax = fig1.add_axes([0, 0, 1, 1], frameon=False, aspect=1.)

ax.set_xlim(0, 4 * ncol)
ax.set_ylim(0, figheight)
```

```

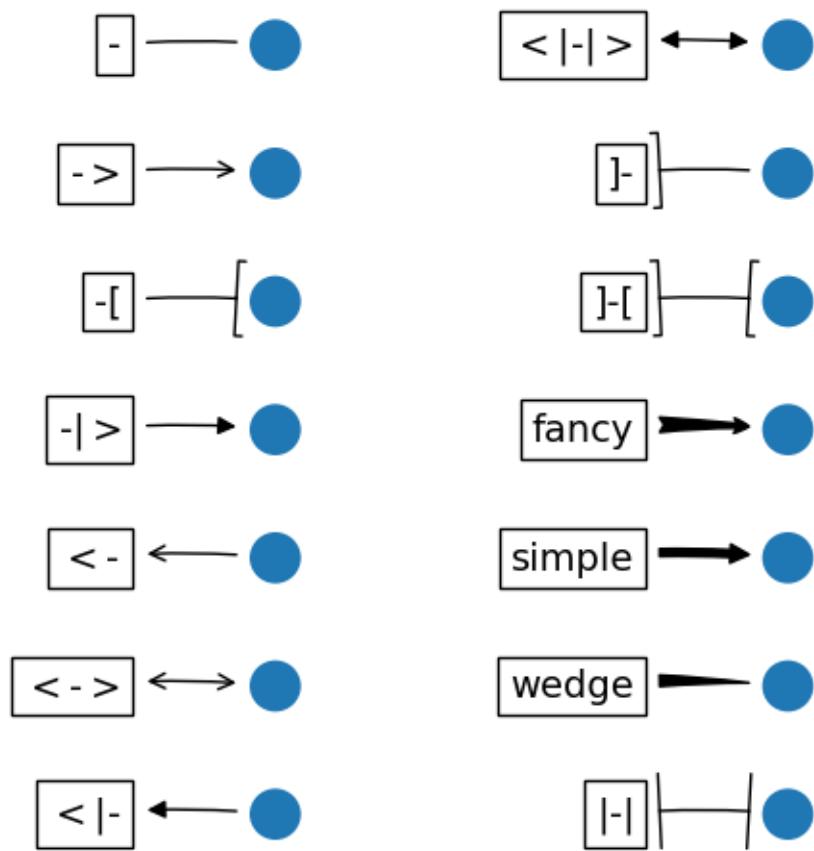
def to_texstring(s):
    s = s.replace("<", r"$<$")
    s = s.replace(">", r"$>$")
    s = s.replace("|", r"$|$")
    return s

for i, (stylename, styleclass) in enumerate(sorted(styles.items())):
    x = 3.2 + (i // nrow) * 4
    y = (figheight - 0.7 - i % nrow) # /figheight
    p = mpatches.Circle((x, y), 0.2)
    ax.add_patch(p)

    ax.annotate(to_texstring(stylename), (x, y),
                (x - 1.2, y),
                ha="right", va="center",
                size=fontsize,
                arrowprops=dict(arrowstyle=stylename,
                               patchB=p,
                               shrinkA=5,
                               shrinkB=5,
                               fc="k", ec="k",
                               connectionstyle="arc3,rad=-0.05",
                               ),
                bbox=dict(boxstyle="square", fc="w"))

ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)

```



1.1 Draw vector fields and streams

```
In [40]: from matplotlib.pyplot import cm
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

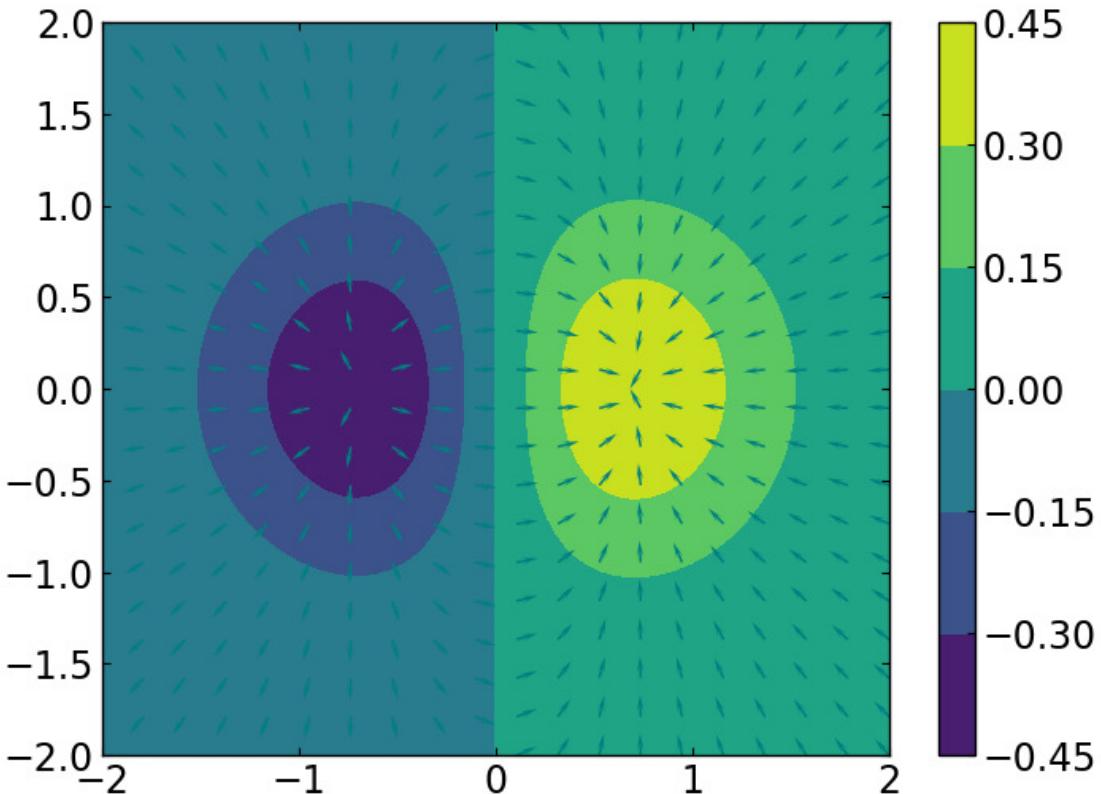
# Contour Plot
X, Y = np.mgrid[-2:2:100j, -2:2:100j]
Z = X*np.exp(-(X**2 + Y**2))
cp = plt.contourf(X, Y, Z)
cb = plt.colorbar(cp)

# Vector Field
Y, X = np.mgrid[-2:2:20j, -2:2:20j]
```

```

U =(1 - 2*(X**2))*np.exp(-((X**2)+(Y**2)))
V = -2*X*Y*np.exp(-((X**2)+(Y**2)))
speed = np.sqrt(U**2 + V**2)
UN = U/speed
VN = V/speed
quiv = plt.quiver(X, Y, UN, VN, # assign to var
                   color='Teal',
                   headlength=7)

```



```

In [42]: import sys
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

%matplotlib inline

def E(q, r0, x, y):
    """Return the electric field vector E=(Ex,Ey) due to charge q at r0."""
    den = ((x-r0[0])**2 + (y-r0[1])**2)**1.5
    return q * (x - r0[0]) / den, q * (y - r0[1]) / den

```

```

# Grid of x, y points
nx, ny = 64, 64
x = np.linspace(-2, 2, nx)
y = np.linspace(-2, 2, ny)
X, Y = np.meshgrid(x, y)

# Create a multipole with nq charges of alternating sign, equally spaced
# on the unit circle.
#nq = 2**int(sys.argv[1])
nq = 2
charges = []
for i in range(nq):
    q = i%2 * 2 - 1
    charges.append((q, (np.cos(2*np.pi*i/nq), np.sin(2*np.pi*i/nq)))))

# Electric field vector, E=(Ex, Ey), as separate components
Ex, Ey = np.zeros((ny, nx)), np.zeros((ny, nx))
for charge in charges:
    ex, ey = E(*charge, x=X, y=Y)
    Ex += ex
    Ey += ey

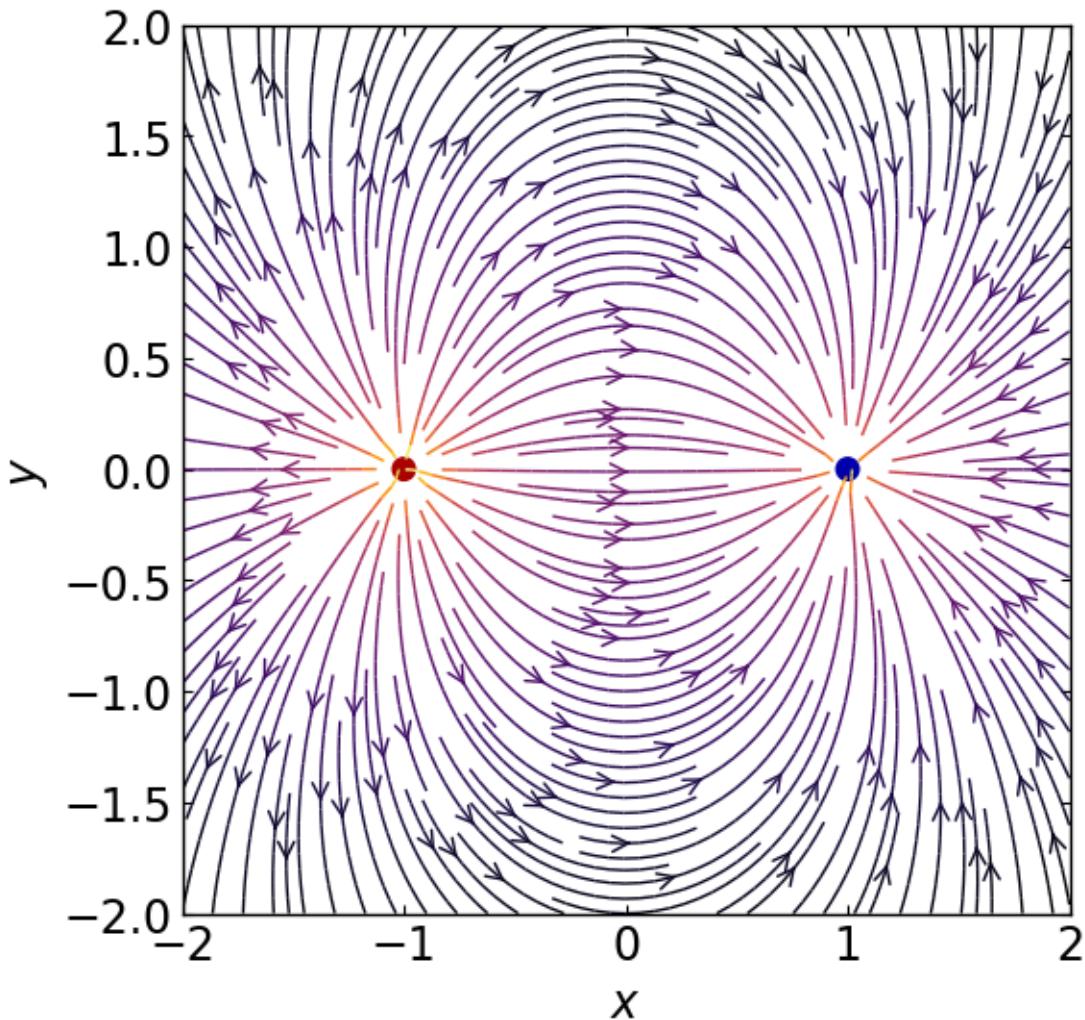
fig = plt.figure()
ax = fig.add_subplot(111)

# Plot the streamlines with an appropriate colormap and arrow style
color = np.log(np.sqrt(Ex**2 + Ey**2))
ax.streamplot(x, y, Ex, Ey, color=color, linewidth=1, cmap=plt.cm.inferno,
              density=2, arrowstyle='->', arrowsize=1.5)

# Add filled circles for the charges themselves
charge_colors = {True: '#aa0000', False: '#0000aa'}
for q, pos in charges:
    ax.add_artist(Circle(pos, 0.05, color=charge_colors[q>0]))

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_xlim(-2,2)
ax.set_ylim(-2,2)
ax.set_aspect('equal')

```



```
In [46]: '''
```

```
=====
```

Demonstration of advanced quiver and quiverkey functions

```
=====
```

Known problem: the plot autoscaling does not take into account the arrows, so those on the boundaries are often out of the picture. This is *not* an easy problem to solve in a perfectly general way. The workaround is to manually expand the axes.

```
'''
```

```
import matplotlib.pyplot as plt
import numpy as np
from numpy import ma
```

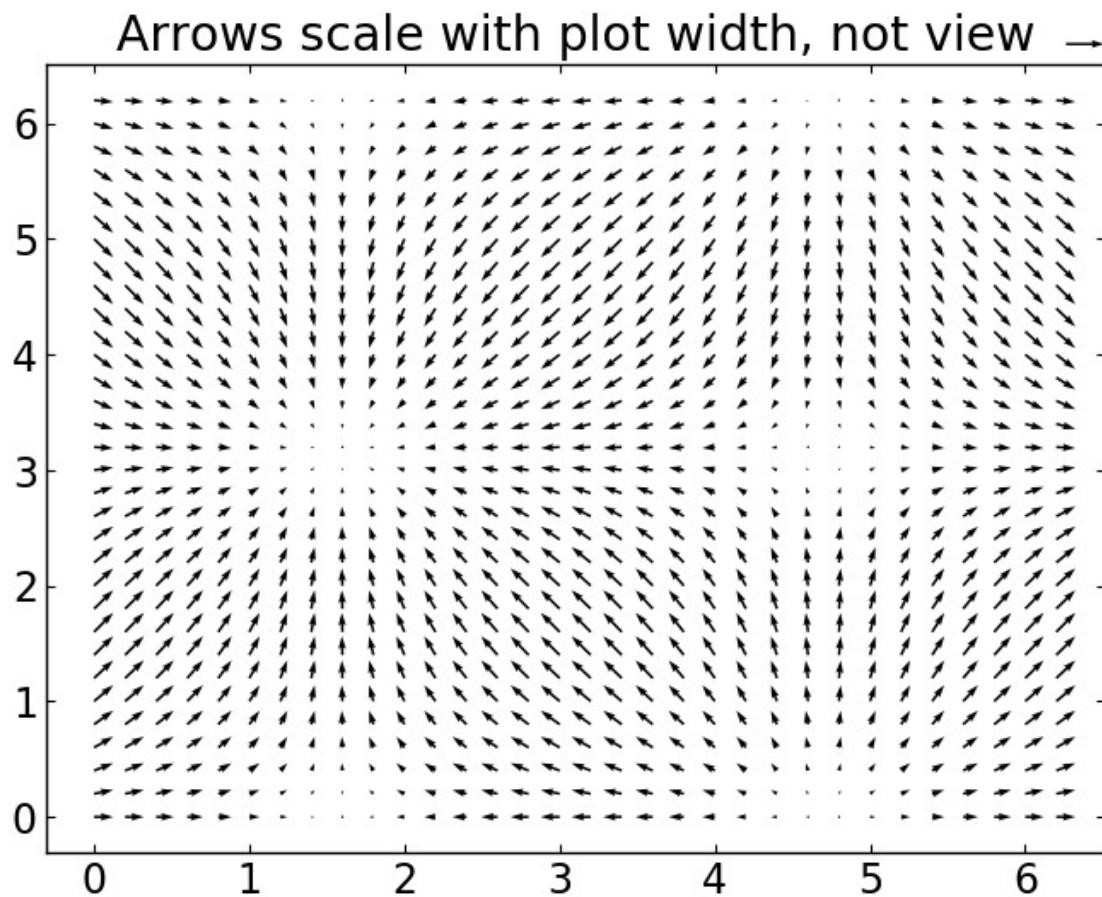
```
%matplotlib inline
```

```

X, Y = np.meshgrid(np.arange(0, 2 * np.pi, .2), np.arange(0, 2 * np.pi, .2))
U = np.cos(X)
V = np.sin(Y)

plt.figure()
plt.title('Arrows scale with plot width, not view')
Q = plt.quiver(X, Y, U, V, units='width')
qk = plt.quiverkey(Q, 0.9, 0.9, 2, r'$2 \frac{m}{s}$', labelpos='E',
                    coordinates='figure')

```



In [44]: ?plt.quiver

matplotlib4

January 30, 2019

0.0.1 Only axes

```
In [9]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.ticker as ticker

%matplotlib inline

# Setup a plot such that only the bottom spine is shown
def setup(ax):
    ax.spines['right'].set_color('none')
    ax.spines['left'].set_color('none')
    ax.yaxis.set_major_locator(ticker.NullLocator())
    ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
    ax.tick_params(which='major', width=1.00)
    ax.tick_params(which='major', length=5)
    ax.tick_params(which='minor', width=0.75)
    ax.tick_params(which='minor', length=2.5)
    ax.set_xlim(0, 5)
    ax.set_ylim(0, 1)
    ax.patch.set_alpha(0.0)

plt.figure(figsize=(8, 6))
n = 8

# Null Locator
ax = plt.subplot(n, 1, 1)
setup(ax)
ax.xaxis.set_major_locator(ticker.NullLocator())
ax.xaxis.set_minor_locator(ticker.NullLocator())
ax.text(0.0, 0.1, "NullLocator()", fontsize=14, transform=ax.transAxes)

# Multiple Locator
ax = plt.subplot(n, 1, 2)
setup(ax)
```

```

ax.xaxis.set_major_locator(ticker.MultipleLocator(0.5))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.1))
ax.text(0.0, 0.1, "MultipleLocator(0.5)", fontsize=14,
        transform=ax.transAxes)

# Fixed Locator
ax = plt.subplot(n, 1, 3)
setup(ax)
majors = [0, 1, 5]
ax.xaxis.set_major_locator(ticker.FixedLocator(majors))
minors = np.linspace(0, 1, 11)[1:-1]
ax.xaxis.set_minor_locator(ticker.FixedLocator(minors))
ax.text(0.0, 0.1, "FixedLocator([0, 1, 5])", fontsize=14,
        transform=ax.transAxes)

# Linear Locator
ax = plt.subplot(n, 1, 4)
setup(ax)
ax.xaxis.set_major_locator(ticker.LinearLocator(3))
ax.xaxis.set_minor_locator(ticker.LinearLocator(31))
ax.text(0.0, 0.1, "LinearLocator(numticks=3)",
        fontsize=14, transform=ax.transAxes)

# Index Locator
ax = plt.subplot(n, 1, 5)
setup(ax)
ax.plot(range(0, 5), [0]*5, color='White')
ax.xaxis.set_major_locator(ticker.IndexLocator(base=.5, offset=.25))
ax.text(0.0, 0.1, "IndexLocator(base=0.5, offset=0.25)",
        fontsize=14, transform=ax.transAxes)

# Auto Locator
ax = plt.subplot(n, 1, 6)
setup(ax)
ax.xaxis.set_major_locator(ticker.AutoLocator())
ax.xaxis.set_minor_locator(ticker.AutoMinorLocator())
ax.text(0.0, 0.1, "AutoLocator()", fontsize=14, transform=ax.transAxes)

# MaxN Locator
ax = plt.subplot(n, 1, 7)
setup(ax)
ax.xaxis.set_major_locator(ticker.MaxNLocator(4))
ax.xaxis.set_minor_locator(ticker.MaxNLocator(40))
ax.text(0.0, 0.1, "MaxNLocator(n=4)", fontsize=14, transform=ax.transAxes)

# Log Locator
ax = plt.subplot(n, 1, 8)
setup(ax)

```

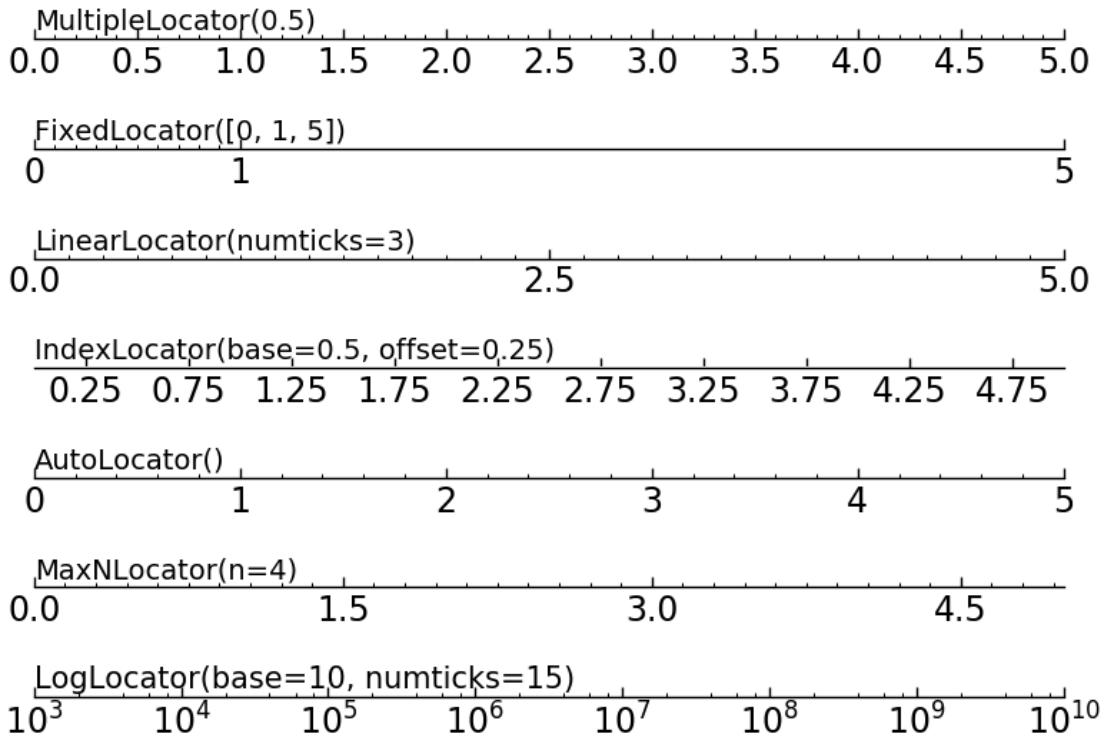
```

ax.set_xlim(10**3, 10**10)
ax.set_xscale('log')
ax.xaxis.set_major_locator(ticker.LogLocator(base=10.0, numticks=15))
ax.text(0.0, 0.1, "LogLocator(base=10, numticks=15)",
        fontsize=15, transform=ax.transAxes)

# Push the top of the top axes outside the figure because we only show the
# bottom spine.
plt.subplots_adjust(left=0.05, right=0.95, bottom=0.05, top=1.05)

```

NullLocator()



```

In [24]: import numpy as np
         import matplotlib.pyplot as plt
         import matplotlib.ticker as ticker

%matplotlib inline

# Setup a plot such that only the bottom spine is shown
def setup(ax):

```

```

        ax.spines['top'].set_color('none')
        ax.spines['bottom'].set_color('none')
        ax.xaxis.set_major_locator(ticker.NullLocator())
        ax.spines['left'].set_color('none')
        ax.yaxis.set_ticks_position('right')
        ax.tick_params(which='major', width=1.00)
        ax.tick_params(which='major', length=5)
        ax.tick_params(which='minor', width=0.75)
        ax.tick_params(which='minor', length=2.5)
        ax.set_xlim(0, 5)
        ax.set_ylim(0, 1)
        ax.patch.set_alpha(0.0)

plt.figure(figsize=(8, 6))
n = 8

# Null Locator
ax = plt.subplot(1, n, 1)
setup(ax)
ax.yaxis.set_major_locator(ticker.NullLocator())
ax.yaxis.set_minor_locator(ticker.NullLocator())
ax.text(1.2, 0.5, "NullLocator()", fontsize=14, transform=ax.transAxes, rotation=90, va='center')

# Multiple Locator
ax = plt.subplot(1, n, 2)
setup(ax)
ax.yaxis.set_major_locator(ticker.MultipleLocator(0.5))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(0.1))
ax.text(1.8, 0.5, "MultipleLocator(0.5)", fontsize=14, rotation=90, \
        transform=ax.transAxes, va='center')

# Fixed Locator
ax = plt.subplot(1, n, 3)
setup(ax)
majors = [0, 1, 5]
ax.yaxis.set_major_locator(ticker.FixedLocator(majors))
minors = np.linspace(0, 1, 11)[1:-1]
ax.yaxis.set_minor_locator(ticker.FixedLocator(minors))
ax.text(1.8, 0.5, "FixedLocator([0, 1, 5])", fontsize=14, rotation=90, \
        transform=ax.transAxes, va='center')

# Linear Locator
ax = plt.subplot(1, n, 4)
setup(ax)
ax.yaxis.set_major_locator(ticker.LinearLocator(3))
ax.yaxis.set_minor_locator(ticker.LinearLocator(31))
ax.text(1.8, 0.5, "LinearLocator(numticks=3)", rotation=90, \
        transform=ax.transAxes, va='center')

```

```

    fontsize=14, transform=ax.transAxes, va='center')

# Index Locator
ax = plt.subplot(1, n, 5)
setup(ax)
ax.plot(range(0, 5), [0]*5, color='White')
ax.yaxis.set_major_locator(ticker.IndexLocator(base=.5, offset=.25))
ax.text(1.8, 0.5, "IndexLocator(base=0.5, offset=0.25)", rotation=90, \
        fontsize=14, transform=ax.transAxes, va='center')

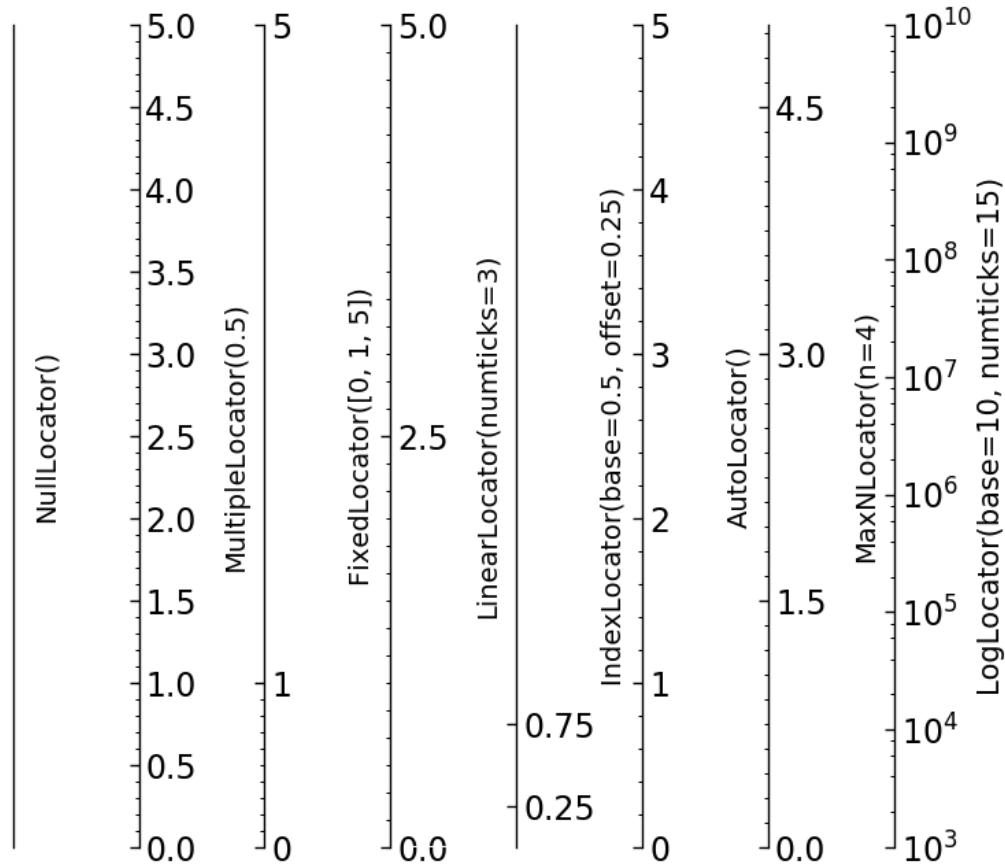
# Auto Locator
ax = plt.subplot(1, n, 6)
setup(ax)
ax.yaxis.set_major_locator(ticker.AutoLocator())
ax.yaxis.set_minor_locator(ticker.AutoMinorLocator())
ax.text(1.8, 0.5, "AutoLocator()", fontsize=14, transform=ax.transAxes, rotation=90, va='center')

# MaxN Locator
ax = plt.subplot(1, n, 7)
setup(ax)
ax.yaxis.set_major_locator(ticker.MaxNLocator(4))
ax.yaxis.set_minor_locator(ticker.MaxNLocator(40))
ax.text(1.8, 0.5, "MaxNLocator(n=4)", fontsize=14, transform=ax.transAxes, rotation=90, va='center')

# Log Locator
ax = plt.subplot(1, n, 8)
setup(ax)
ax.set_ylim(10**3, 10**10)
ax.set_yscale('log')
ax.yaxis.set_major_locator(ticker.LogLocator(base=10.0, numticks=15))
ax.text(1.8, 0.5, "LogLocator(base=10, numticks=15)", rotation=90, \
        fontsize=15, transform=ax.transAxes, va='center')

# Push the top of the top axes outside the figure because we only show the
# bottom spine.
plt.subplots_adjust(left=0.05, right=0.95, bottom=0.05, top=1.05)

```



0.0.2 scaling the tick for the data with different units

In [2]: *""" scaling the tick for the data with different units """*

```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from scipy.constants import h, c, e

%matplotlib inline

def E(wavelength):
    return (h*c)/(wavelength*e)
def getWaveLength(energy):
    return (h*c)/(energy*e)
def getCounts(normcounts):
    return normcounts*1000

wavelen = np.linspace(800e-9,1600e-9,200)
E_eV = E(wavelen)

```

```

loc, scale = 950e-9, 3.0
counts = mlab.normpdf(wavelen,950e-9,100e-9)/100
counts_norm = counts/10000

fig, ax1 = plt.subplots()

ax2 = ax1.twinx()
ax3 = ax1.twiny()

plt.ticklabel_format(style='sci', scilimits=(0,0))

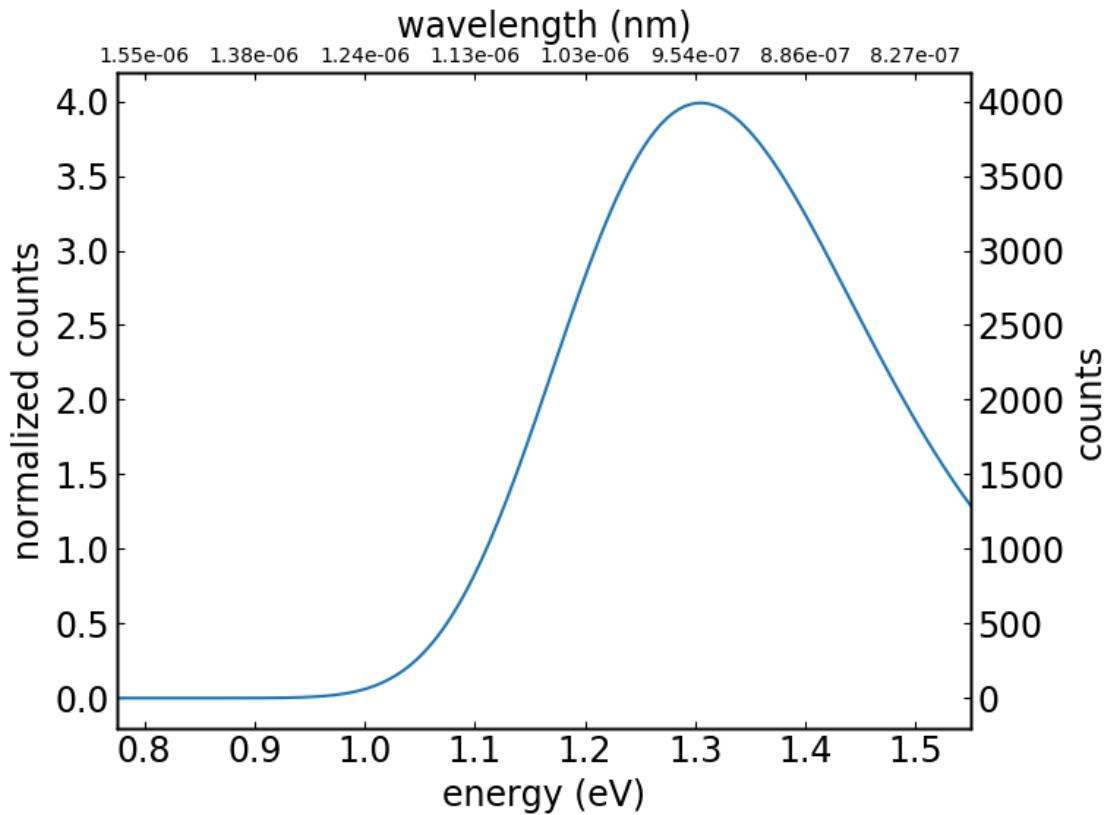
ax1.plot(E_eV, counts_norm)
ax1.set_xlim(E(1600e-9),E(800e-9))
ax1.set_ylabel('normalized counts')
ax1.set_xlabel('energy (eV)')
ax2.set_ylabel('counts')
ax3.set_xlabel('wavelength (nm)')
ax3.ticklabel_format(style='plain')

# get the primary axis x tick locations in plot units
xtickloc = ax1.get_xticks()
# set the second axis ticks to the same locations
ax3.set_xticks(xtickloc)
# calculate new values for the second axis tick labels, format them, and set them
x2labels = ['{:3g}'.format(x) for x in getWaveLength(xtickloc)]
ax3.set_xticklabels(x2labels, fontsize=10)
# force the bounds to be the same
ax3.set_xlim(ax1.get_xlim())

#same for y
ytickloc = ax1.get_yticks()
ax2.set_yticks(ytickloc)
ax2.set_yticklabels([str(int(y)) for y in getCounts(ytickloc)])
ax2.set_ylim(ax1.get_ylim())

plt.tight_layout()

```



```
In [4]: import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)
ax2 = ax1.twiny()

X = np.linspace(0,1,1000)
Y = np.cos(X*20)

ax1.plot(X,Y)
ax1.set_xlabel(r"Original x-axis: $X$")

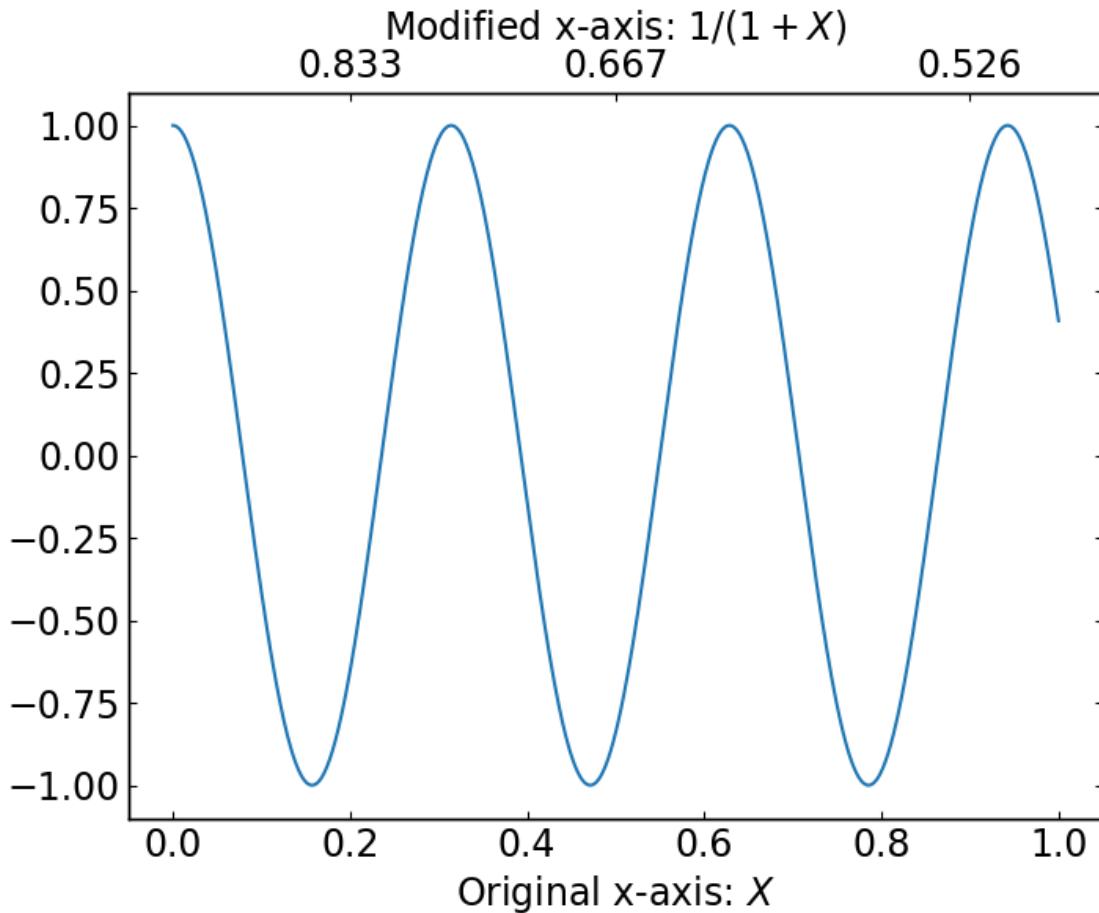
new_tick_locations = np.array([.2, .5, .9])

def tick_function(X):
    V = 1/(1+X)
    return ["%.3f" % z for z in V]

ax2.set_xlim(ax1.get_xlim())
ax2.set_xticks(new_tick_locations)
```

```
ax2.set_xticklabels(tick_function(new_tick_locations))
ax2.set_xlabel(r"Modified x-axis: $1/(1+X)$")
```

Out [4]: <matplotlib.text.Text at 0x7f687647f810>



```
In [5]: import numpy as np
import matplotlib.pyplot as plt

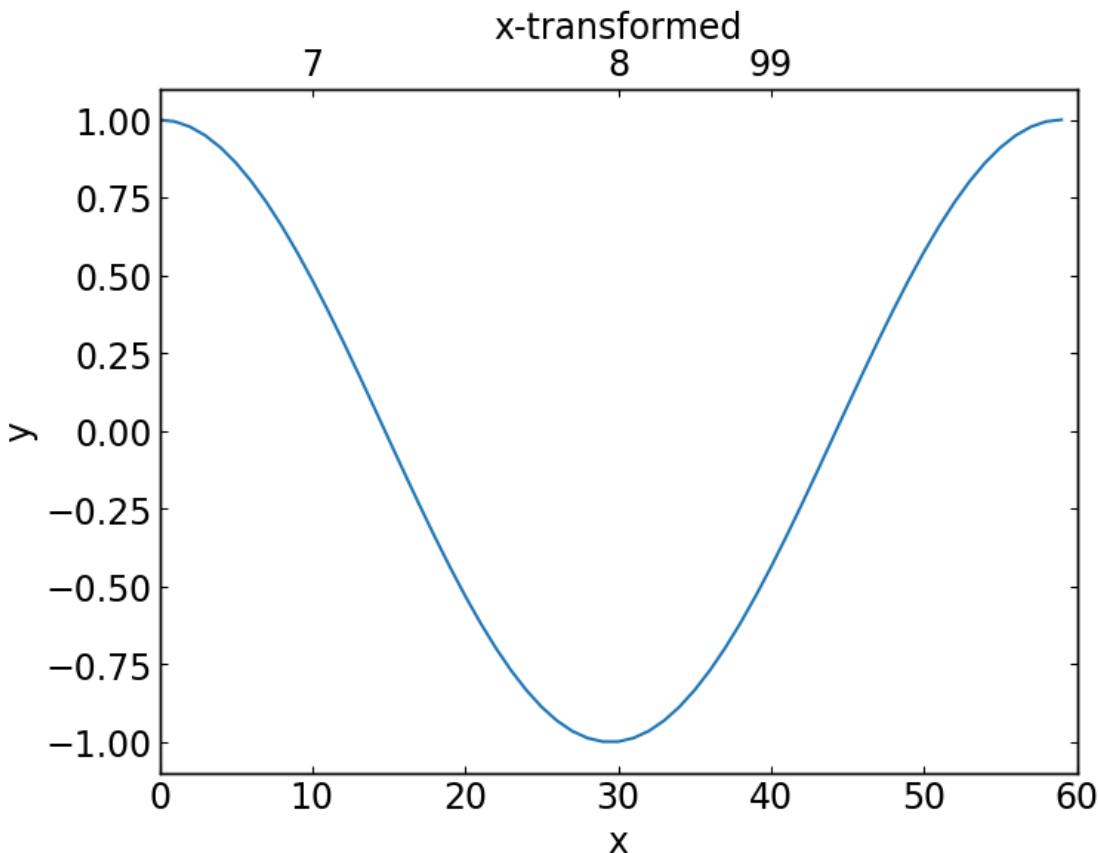
fig = plt.figure()
ax1 = fig.add_subplot(111)

a = np.cos(2*np.pi*np.linspace(0, 1, 60.))
ax1.plot(range(60), a)

ax1.set_xlim(0, 60)
ax1.set_xlabel("x")
ax1.set_ylabel("y")
```

```
ax2 = ax1.twiny()
ax2.set_xlabel("x-transformed")
ax2.set_xlim(0, 60)
ax2.set_xticks([10, 30, 40])
ax2.set_xticklabels(['7', '8', '99'])
```

Out[5]: [`<matplotlib.text.Text at 0x7f6874fdce50>`,
`<matplotlib.text.Text at 0x7f6875003610>`,
`<matplotlib.text.Text at 0x7f6874fc1410>`]



0.1 Name Colors

```
In [1]: from __future__ import division

import matplotlib.pyplot as plt
from matplotlib import colors as mcolors

colors = dict(mcolors.BASE_COLORS, **mcolors.CSS4_COLORS)
```

```

# Sort colors by hue, saturation, value and name.
by_hsv = sorted((tuple(mcolors.rgb_to_hsv(mcolors.to_rgba(color)[:-3])), name)
                 for name, color in colors.items())
sorted_names = [name for hsv, name in by_hsv]

n = len(sorted_names)
ncols = 4
nrows = n // ncols + 1

fig, ax = plt.subplots(figsize=(8, 5))

# Get height and width
X, Y = fig.get_dpi() * fig.get_size_inches()
h = Y / (nrows + 1)
w = X / ncols

for i, name in enumerate(sorted_names):
    col = i % ncols
    row = i // ncols
    y = Y - (row * h) - h

    xi_line = w * (col + 0.05)
    xf_line = w * (col + 0.25)
    xi_text = w * (col + 0.3)

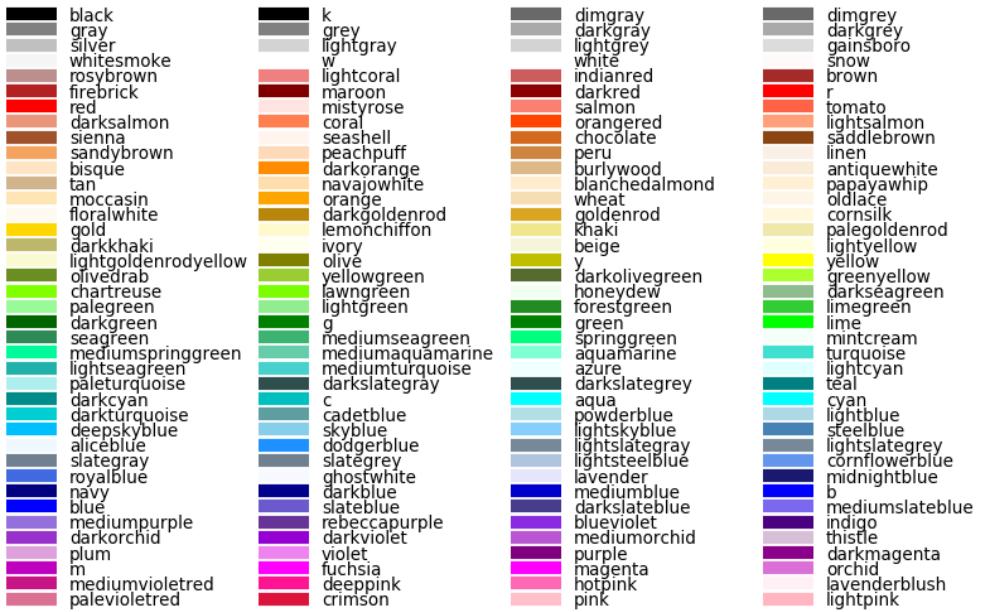
    ax.text(xi_text, y, name, fontsize=(h * 0.8),
            horizontalalignment='left',
            verticalalignment='center')

    ax.hlines(y + h * 0.1, xi_line, xf_line,
              color=colors[name], linewidth=(h * 0.6))

ax.set_xlim(0, X)
ax.set_ylim(0, Y)
ax.set_axis_off()

fig.subplots_adjust(left=0, right=1,
                    top=1, bottom=0,
                    hspace=0, wspace=0)
plt.show()

```



0.2 Handle a NAN values in the plot

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

t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(2 * 2*np.pi * t)
t[41:60] = np.nan

plt.subplot(2, 1, 1)
plt.plot(t, s, '-.', lw=2)

plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('A sine wave with a gap of NaNs between 0.4 and 0.6')
plt.grid(True)

plt.subplot(2, 1, 2)
t[0] = np.nan
t[-1] = np.nan
plt.plot(t, s, '-.', lw=2)
plt.title('Also with NaN in first and last point')

plt.xlabel('time (s)')
plt.ylabel('more nans')
```

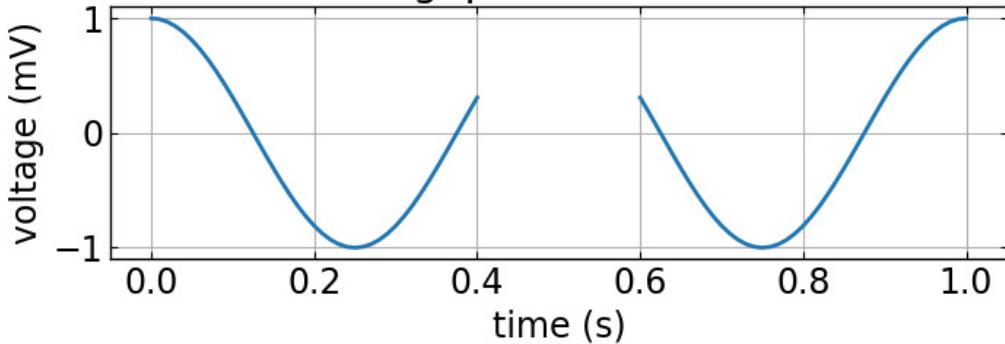
```

plt.grid(True)

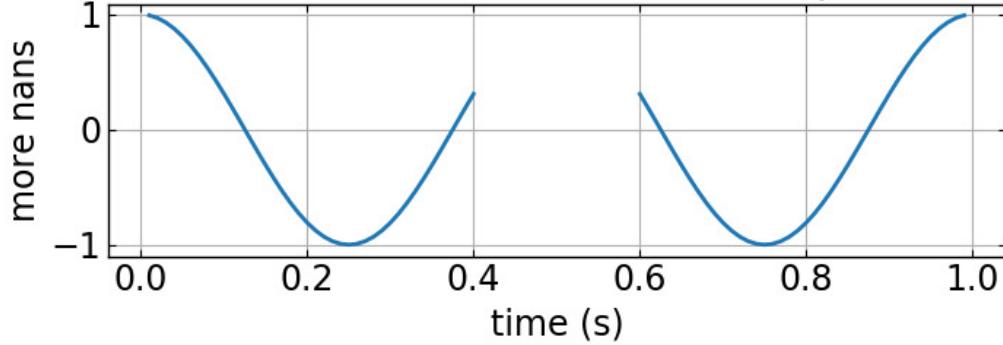
plt.tight_layout()
plt.show()

```

A sine wave with a gap of NaNs between 0.4 and 0.6



Also with NaN in first and last point



0.3 Plot with masked data

```

In [3]: import matplotlib.pyplot as plt
        import numpy as np

x = np.arange(0, 2*np.pi, 0.02)
y = np.sin(x)
y1 = np.sin(2*x)
y2 = np.sin(3*x)
ym1 = np.ma.masked_where(y1 > 0.5, y1)
ym2 = np.ma.masked_where(y2 < -0.5, y2)

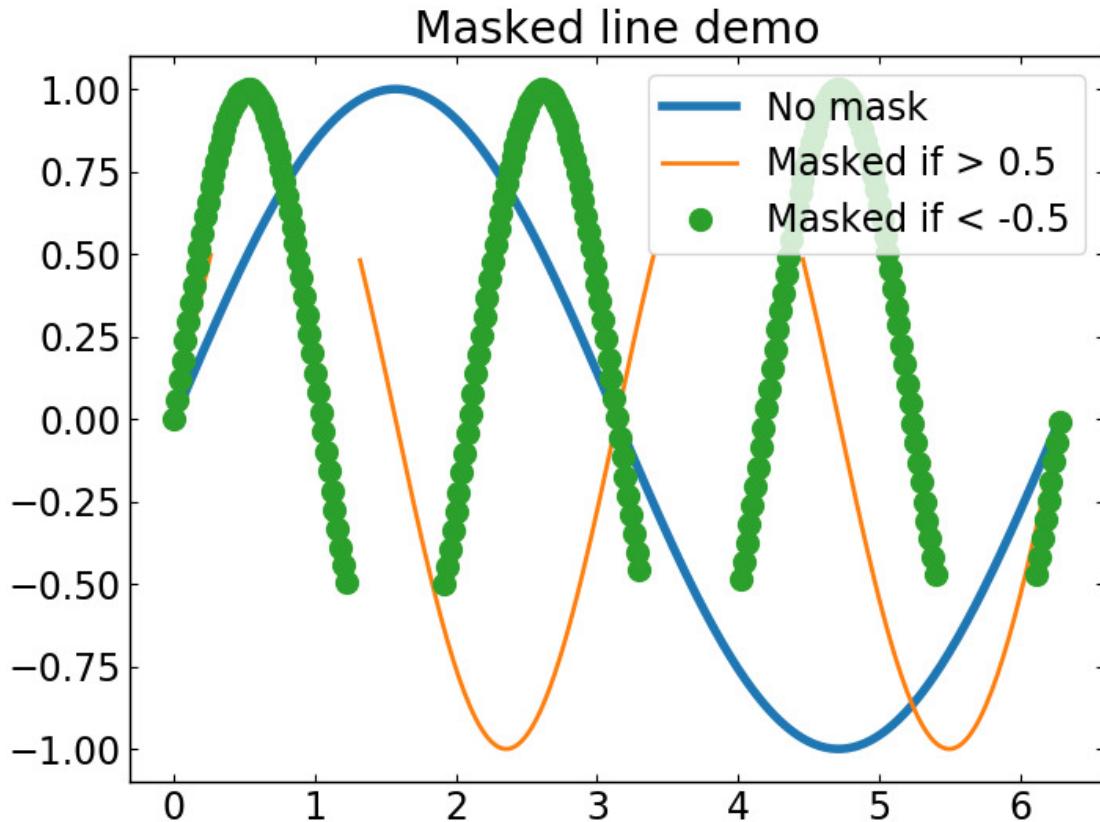
lines = plt.plot(x, y, x, ym1, x, ym2, 'o')
plt.setp(lines[0], linewidth=4)
plt.setp(lines[1], linewidth=2)
plt.setp(lines[2], markersize=10)

```

```

plt.legend(['No mask', 'Masked if > 0.5', 'Masked if < -0.5'],
          loc='upper right')
plt.title('Masked line demo')
plt.show()

```



0.4 Stack Plot

```

In [4]: import numpy as np
        import matplotlib.pyplot as plt

        x = [1, 2, 3, 4, 5]
        y1 = [1, 1, 2, 3, 5]
        y2 = [0, 4, 2, 6, 8]
        y3 = [1, 3, 5, 7, 9]

        y = np.vstack([y1, y2, y3])

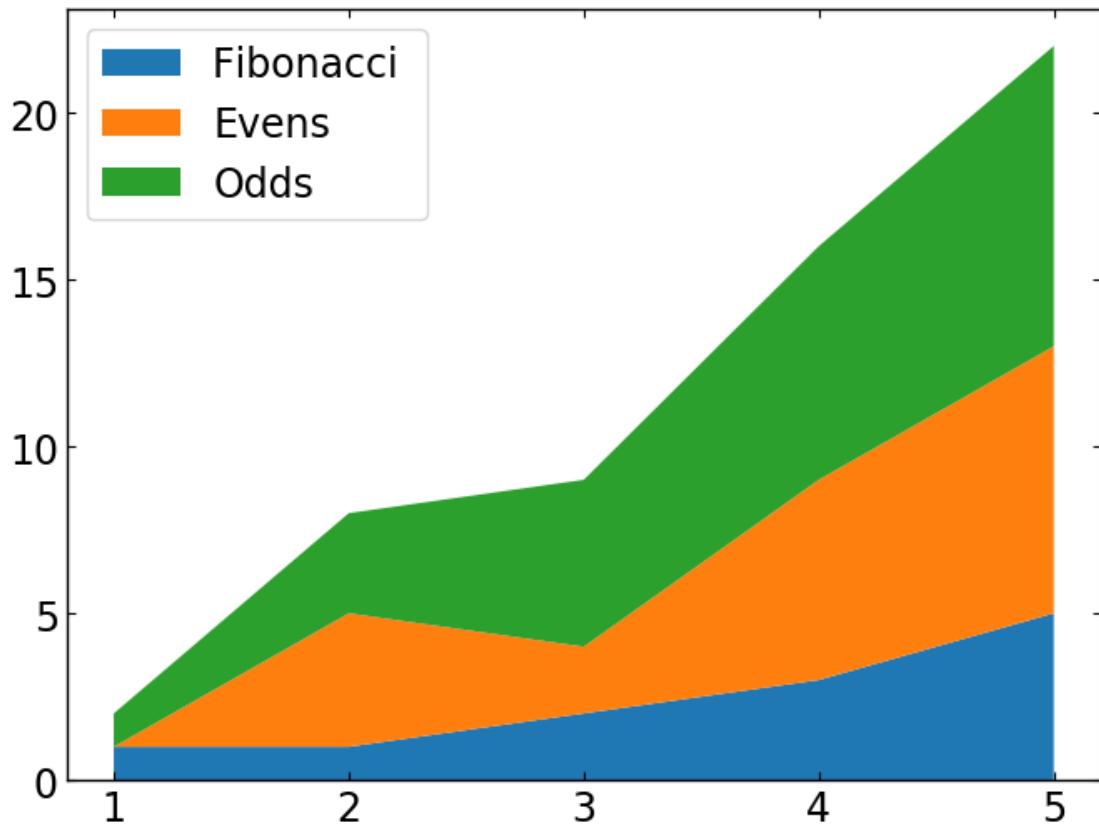
        labels = ["Fibonacci ", "Evens", "Odds"]

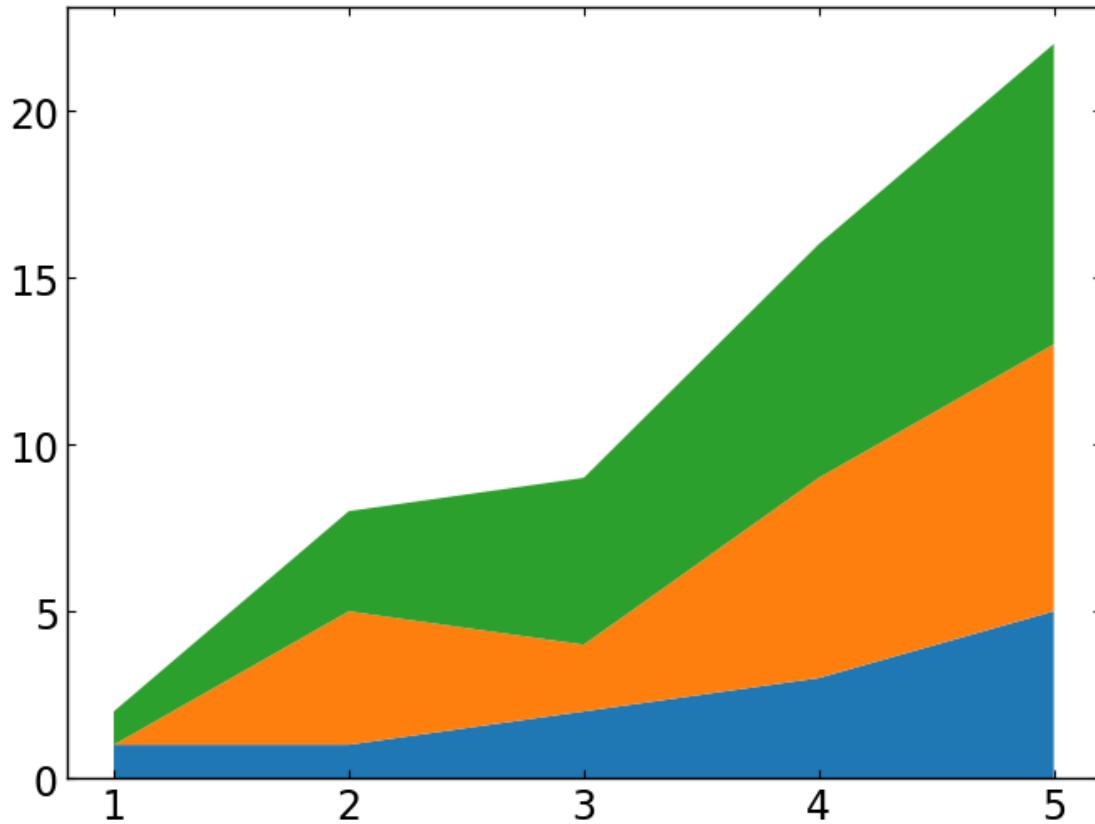
        fig, ax = plt.subplots()

```

```
ax.stackplot(x, y1, y2, y3, labels=labels)
ax.legend(loc=2)
plt.show()

fig, ax = plt.subplots()
ax.stackplot(x, y)
plt.show()
```





0.5 Color by y-value

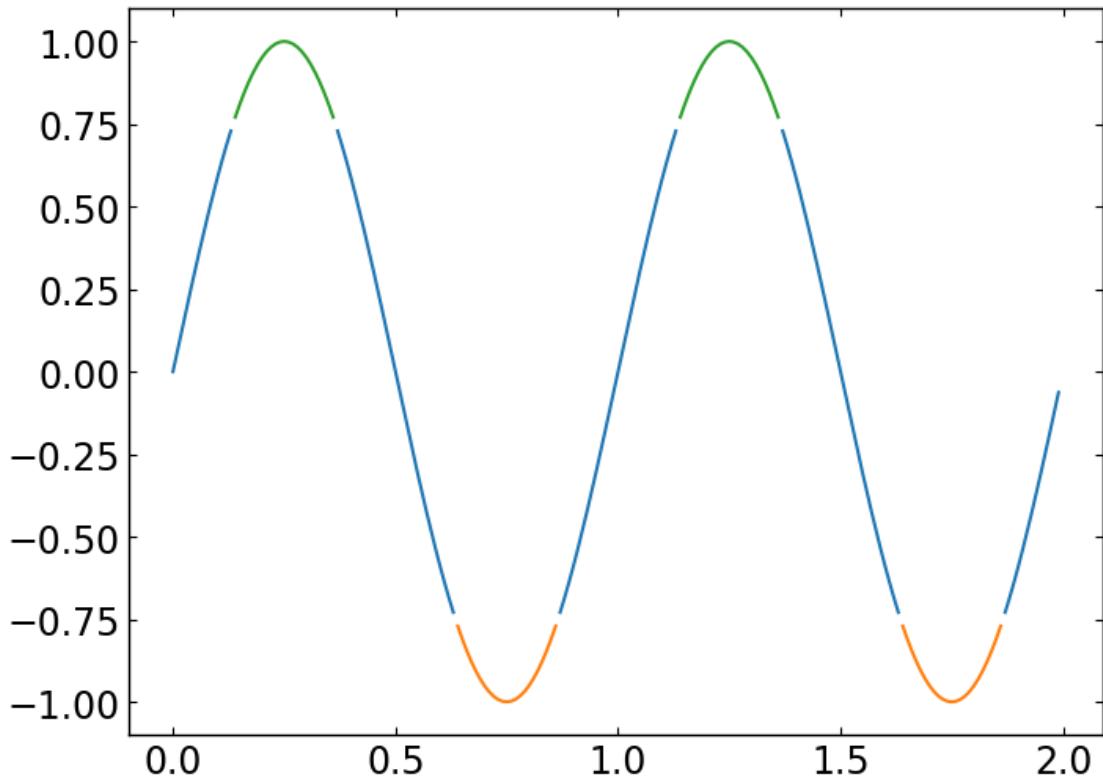
```
In [5]: import numpy as np
        import matplotlib.pyplot as plt

        t = np.arange(0.0, 2.0, 0.01)
        s = np.sin(2 * np.pi * t)

        upper = 0.77
        lower = -0.77

        supper = np.ma.masked_where(s < upper, s)
        slower = np.ma.masked_where(s > lower, s)
        smiddle = np.ma.masked_where(np.logical_or(s < lower, s > upper), s)

        fig, ax = plt.subplots()
        ax.plot(t, smiddle, t, slower, t, supper)
        plt.show()
```



```
In [6]: print supper
```

```
-- -- -- -- -- -- -- -- -- 0.7705132427757893
0.8090169943749475 0.8443279255020151 0.8763066800438637
0.9048270524660196 0.9297764858882513 0.9510565162951535
0.9685831611286311 0.9822872507286886 0.9921147013144779
0.9980267284282716 1.0 0.9980267284282716 0.9921147013144778
0.9822872507286886 0.9685831611286312 0.9510565162951536
0.9297764858882513 0.9048270524660195 0.8763066800438635 0.844327925502015
0.8090169943749475 0.7705132427757893 -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- 0.7705132427757893
0.8090169943749478 0.8443279255020147 0.8763066800438631
0.9048270524660194 0.9297764858882511 0.9510565162951535 0.968583161128631
0.9822872507286886 0.9921147013144778 0.9980267284282716 1.0
0.9980267284282716 0.9921147013144779 0.9822872507286886
0.9685831611286311 0.9510565162951536 0.9297764858882517
0.9048270524660192 0.8763066800438634 0.8443279255020151
0.8090169943749477 0.7705132427757886 -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- --
```

--]

mimic_alpha

January 30, 2019

0.0.1 mimic_alpha

- Since transparency (alpha) is not allowed in .eps figures, so this routine help to postulate the color blended with background for mimicking transparency.

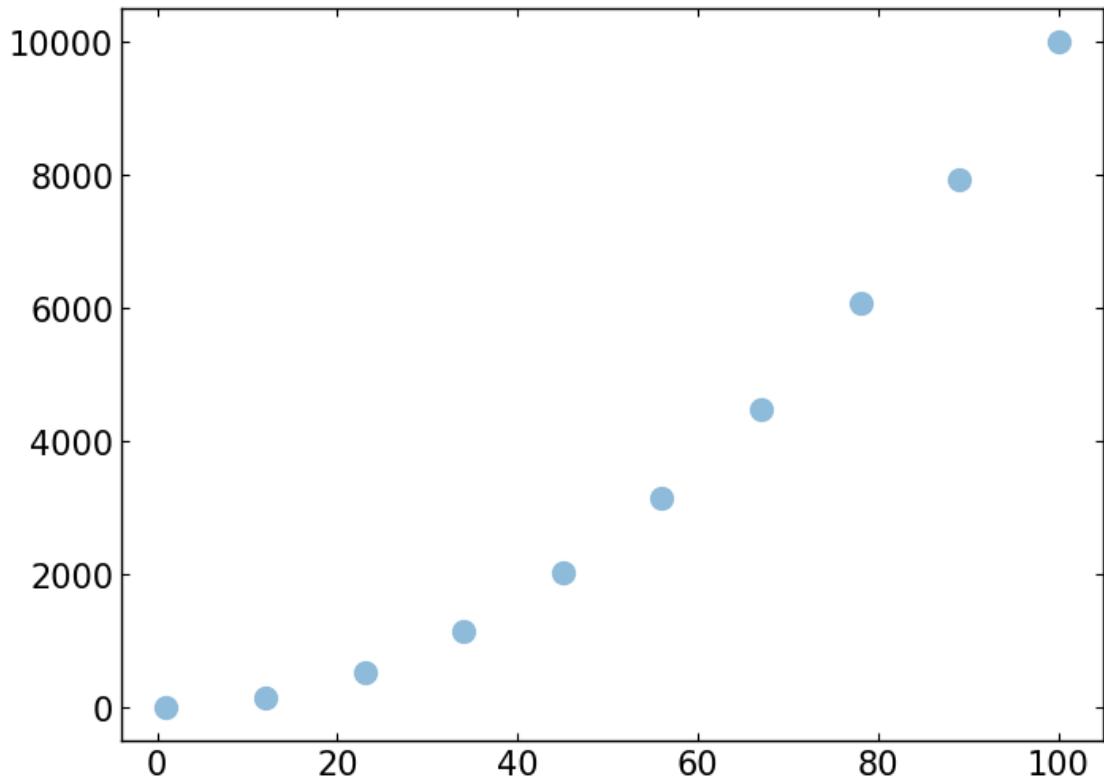
```
In [32]: import mimic_alpha as ma
          import matplotlib.pyplot as plt
          import numpy as np
          %matplotlib inline

fig = plt.figure()
ax  = fig.add_subplot(111)

x = np.linspace(1,100,10)
y = x*x

alpcolor = ma.colorAlpha_to_rgb('C0',0.5)
ax.plot(x,y,'o',markersize=10,color=alpcolor[0])

Out[32]: []
```



```
In [31]: print type(alpcolor), type(alpcolor[0])
```

```
<type 'list'> <type 'numpy.ndarray'>
```

numpy1

January 30, 2019

```
In [1]: import numpy as np
```

```
In [2]: %pylab inline
```

Populating the interactive namespace from numpy and matplotlib

```
In [3]: np.cast['f'](np.pi)
```

```
Out[3]: array(3.1415927410125732, dtype=float32)
```

```
In [4]: a = np.array([0,1,2,3])
      print type(a)
```

```
      print a
```

```
<type 'numpy.ndarray'>
[0 1 2 3]
```

```
In [5]: L = range(100)
      %timeit [i**2 for i in L]
```

```
a = np.arange(100)
%timeit a**2
```

The slowest run took 4.65 times longer than the fastest. This could mean that an intermediate 100000 loops, best of 3: 5.85 µs per loop

The slowest run took 30.90 times longer than the fastest. This could mean that an intermediate 1000000 loops, best of 3: 455 ns per loop

```
In [6]: np.lookfor('square root')
```

```
Search results for 'square root'
-----
```

```
numpy.sqrt
```

```
    Return the positive square-root of an array, element-wise.
numpy.ma.sqrt
```

```
Return the positive square-root of an array, element-wise.  
numpy.roots  
    Return the roots of a polynomial with coefficients given in p.  
numpy.poly  
    Find the coefficients of a polynomial with the given sequence of roots.  
numpy.std  
    Compute the standard deviation along the specified axis.  
numpy.nanstd  
    Compute the standard deviation along the specified axis, while  
numpy.poly1d  
    A one-dimensional polynomial class.  
numpy.histogram  
    Compute the histogram of a set of data.  
numpy.linalg.cond  
    Compute the condition number of a matrix.
```

In [7]: `#np.con*`?

```
In [8]: import numpy as np  
def addsubtract(a,b):  
    if a > b:  
        return a - b  
    else:  
        return a + b  
  
vec_addsubtract = np.vectorize(addsubtract)  
  
vec_addsubtract([0,3,6,9],[1,3,5,7])
```

Out[8]: `array([1, 6, 1, 2])`

0.1 Arrays

In [9]: `a = np.array([0,1,2,3])`

```
print a  
print a.ndim  
print a.shape  
print len(a)  
  
b = a.copy()  
print type(b)  
print b  
c = b.tolist()  
print type(c)  
print c
```

```
[0 1 2 3]  
1
```

```

(4,)
4
<type 'numpy.ndarray'>
[0 1 2 3]
<type 'list'>
[0, 1, 2, 3]

In [10]: b = np.array([[0,1,2],[3,4,5]])      # 2x3 array
          print b
          print b.ndim
          print b.shape
          print len(b)

          c = np.array([[[1],[2]],[[3],[4]]])
          print c
          print c.ndim
          print c.shape
          print len(c)

[[0 1 2]
 [3 4 5]]
2
(2, 3)
2
[[[1]
 [2]]]

[[3]
 [4]]]
3
(2, 2, 1)
2

In [11]: a = np.arange(10)
          print a

          b = np.arange(1,9,2)
          print b

          c = np.linspace(0,1,6)  # start, end, num-points
          print c

          d = np.linspace(0,1,5,endpoint=False)
          print d

[0 1 2 3 4 5 6 7 8 9]
[1 3 5 7]

```

```
[ 0.  0.2  0.4  0.6  0.8  1. ]
[ 0.  0.2  0.4  0.6  0.8]
```

```
In [12]: a = np.ones((3,3))    # tuple
          print a

          b = np.zeros((2,2))
          print b

          c = np.eye(3)
          print c

          d = np.diag(np.array([1,2,3,4]))
          print d
```

```
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
[[ 0.  0.]
 [ 0.  0.]]
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

```
In [13]: a = np.random.rand(4)    # uniform
          print a

          b = np.random.rand(4)    # Gaussian
          print b

          np.random.seed(1234)    # setting the random seed

[ 0.75398477  0.5853438   0.39072521  0.27453421]
[ 0.75006763  0.82466875  0.88059924  0.86332917]
```

```
In [14]: a = np.arange(10)
          a[5:] = 10
          print a

          b = np.arange(5)
          a[5:] = b[::-1]
          print a
```

```
[ 0  1  2  3  4 10 10 10 10 10]  
[0  1  2  3  4  4 3  2  1  0]
```

```
In [15]: a = np.arange(5) + np.arange(0,51,10)[:,np.newaxis]  
        print a
```

```
[[ 0  1  2  3  4]  
 [10 11 12 13 14]  
 [20 21 22 23 24]  
 [30 31 32 33 34]  
 [40 41 42 43 44]  
 [50 51 52 53 54]]
```

```
In [1]: a = np.tile(np.arange(0,40,10),(3,1))
```

```
        print a
```

```
        a = a.T
```

```
        print a
```

```
        b,c,d,e = a  
        print b  
        print c  
        print d  
        print e
```

```
        print a[0]
```

```
[[ 0 10 20 30]  
 [ 0 10 20 30]  
 [ 0 10 20 30]]  
[[ 0  0  0]  
 [10 10 10]  
 [20 20 20]  
 [30 30 30]]  
[0 0 0]  
[10 10 10]  
[20 20 20]  
[30 30 30]  
[0 0 0]
```

```
In [17]: a = np.arange(0,40,10)
```

```
        print a
```

```
        print a.shape
```

```
        a = a[:, np.newaxis]    # adds a new axis -> 2D array
```

```
        print a
```

```
        print a.shape
```

```

b = np.array([1,2,3])
c = a + b
print c
print c.shape

[ 0 10 20 30]
(4,)
[[ 0]
 [10]
 [20]
 [30]]
(4, 1)
[[ 1  2  3]
 [11 12 13]
 [21 22 23]
 [31 32 33]]
(4, 3)

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

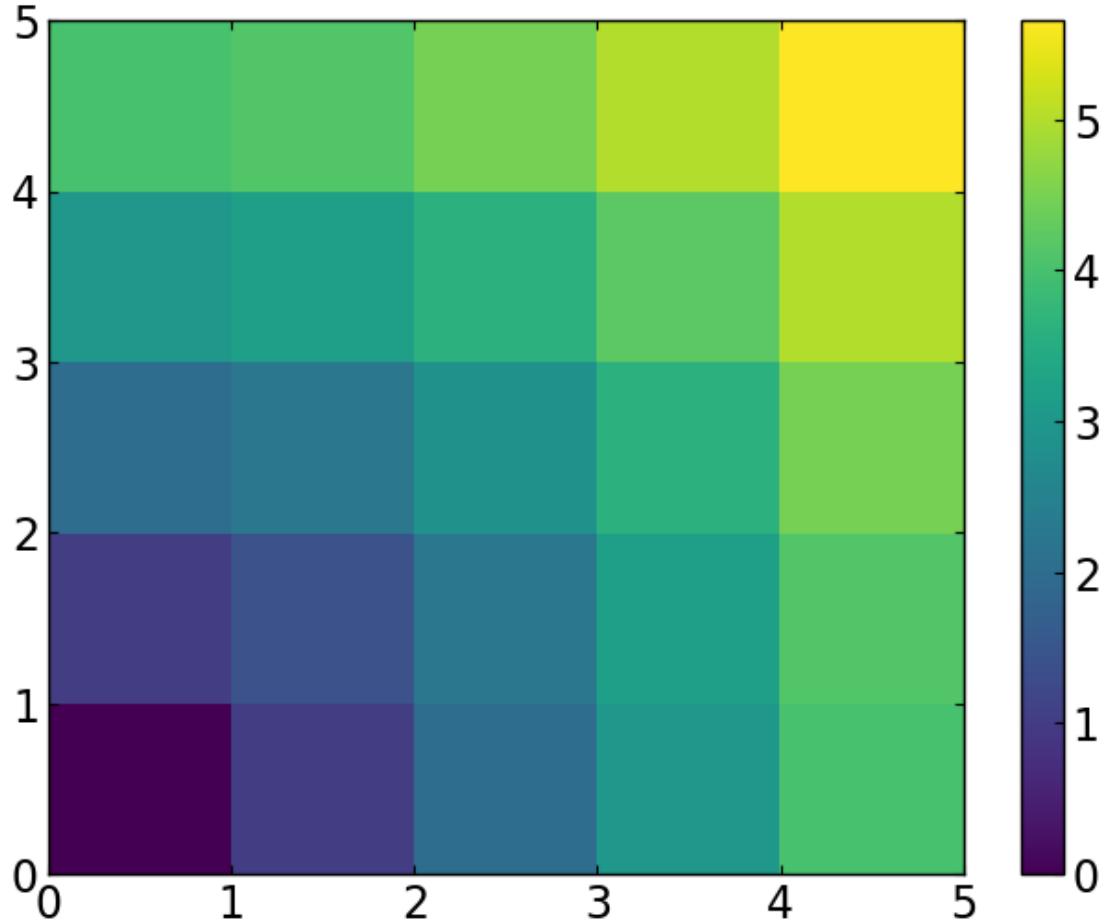
x, y = np.arange(5), np.arange(5)[:, np.newaxis]
print x
print y
distance = np.sqrt(x**2 + y**2)
print distance
print distance.shape

plt.pcolor(distance)
plt.colorbar()

[0 1 2 3 4]
[[0]
 [1]
 [2]
 [3]
 [4]]
[[ 0.          1.          2.          3.          4.        ]
 [ 1.          1.41421356  2.23606798  3.16227766  4.12310563]
 [ 2.          2.23606798  2.82842712  3.60555128  4.47213595]
 [ 3.          3.16227766  3.60555128  4.24264069  5.        ]
 [ 4.          4.12310563  4.47213595  5.          5.65685425]]
(5, 5)

Out[2]: <matplotlib.colorbar.Colorbar at 0x7f5270ce4f10>

```



```
In [19]: x, y = np.ogrid[0:5, 0:5]
print x
print y

print x.shape, y.shape

distance = np.sqrt(x**2 + y**2)
print distance

[[0]
 [1]
 [2]
 [3]
 [4]]
[[0 1 2 3 4]]
(5, 1) (1, 5)
[[ 0.          1.          2.          3.          4.        ]
 [ 1.          1.41421356  2.23606798  3.16227766  4.12310563]]
```

```
[ 2.          2.23606798  2.82842712  3.60555128  4.47213595]
[ 3.          3.16227766  3.60555128  4.24264069  5.          ]
[ 4.          4.12310563  4.47213595  5.          5.65685425]]
```

```
In [20]: x, y = np.mgrid[0:4, 0:4]
```

```
print x
print y
print x.shape, y.shape
```

```
[[0 0 0 0]
 [1 1 1 1]
 [2 2 2 2]
 [3 3 3 3]]
 [[0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]]
 (4, 4) (4, 4)
```

0.1.1 Array shaping

```
In [21]: a = np.array([[1,2,3],[4,5,6]])
```

```
print a
print a.shape
print a.ravel()
print a.T
print a.T.ravel()

b = a.ravel()
b = b.reshape((3,2))
print b
print b.shape

c = a.reshape((3,-1))    # unsapecified (-1) value is inferred
print c
print c.shape
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
[1 2 3 4 5 6]
[[1 4]
 [2 5]
 [3 6]]
[1 4 2 5 3 6]
[[1 2]
 [3 4]]
```

```
[5 6]  
(3, 2)  
[[1 2]  
 [3 4]  
 [5 6]]  
(3, 2)
```

```
In [22]: a.resize?
```

```
In [23]: a = np.arange(4)  
         a.resize((8,))  
         a
```

```
Out[23]: array([0, 1, 2, 3, 0, 0, 0, 0])
```

```
In [24]: a = np.arange(4)  
         print a  
         #b = a  
         c = a.copy()  
         a.resize((8,))  
         print a  
  
         #b.resize((8,))    # referencing array cannot be resized !!  
         #print b  
  
         c.resize((8,))  
         print c
```

```
[0 1 2 3]  
[0 1 2 3 0 0 0 0]  
[0 1 2 3 0 0 0 0]
```

```
In [3]: """ copy 2D to 3D arrays by np.repeat"""  
       import numpy as np
```

```
a = np.array([[1, 2], [3, 4]])  
print "original array"  
print a  
print(a.shape)  
# (2, 2)
```

```
# indexing with np.newaxis inserts a new 3rd dimension, which we then repeat the  
# array along, (you can achieve the same effect by indexing with None, see below)  
b = np.repeat(a[:, :, np.newaxis], 3, axis=2)
```

```
print "extended array"  
print(b.shape)
```

```

# (2, 2, 3)

print(b[:, :, 0])
# [[1 2]
#  [1 2]]

print(b[:, :, 1])
# [[1 2]
#  [1 2]]

print(b[:, :, 2])
# [[1 2]
#  [1 2]]]

original array
[[1 2]
 [3 4]]
(2L, 2L)
extended array
(2L, 2L, 3L)
[[[1 2]
 [3 4]]
 [[1 2]
 [3 4]]
 [[1 2]
 [3 4]]]

```

0.1.2 Sort

```
In [25]: a = np.array([[4,3,5],[1,2,1]])
b = np.sort(a, axis=1)
```

```

print b

# in-place sort
a.sort(axis=1)
print a
```

```
[[3 4 5]
 [1 1 2]]
[[3 4 5]
 [1 1 2]]
```

```
In [26]: a = np.array([4.,3.,1.,2.])
j = np.argsort(a)
print a
```

```
print j
print a[j]

[ 4.  3.  1.  2.]
[2 3 1 0]
[ 1.  2.  3.  4.]
```

```
In [27]: a = np.array([4.,3.,1.,2.])
j_max = np.argmax(a)
j_min = np.argmin(a)
```

```
print j_max, j_min
print a[j_max], a[j_min]
```

```
0 2
4.0 1.0
```

0.1.3 Data type

```
In [28]: a = np.array([1,2,3,4])
print type(a)
print a.dtype

b = np.array([1.,2.,3.,4.])
print type(b)
print b.dtype

c = np.array([1,2,3,4], dtype=float)
print type(c)
print c.dtype
```

```
<type 'numpy.ndarray'>
int64
<type 'numpy.ndarray'>
float64
<type 'numpy.ndarray'>
float64
```

```
In [29]: d = np.array([1+2j, 3+4j, 5+6j])
print d.dtype

e = np.array([True, False, False, True])
print e.dtype

f = np.array(['Bonjour', 'Hello', 'Hallo'])
print f.dtype
```

```
complex128
bool
|S7
```

```
In [30]: a = np.array([1.7,1.2,1.6])
      b = a.astype(int)    # truncate to integer
      print b
```

```
[1 1 1]
```

```
In [31]: a = np.array([1.2,1.5,1.6,2.5,3.5,4.5])
      b = np.around(a)
      print b

      c = np.around(a).astype(int)
      print c
```

```
[ 1.  2.  2.  2.  4.  4.]
[1 2 2 2 4 4]
```

```
In [32]: """ Integer
           int8, int16, int32, int64 (same as int on 64-bit platform) """

           print np.array([1],dtype=int).dtype
           print np.iinfo(np.int32).max, 2**31 - 1
           print np.iinfo(np.int64).max, 2**63 - 1
```

```
int64
2147483647 2147483647
9223372036854775807 9223372036854775807
```

```
In [33]: """ Unsigned Integer
           uint8, uint16, uint32, uint64 """

           print np.iinfo(np.uint32).max, 2**32 - 1
           print np.iinfo(np.uint64).max, 2**64 - 1
```

```
4294967295 4294967295
18446744073709551615 18446744073709551615
```

```
In [34]: """ Float
           float16, float32, float64 (same as float),
           float96, float128 (same as np.longdouble) """


```

```

print np.finfo(np.float32).eps # The smallest representable positive number
print np.finfo(np.float64).eps # same as float

print np.float32(1e-8) + np.float32(1) == 1
print np.float64(1e-8) + np.float64(1) == 1

1.19209e-07
2.22044604925e-16
True
False

```

0.1.4 Structured Data Type

In [4]: *""" Structured data type """*

```

samples = np.zeros((6,), dtype=[('sensor_code','S4'),
                               ('position',float),('value',float)])
print samples.ndim
print samples.shape
print samples.dtype
print samples.dtype.names

print samples
samples[:] = [('ALFA', 1, 0.37), ('BETA', 1, 0.11), ('TAU', 1, 0.13),
              ('ALFA', 1.5, 0.37), ('ALFA', 3, 0.11), ('TAU', 1.2, 0.13)]
print samples
print samples['sensor_code']
print samples['sensor_code'][1]
print samples[0]
print samples['position'].dtype
print samples[['position', 'value']]
print samples[samples['sensor_code'] == 'ALFA']
print samples['position'][samples['sensor_code'] == 'ALFA']

1
(6,)
[('sensor_code', 'S4'), ('position', '<f8'), ('value', '<f8')]
('sensor_code', 'position', 'value')
[(', 0.0, 0.0) (', 0.0, 0.0) (', 0.0, 0.0) (', 0.0, 0.0) (', 0.0, 0.0)
 (', 0.0, 0.0)]
[('ALFA', 1.0, 0.37) ('BETA', 1.0, 0.11) ('TAU', 1.0, 0.13)
 ('ALFA', 1.5, 0.37) ('ALFA', 3.0, 0.11) ('TAU', 1.2, 0.13)]
['ALFA' 'BETA' 'TAU' 'ALFA' 'ALFA' 'TAU']
BETA
('ALFA', 1.0, 0.37)
float64
[(1.0, 0.37) (1.0, 0.11) (1.0, 0.13) (1.5, 0.37) (3.0, 0.11) (1.2, 0.13)]

```

```
[('ALFA', 1.0, 0.37) ('ALFA', 1.5, 0.37) ('ALFA', 3.0, 0.11)]  
[ 1. 1.5 3. ]
```

```
In [11]: x = np.array([(1,2.,'Hello'), (2,3.,"World")],dtype=[('foo', 'i4'),('bar', 'f4'), ('ba
```

```
x
```

```
Out[11]: array([(1, 2.0, 'Hello'), (2, 3.0, 'World')],  
                 dtype=[('foo', '<i4'), ('bar', '<f4'), ('baz', 'S10')])
```

```
In [12]: """ Structured Arrays  
          1) String argument  
          """
```

```
x = np.zeros(3, dtype='3int8, float32, (2,3)float64')  
x
```

```
Out[12]: array([[0, 0, 0], 0.0, [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]],  
                [[0, 0, 0], 0.0, [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]],  
                [[0, 0, 0], 0.0, [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]]],  
                dtype=[('f0', 'i1', (3,)), ('f1', '<f4'), ('f2', '<f8', (2, 3))])
```

```
In [13]: """ Structured Arrays  
          2) Tuple argument  
          """
```

```
x = np.zeros(3, dtype=('i4',[('r','u1'), ('g','u1'), ('b','u1'), ('a','u1')]))  
x
```

```
Out[13]: array([0, 0, 0], dtype=int32)
```

```
In [14]: """ Structured Arrays  
          3) List argument  
          """
```

```
x = np.zeros(3, dtype=[('x','f4'),('y',np.float32),('value','f4',(2,2))])  
x
```

```
Out[14]: array([(0.0, 0.0, [[0.0, 0.0], [0.0, 0.0]]),  
                  (0.0, 0.0, [[0.0, 0.0], [0.0, 0.0]]),  
                  (0.0, 0.0, [[0.0, 0.0], [0.0, 0.0]])],  
                  dtype=[('x', '<f4'), ('y', '<f4'), ('value', '<f4', (2, 2))])
```

```
In [16]: """ Structured Arrays  
          4) dictionary argument  
          """
```

```
x = np.zeros(3, dtype={'names':['col1', 'col2'], 'formats':['i4','f4']})  
x
```

```
Out[16]: array([(0, 0.0), (0, 0.0), (0, 0.0)],  
                 dtype=[('col1', '<i4'), ('col2', '<f4')])
```

0.2 Maskedarray (Missing Data)

```
In [36]: ?np.ma
```

```
In [37]: x = np.ma.array([1,2,3,4], mask=[0,1,0,1])
x
```

```
Out[37]: masked_array(data = [1 -- 3 --],
                       mask = [False  True False  True],
                       fill_value = 999999)
```

```
In [38]: y = np.ma.array([1,2,3,4], mask=[0,1,1,1])
x+y
```

```
Out[38]: masked_array(data = [2 -- -- --],
                       mask = [False  True  True  True],
                       fill_value = 999999)
```

```
In [39]: np.sqrt([1,-1,2,-2])
```

```
Out[39]: array([ 1. , nan,  1.41421356, nan])
```

```
In [40]: np.ma.sqrt([1,-1,2,-2])
```

```
Out[40]: masked_array(data = [1.0 -- 1.4142135623730951 --],
                       mask = [False  True False  True],
                       fill_value = 1e+20)
```

```
In [41]: x = np.array([2, 1, 3, np.nan, 5, 2, 3, np.nan])
print np.mean(x)
```

```
m = np.ma.masked_array(x, np.isnan(x))
print m
print np.mean(m), (2.+1.+3.+5.+2.+3.)/6.
```

```
nan
[2.0 1.0 3.0 -- 5.0 2.0 3.0 --]
2.666666666667 2.666666666667
```

0.2.1 Copies & Views

```
In [42]: a = np.arange(10)
print a
```

```
b = a[::2]
print b

print np.may_share_memory(a,b)
```

```

b[0] = 12
print b
print a

c = a[::-2].copy() # not share the memory. Copy of the array
print np.may_share_memory(a,c)
c[0] = 1
print c
print a

[0 1 2 3 4 5 6 7 8 9]
[0 2 4 6 8]
True
[12 2 4 6 8]
[12 1 2 3 4 5 6 7 8 9]
False
[1 2 4 6 8]
[12 1 2 3 4 5 6 7 8 9]

```

In [43]: `np.random.seed(3)`
`a = np.random.random_integers(0,20,15)`

```

print a

mask = (a % 3 == 0)
print mask

b = a[mask]
print b

c = a.copy()
c[mask] = -1
print c

[10 3 8 0 19 10 11 9 10 6 0 20 12 7 14]
[False True False True False False False True False True True False
 True False False]
[ 3 0 9 6 0 12]
[10 -1 8 -1 19 10 11 -1 10 -1 -1 20 -1 7 14]
```

/home/astrodoo/anaconda2/lib/python2.7/site-packages/ipykernel/_main_.py:2: DeprecationWarning:
from ipykernel import kernelapp as app

In [44]: `a = np.arange(0,100,10)`
`print a`

```

b = a[[2,3,2,4,2]] #note: [2,3,2,4,2] is a Python list
print b

c = a[[9,7]]
print c

[ 0 10 20 30 40 50 60 70 80 90]
[20 30 20 40 20]
[90 70]

```

0.3 Operator

```

In [45]: a = np.ones((3,3))
          print a

          b = a*a      # NOT matrix multiplication
          print b

          c = a.dot(a) # matrix multiplication
          print c

[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
[[ 3.  3.  3.]
 [ 3.  3.  3.]
 [ 3.  3.  3.]]

```

```

In [46]: a = np.array([1,2,3,4])
          b = np.array([4,2,2,4])

          print a == b
          print a > b

          c = np.array([1,2,3,4])
          print np.array_equal(a,b)
          print np.array_equal(a,c)

[False  True False  True]
[False False  True False]
False
True

```

```
In [47]: a = np.array([1,1,0,0], dtype=bool)
b = np.array([1,0,1,0], dtype=bool)

print np.logical_or(a,b)

print np.logical_and(a,b)

print np.all([True, True, False])
print np.any([True, True, False])

c = np.zeros((100,100))
print np.any(c != 0)
print np.all(c == c)

d = np.array([1,2,3,2])
e = np.array([2,2,3,2])
f = np.array([6,4,4,5])
print ((d <= e) & (e <=f)).all()

[ True  True  True False]
[ True False False False]
False
True
False
True
True
```

```
In [48]: x = np.array([1,2,3,4])
print np.sum(x)

print x.sum()

x = np.array([[1,1],[2,2],[3,3]])
print x
print x.sum(axis=0)
print x[:,0].sum(), x[:,1].sum()
print x.sum(axis=1)

x = np.random.rand(2,2,2)
print x.sum(axis=2)[0,1]
print x[0,1,:].sum()

10
10
[[1 1]
 [2 2]
 [3 3]]
```

```
[6 6]
6 6
[2 4 6]
1.2671177194
1.2671177194
```

```
In [49]: x = np.array([1,3,2])
    print x.min()
    print x.argmin()      # index of minimum
    print x.max()
    print x.argmax()      # index of maximum
```

```
1
0
3
1
```

```
In [50]: x = np.array([1,2,3,1])
y = np.array([[1,2,3],[5,6,1]])
```

```
print x.mean()
print np.median(x)
print np.median(y, axis=-1)
print x.std()
```

```
1.75
1.5
[ 2.  5.]
0.829156197589
```

0.3.1 Polynomials

```
In [51]: # 3x^2 + 2x -1:
```

```
p = np.poly1d([3,2,-1])
print p
print p(3)

print p.roots
print p.order

2
3 x + 2 x - 1
32
[-1.          0.33333333]
2
```

```
In [52]: import matplotlib.pyplot as plt
import numpy as np

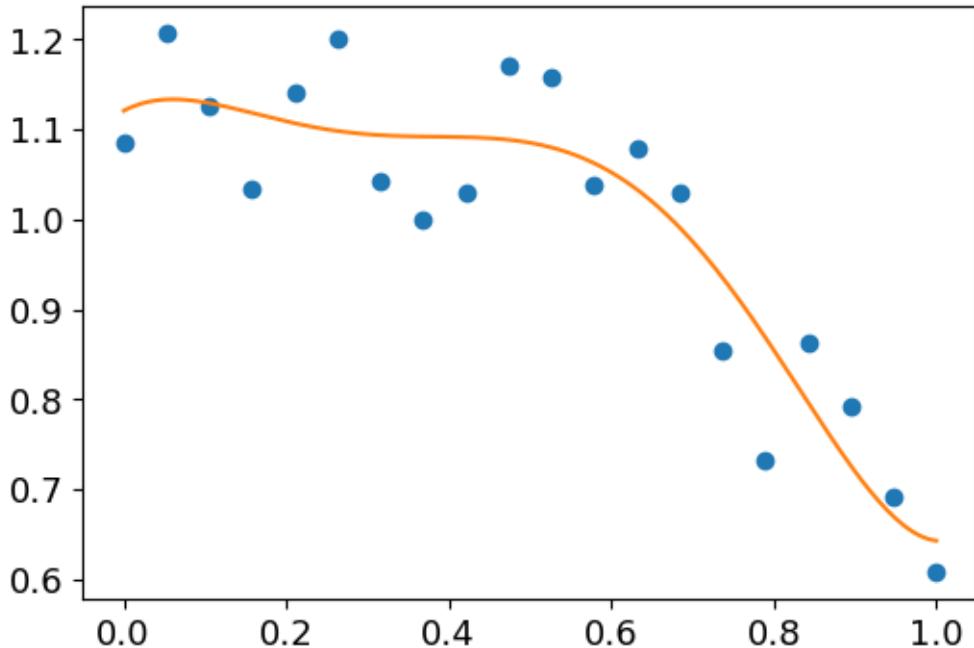
x = np.linspace(0,1,20)
y = np.cos(x) + 0.3*np.random.rand(20)
p = np.poly1d(np.polyfit(x,y,5))

print p

t = np.linspace(0,1,200)
plt.plot(x,y,'o', t,p(t),'-')

5      4      3      2
10.38 x - 23.86 x + 17.9 x - 5.374 x + 0.4737 x + 1.121
```

```
Out[52]: [,
<matplotlib.lines.Line2D at 0x7f4170e55a50>]
```



```
In [53]: # 3x^2 + 2x -1:

p = np.polynomial.Polynomial([-1,2,3])    # coeffs in different order!
print p

print p.roots()
print p.degree    # No 'order' for general polynomials
```

```

poly([-1.  2.  3.])
[-1.          0.33333333]
<bound method Polynomial.degree of Polynomial([-1.,  2.,  3.], [-1,  1], [-1,  1])>

In [54]: import matplotlib.pyplot as plt
         import numpy as np

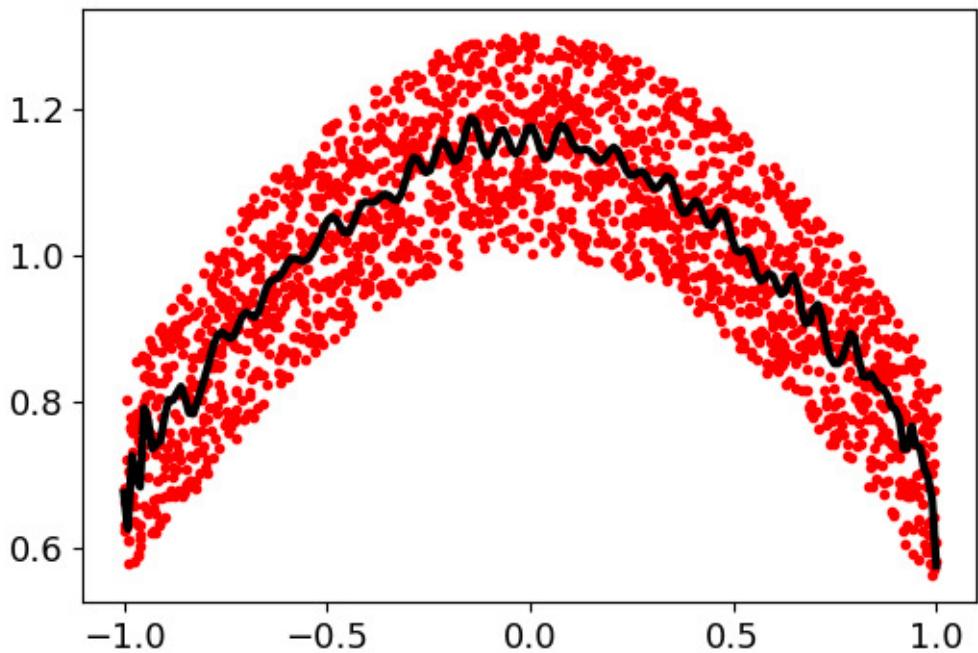
         x = np.linspace(-1,1,2000)
         y = np.cos(x) + 0.3*np.random.rand(2000)
         p = np.polynomial.Chebyshev.fit(x,y,90)
         print p

         t = np.linspace(-1,1,200)
         plt.plot(x,y,'r.')
         plt.plot(t,p(t),'k-',lw=3)

cheb([ 9.14043761e-01   2.19222458e-03  -2.35169376e-01   3.20701401e-03
      2.76701731e-03  -2.09022617e-03  -8.67722149e-03  -5.18024682e-04
     -2.17178977e-03   2.36010868e-03  -1.95651621e-03   6.34399746e-03
      6.07882923e-04   1.03595466e-03   3.19772166e-03   8.78653016e-04
     1.56587751e-03   8.00870944e-04   4.12908733e-03   1.04561426e-03
     1.48051061e-03   6.18256714e-03   4.34691664e-03   4.43694970e-03
     3.20068400e-03   5.24654701e-04   1.40123706e-03  -1.23420448e-03
     1.09937361e-03   3.73652371e-03  -1.51325787e-03   2.99222408e-03
    -5.38920509e-03  -1.43374846e-03  -8.77803366e-04  -4.98137092e-03
     3.93201308e-03  -3.73722903e-03   3.34900302e-03  -1.68506051e-03
     5.26052685e-04   9.65610689e-04  -2.83980221e-03  -8.60446568e-04
    -7.14496563e-03   3.07402327e-04  -5.63295567e-03   1.71446534e-03
    -2.33432719e-03  -1.30654201e-03  -4.83882538e-03   1.68722125e-03
    -4.62057216e-03  -6.26133776e-03  -5.00293728e-04  -3.89766461e-03
     2.79494010e-03  -3.72870959e-03   2.32839472e-03  -5.89208263e-03
    -2.19352302e-06  -8.28727483e-03  -4.62325296e-03  -6.04706450e-03
    -9.48890484e-04  -4.87012036e-03  -9.22462000e-04  -3.68215847e-03
    -3.70745680e-03  -4.83580228e-03  -2.40725576e-03  -5.49768976e-03
    -9.52885461e-05  -3.34824453e-03  -2.15170664e-03  -4.49157642e-03
    -3.16356793e-03  -3.26949511e-03  -3.59127431e-03  -7.46391304e-03
    -1.63320051e-03  -1.57746562e-03  -4.87803604e-03  -3.37295434e-03
    -3.67855996e-03  -6.37870783e-04  -6.83287932e-03   3.63541826e-03
     2.93594457e-03   1.39682469e-03  -6.64899738e-03])

```

Out[54]: [`<matplotlib.lines.Line2D at 0x7f41716733d0>`]



0.4 Input & Output

In [55]: !cat ./data/colms.dat

```
1 5.23    2345.  
3 3453.   212.  
5 23.     2345.  
6 234.    345.  
7 23.     234.
```

In [56]: data = np.loadtxt('~/data/colms.dat')

```
print data  
print data.dtype  
a,b,c = data.T      # Trick (transpose: columns to variables)  
  
print a  
print b  
print c
```

```
[[ 1.0000000e+00  5.2300000e+00  2.3450000e+03]
 [ 3.0000000e+00  3.4530000e+03  2.1200000e+02]
 [ 5.0000000e+00  2.3000000e+01  2.3450000e+03]
 [ 6.0000000e+00  2.3400000e+02  3.4500000e+02]
 [ 7.0000000e+00  2.3000000e+01  2.3400000e+02]]
float64
[ 1.  3.  5.  6.  7.]
[ 5.23 3453.     23.     234.     23.   ]
[ 2345. 212. 2345. 345. 234.]
```

In [57]: !cat ./data/populations.txt

```
# year    hare      lynx      carrot

1900    30e3     4e3      48300
1901    47.2e3   6.1e3     48200
1902    70.2e3   9.8e3     41500
1902    77.4e3   35.2e3    38200
```

In [58]: data = np.loadtxt('../data/populations.txt')
data

Out[58]: array([[1900., 30000., 4000., 48300.],
 [1901., 47200., 6100., 48200.],
 [1902., 70200., 9800., 41500.],
 [1902., 77400., 35200., 38200.]])

```
In [59]: np.savetxt('../data/pop2.txt',data)
data2 = np.loadtxt('../data/pop2.txt')
print data2
print data2.dtype
print data2.shape
print len(data2)
```

```
[[ 1900. 30000. 4000. 48300.]
 [ 1901. 47200. 6100. 48200.]
 [ 1902. 70200. 9800. 41500.]
 [ 1902. 77400. 35200. 38200.]]
float64
(4, 4)
4
```

```
In [60]: """ Numpy's own format """
```

```
data = np.ones((3,3))
np.save('../data/pop.npy', data)

data3 = np.load('../data/pop.npy')
print data3

[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
```

```
In [1]: """ Save Multiple variables to one file"""
```

```
import numpy as np

d1 = np.ones((3,3))
d2 = np.linspace(0.,10.,5)
d3 = {'a':1., 'b':3.}

# Since 'savez' is basically for saving arrays, the dictionary should be saved as an array
# and restore with appened [0].
np.savez('../data/pop4.npz', d1=d1, d2=d2,d3=[d3])

#load_data = np.load('../data/pop4.npz')
#print type(load_data)

In [2]: load_data = np.load('../data/pop4.npz')
print type(load_data)

<class 'numpy.lib.npyio.NpzFile'>
```

```
In [3]: print load_data.files
```

```
['d2', 'd3', 'd1']
```

```
In [4]: load_d1 = load_data['d1']
load_d2 = load_data['d2']
load_d3 = load_data['d3'][0]

print d1
print d2
print d3
print d3['a']
```

```
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
```

```
[ 1.  1.  1.]]  
[ 0.    2.5   5.    7.5  10. ]  
{'a': 1.0, 'b': 3.0}  
1.0
```

```
In [9]: print d1.dtype, d2.dtype, type(d3)  
float64 float64 <type 'dict'>
```

```
In [9]: import numpy as np  
  
a = np.array([1.,2.,3.])  
  
print a  
  
a = np.delete(a,0)  
  
print a  
[ 1.  2.  3.]  
[ 2.  3.]
```

```
In [11]: a = np.array([1.,2.,3.])  
  
a[1] = np.nan  
  
print a  
[ 1.  nan  3.]
```

```
In [18]: print a[~np.isnan(a)]  
[ 1.  3.]
```

```
In [17]: a = a[~np.isnan(a)]
```

```
ValueError
```

```
Traceback (most recent call last)
```

```
<ipython-input-17-dcd346cc8060> in <module>()  
----> 1 print a[not(np.isnan(a))].all()
```

```
ValueError: The truth value of an array with more than one element is ambiguous. Use a
```

```
In [15]: a=0.1
      b = [1,2,3]
      print 'aaaa %f' %a
      print 'printing list [%s]' % ', '.join('%i' % val for val in b)

aaaa 0.100000
printing list [1, 2, 3]
```

Scipy1

January 30, 2019

In [1]: %pylab inline

Populating the interactive namespace from numpy and matplotlib

In [2]: import scipy as sp
p = sp.poly1d([3,4,5])
print p
print p*p
print p.deriv()
print p.integ(k=6)
print p([4,5])

```
2  
3 x + 4 x + 5  
4      3      2  
9 x + 24 x + 46 x + 40 x + 25  
  
6 x + 4  
3      2  
1 x + 2 x + 5 x + 6  
[ 69 100]
```

1 Linear Algebra

In [3]: from scipy import linalg
import numpy as np

arr = np.array([[1,2],[3,4]])
print arr

determinant of a square matrix
print linalg.det(arr)

inverse of a square matrix
iarr = linalg.inv(arr)

```

print iarr

# check if it is correct
print np.allclose(np.dot(arr,iarr), np.eye(2))

[[1 2]
 [3 4]]
-2.0
[[-2.  1. ]
 [ 1.5 -0.5]]
True

```

2 Fast Fourier Transforms

```

In [4]: #import scipy as sp
         from scipy import fftpack
         import numpy as np
         import matplotlib.pyplot as plt

time_step = 0.02
period = 5.
time_vec = np.arange(0., 20., time_step)
sig = np.sin(2.*np.pi / period * time_vec) + \
      0.5 * np.random.randn(time_vec.size)

sample_freq = fftpack.fftfreq(sig.size, d=time_step)
sig_fft = fftpack.fft(sig)

pidxs = np.where(sample_freq > 0)
freqs = sample_freq[pidxs]
power = np.abs(sig_fft)[pidxs]

freq = freqs[power.argmax()]
print np.allclose(freq, 1./period)

sig_fft[np.abs(sample_freq) > freq] = 0

main_sig = fftpack.ifft(sig_fft)

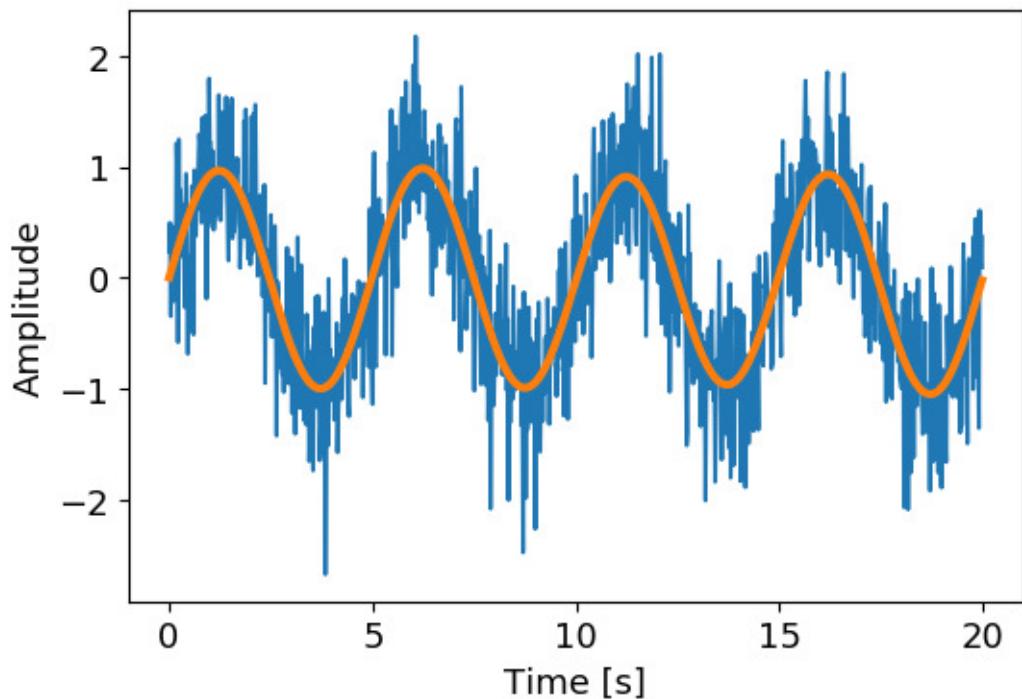
# draw plot
plt.figure()
plt.plot(time_vec,sig)
plt.plot(time_vec, main_sig, linewidth=3)
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')

True

```

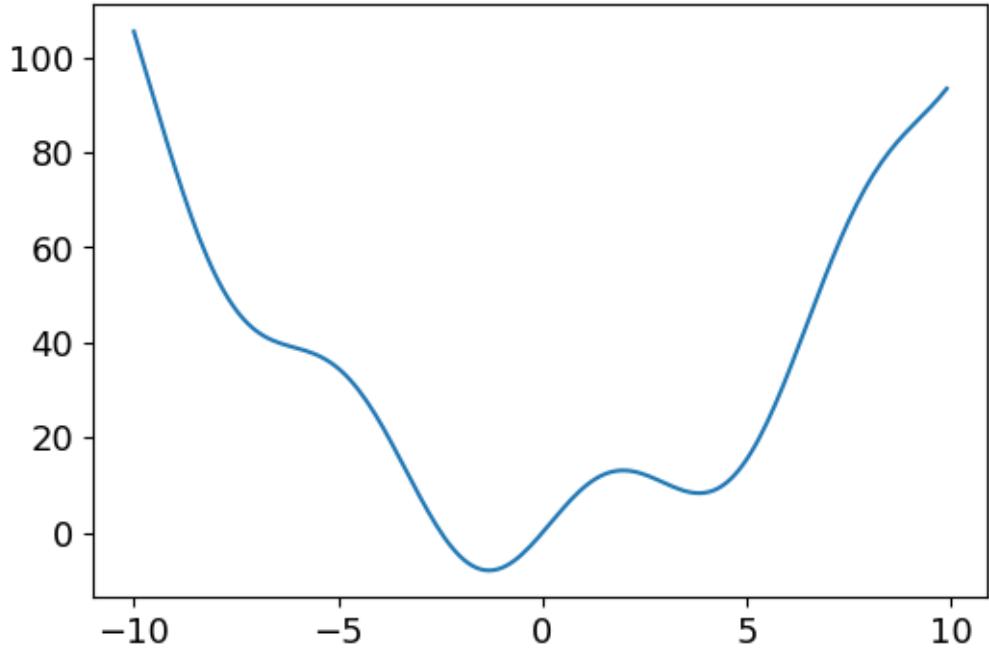
```
/home/astrodoo/anaconda2/lib/python2.7/site-packages/numpy/core/numeric.py:482: ComplexWarning  
    return array(a, dtype, copy=False, order=order)
```

Out [4]: <matplotlib.text.Text at 0x7fb0bca7ddd0>



3 Optimize

```
In [5]: from scipy import optimize  
import matplotlib.pyplot as plt  
  
def f(x):  
    return x**2 + 10.*np.sin(x)  
  
x = np.arange(-10,10,0.1)  
plt.plot(x, f(x))  
plt.show()
```



In [6]: *"^{'''} Finding a global minimum point ^{'''}"*

```
# optimize.fmin_bfgs(f,0)
# this may find a local minimum rather than global minimum
print optimize.fmin_bfgs(f,3)

# finding a global minimum

grid = (-10,10,0.1)      # same as x in tuple
xmin_global = optimize.brute(f,(grid,))
print xmin_global

# For larger grid sizes scipy.optimize.brute() becomes quite
# slow. scipy.optimize.anneal() provides an alternative, using
# simulated annealing.
```

"^{'''} Finding a local minimum point ^{'''}"

```
xmin_local = optimize.fminbound(f, 0, 10)
print xmin_local
```

"^{'''} Finding a root ^{'''}"

```
root = optimize.fsolve(f, 1)    # initial guess is 1
print root      # only one root is found.
```

```

root2 = optimize.fsolve(f, -2.5) # different initial guess
print root2

""" Curve fitting """
xdata = np.linspace(-10,10, num=20)
ydata = f(xdata) + np.random.randn(xdata.size)

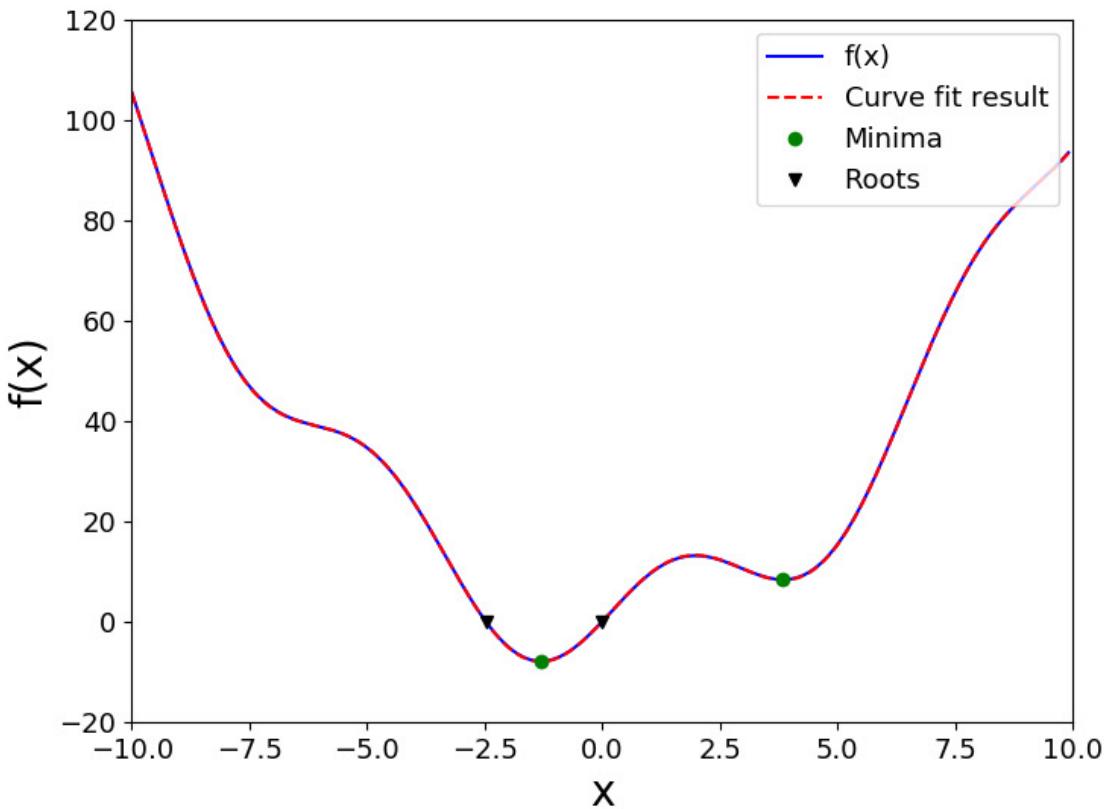
def f2(x,a,b):
    return a*x**2 + b*np.sin(x)

guess = [2,2]
params, params_covariance = optimize.curve_fit(f2, xdata, ydata, guess)
print params

""" Draw """
plt.figure(figsize=(8,6))
plt.plot(x, f(x), 'b-',label='f(x)')
plt.plot(x, f2(x,params[0],params[1]),'r--',label='Curve fit result')
plt.plot([xmin_global,xmin_local],[f(xmin_global),f(xmin_local)], \
         'go',label='Minima')
#plt.scatter([root],[0],color='green',s=50, label='Roots')
plt.plot([root,root2],[0,0],'kv',label='Roots')
plt.xlim(-10, 10)
plt.ylim(-20, 120)
plt.xlabel('x', fontsize=20)
plt.ylabel('f(x)', fontsize=20)
plt.legend(loc='upper right')
plt.show()

Optimization terminated successfully.
    Current function value: 8.315586
    Iterations: 6
    Function evaluations: 21
    Gradient evaluations: 7
[ 3.83746709]
[-1.30641113]
3.8374671195
[ 0.]
[-2.47948183]
[ 0.99912857  10.08807092]

```



3.0.1 Interpolation

```
In [7]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy.interpolate import interp1d

measured_time = np.linspace(0,1,10)
noise = (np.random.random(10)*2 - 1) * 1e-1
measures = np.sin(2.*np.pi*measured_time) + noise

linear_interp = interp1d(measured_time, measures)

computed_time = np.linspace(0,1,50)
linear_results = linear_interp(computed_time)

cubic_interp = interp1d(measured_time, measures, kind='cubic')
cubic_results = cubic_interp(computed_time)

""" Draw """

```

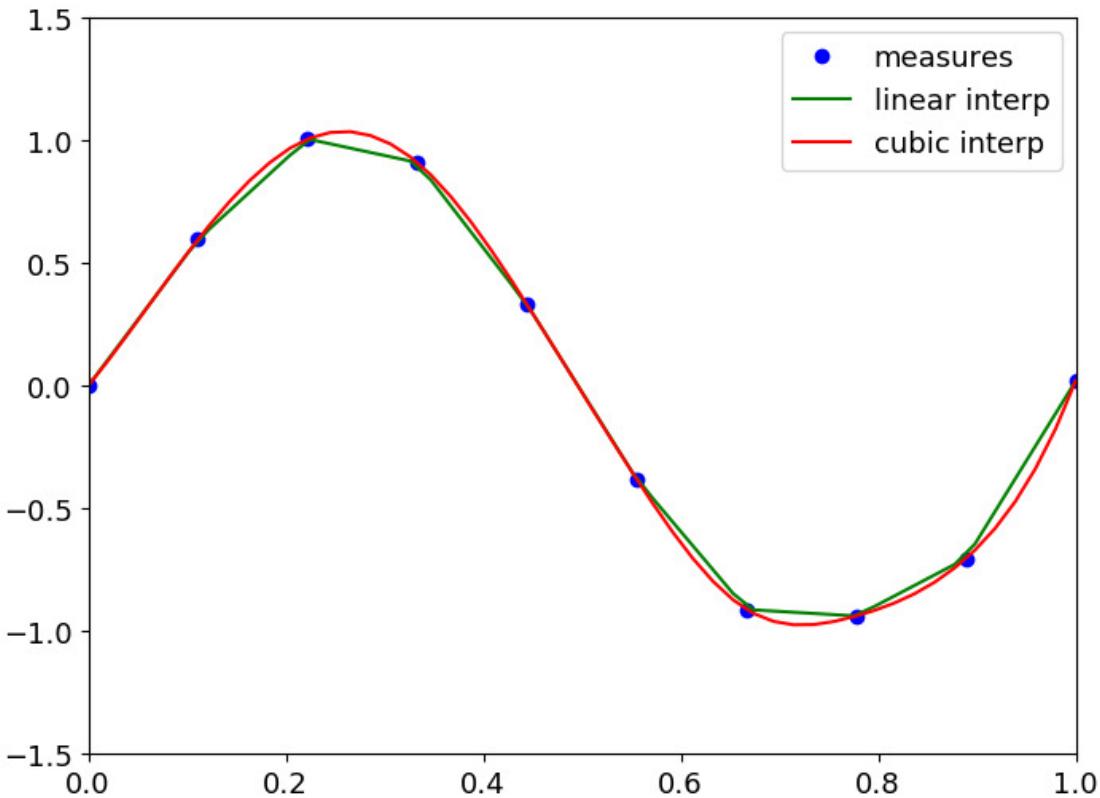
```

plt.figure(figsize=(8,6))

plt.plot(measured_time, measures, 'bo', label='measures')
plt.xlim(0.,1.)
plt.ylim(-1.5,1.5)
plt.plot(computed_time, linear_results, 'g-', label='linear interp')
plt.plot(computed_time, cubic_results, 'r-', label='cubic interp')
plt.legend(loc='upper right')

plt.show()

```



In [8]: %matplotlib inline

```

""" interpolate 2d """

import numpy as np
from scipy.interpolate import griddata
import matplotlib.pyplot as plt

x = np.linspace(-1,1,100)
y = np.linspace(-1,1,100)

```

```

X, Y = np.meshgrid(x,y)

def f(x, y):
    s = np.hypot(x, y)
    phi = np.arctan2(y, x)
    tau = s + s*(1-s)/5 * np.sin(6*phi)
    return 5*(1-tau) + tau

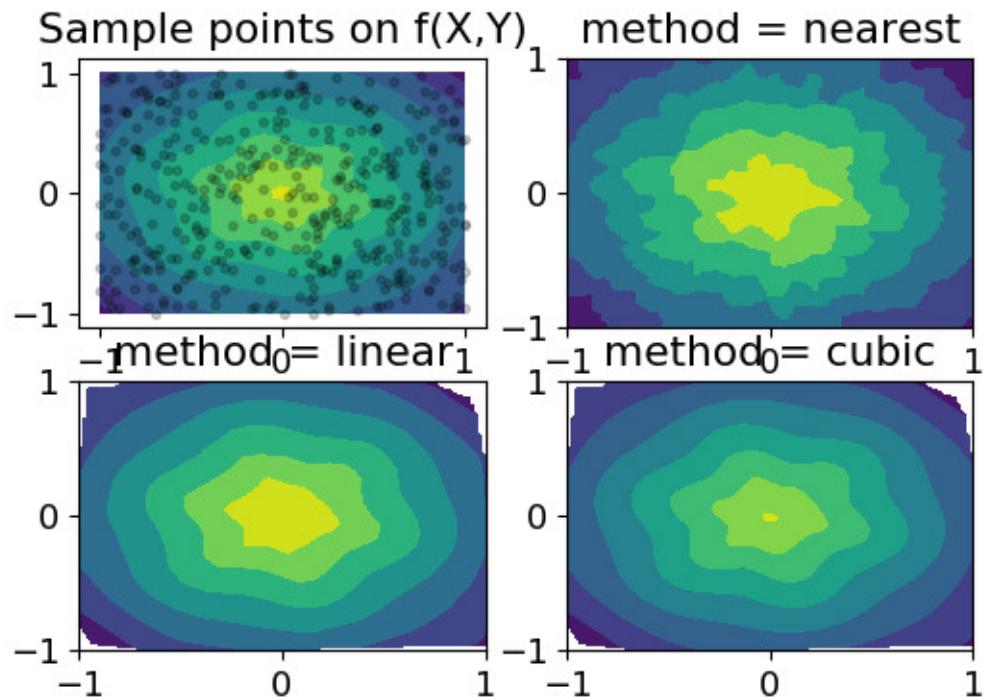
T = f(X, Y)
# Choose npts random point from the discrete domain of our model function
npts = 400
px, py = np.random.choice(x, npts), np.random.choice(y, npts)

fig, ax = plt.subplots(nrows=2, ncols=2)
# Plot the model function and the randomly selected sample points
ax[0,0].contourf(X, Y, T)
ax[0,0].scatter(px, py, c='k', alpha=0.2, marker='.')
ax[0,0].set_title('Sample points on f(X,Y)')

# Interpolate using three different methods and plot
for i, method in enumerate(('nearest', 'linear', 'cubic')):
    Ti = griddata((px, py), f(px,py), (X, Y), method=method)
    r, c = (i+1) // 2, (i+1) % 2
    ax[r,c].contourf(X, Y, Ti)
    ax[r,c].set_title('method = {}'.format(method))

plt.show()

```



3.0.2 integrate

```
In [9]: from scipy.integrate import quad
```

```
res, err = quad(np.sin, 0, np.pi/2)
print np.allclose(res,1)
print np.allclose(err, 1-res)
```

```
True
```

```
True
```

```
In [10]: """ first-order ODE equations
```

```
dy/dt = rhs(y1,y2,...,t0,...) """
```

```
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt
```

```
# dy/dt = -2y (t=0 to 4) & initial y(t=0) = 1
def calc_derivative(ypos, time, counter_arr):
    counter_arr += 1
    return -2*ypos
```

```
counter = np.zeros((1,), dtype=np.uint16)
print counter
```

```
time_vec = np.linspace(0,4,40)
yvec, info = odeint(calc_derivative, 1, time_vec,
                     args=(counter,), full_output=True)
print counter
```

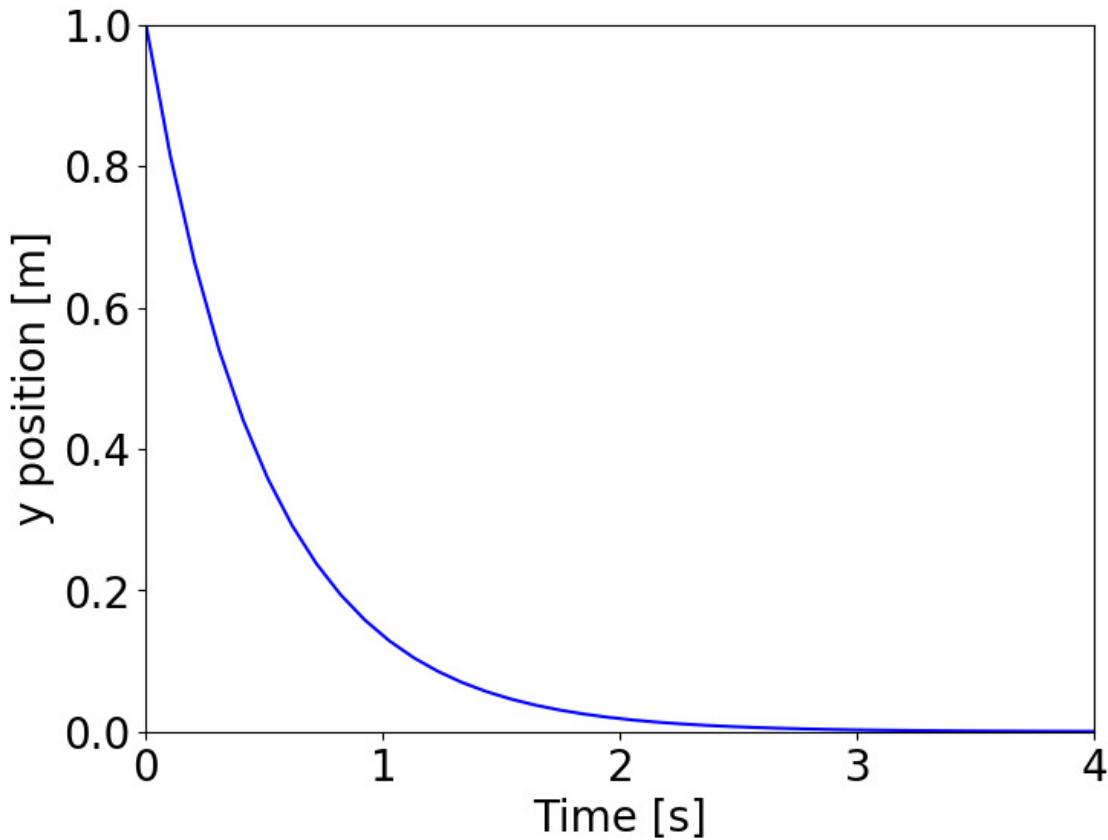
```
""" Draw """
plt.figure(figsize=(8,6))
```

```
#plt.figtext(fontsize=30)
```

```
plt.plot(time_vec, yvec, 'b-')
plt.tick_params(labelsize=20)
plt.xlabel('Time [s]', fontsize=20)
plt.ylabel('y position [m]', fontsize=20)
plt.xlim(0.,4.)
plt.ylim(0.,1.)
```

```
plt.show()
```

```
[0]  
[129]
```



```
In [11]: """ Second-order ODE equations  
y'' + 2 eps wo y' + wo^2 y = 0  
wo^2 = k/m  
eps = c/(2 m wo) """  
  
from scipy.integrate import odeint  
import numpy as np  
import matplotlib.pyplot as plt  
  
m = 0.5  
k = 4  
c = 0.4  
  
eps = c / (2.*m*np.sqrt(k/m))  
  
# coeff: nu = 2*eps*wo = c/m  
# om = wo^2 = k/m
```

```

nu_coef = c/m
om_coef = k/m

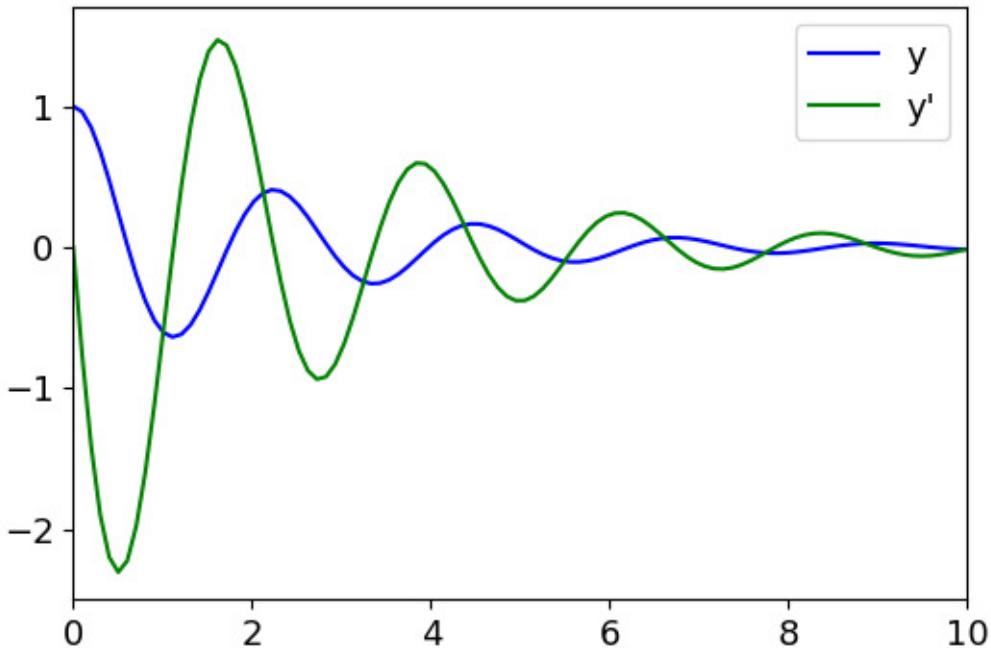
def calc_der(yvec, time, nuc, omc):
    return (yvec[1], -nuc*yvec[1] - omc*yvec[0])

time_vec = np.linspace(0,10,100)
yarr = odeint(calc_der, (1,0), time_vec, args=(nu_coef,om_coef))

""" Draw """
plt.plot(time_vec, yarr[:,0], 'b-',label='y')
plt.plot(time_vec, yarr[:,1], 'g-',label="y'")

plt.xlim(0.,10.)
plt.ylim(-2.5,1.7)
plt.legend(loc='upper right')
plt.show()

```



3.1 Fitting

```
In [6]: import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
%matplotlib inline
```

```

def func(x, a, b, c):
    return a * np.exp(-b * x) + c

# define the data to be fit with some noise

xdata = np.linspace(0, 4, 100)
y = func(xdata, 2.5, 1.3, 0.5)
y_noise = 0.2 * np.random.normal(size=xdata.size)
ydata = y + y_noise

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(xdata, ydata, 'bo', label='data')

# Fit for the parameters a, b, c of the function `func`

popt, pcov = curve_fit(func, xdata, ydata)
ax.plot(xdata, func(xdata, *popt), 'r-', label='fit', linewidth=3)

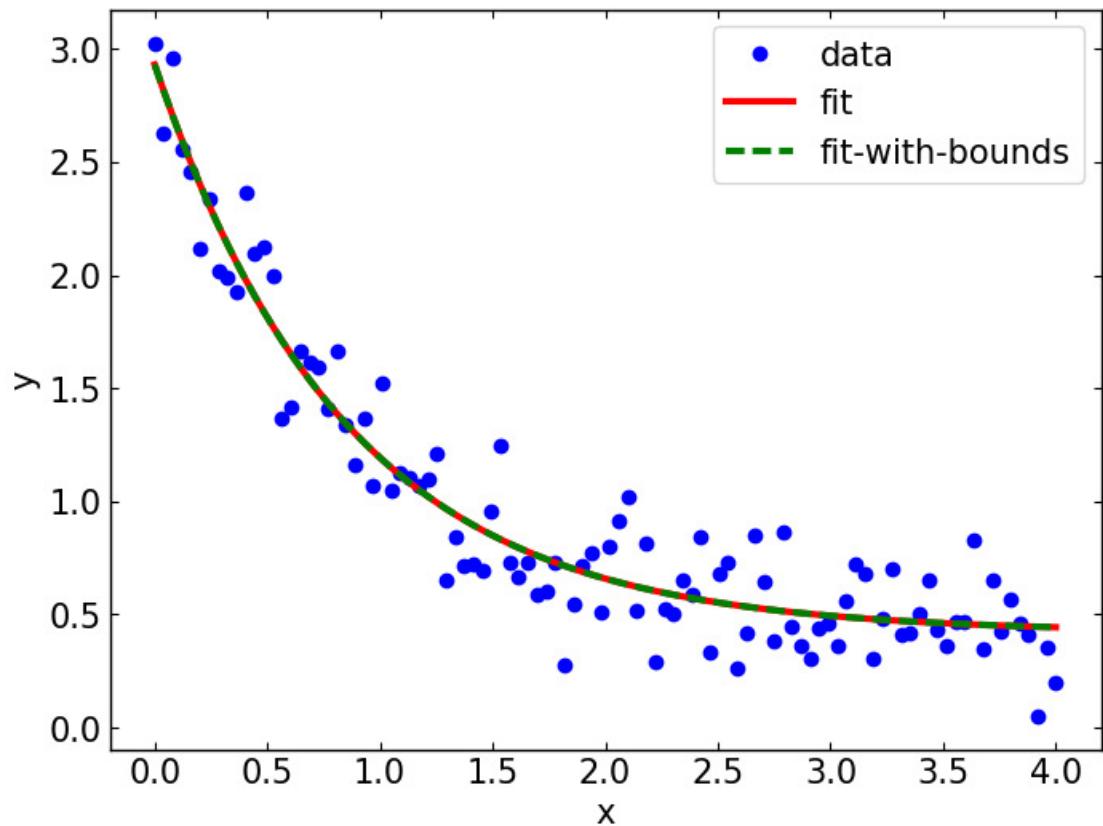
# Constrain the optimization to the region of ``0 < a < 3``, ``0 < b < 2``
# and ``0 < c < 1``:

popt, pcov = curve_fit(func, xdata, ydata, bounds=(0, [3., 2., 1.]))
ax.plot(xdata, func(xdata, *popt), 'g--', label='fit-with-bounds', linewidth=3)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend(loc='best')

```

Out[6]: <matplotlib.legend.Legend at 0x7f5e69ae75d0>



Sod ShockTube

January 30, 2019

1 pyds.numerical.sodshock

- Calculate the analytical values for the sod shocktube problem in adiabatic equation of states

```
In [1]: from pyds.numerical import sodshock
```

```
sod = sodshock(d=[1.,0.125], p=[1.,0.1], v=[0.,0.], gamma=1.66667 \
, time=0.245, resol=200, xscale=[0.,1.])
```

```
In [2]: import matplotlib.pyplot as plt
import numpy as np
from pyds.tools import axpos
```

```
%matplotlib inline

# initialize the configuration of multiple plots
axset = axpos(22,pltx0=[30,10],plty0=[20,10],pltxs=100,pltys=100,pltxw=30.,pltyw=25.)

fig = plt.figure(figsize=(8,8.*axset.winys/axset.winxs))

# density plot
ax1 = fig.add_subplot(221)
ax1.plot(sod.x,sod.d)
ax1.axvline(x=sod.xcd, linestyle=':', linewidth=1, color='k')
ax1.axvline(x=sod.xsh, linestyle=':', linewidth=1, color='k')
ax1.set_xlabel('time')
ax1.set_ylabel('density')

# pressure plot
ax2 = fig.add_subplot(222)
ax2.plot(sod.x,sod.p)
ax2.axvline(x=sod.xcd, linestyle=':', linewidth=1, color='k')
ax2.axvline(x=sod.xsh, linestyle=':', linewidth=1, color='k')
ax2.set_xlabel('time')
ax2.set_ylabel('pressure')

# velocity plot
ax3 = fig.add_subplot(223)
```

Sympy

January 30, 2019

- 0.0.1 Sympy is designed to calculate for symbolic mathematics in a full featured computer algebra system that can compete directly with commercial alternatives (Mathematica, Maple).

```
In [1]: import sympy as sp

In [2]: a = sp.Rational(1,2)
         print type(a)
         print a

<class 'sympy.core.numbers.Half'>
1/2

In [3]: print sp.pi**2

         print sp.pi.evalf()
         print (sp.pi*sp.exp(1)).evalf()

pi**2
3.14159265358979
8.53973422267357

In [4]: # infinity is "oo"

         print sp.oo > 99999
         print sp.oo + 1

True
oo

In [5]: x = sp.Symbol('x')
         y = sp.Symbol('y')

         print x+y+x-2*y
         print (x+y)**2

2*x - y
(x + y)**2
```

0.0.2 expand

```
In [6]: print sp.expand((x+y)**3)

print sp.expand(x+y, complex=True)

print sp.expand(sp.cos(x+y), trig=True)

x**3 + 3*x**2*y + 3*x*y**2 + y**3
re(x) + re(y) + I*im(x) + I*im(y)
-sin(x)*sin(y) + cos(x)*cos(y)
```

0.0.3 simplify

```
In [7]: print sp.simplify((x+x+y)/x)

2 + y/x
```

0.1 Calculus

0.1.1 limit

```
In [8]: """ f(x) as x -> 0 """

print sp.limit(sp.sin(x)/x, x, 0)

1
```

```
In [9]: print sp.limit(x,x,sp.oo)
print sp.limit(1/x,x,sp.oo)
print sp.limit(x**x,x,0)

oo
0
1
```

0.1.2 differentiation

```
In [10]: print sp.diff(sp.sin(x),x)
print sp.diff(sp.sin(2*x),x)

print sp.diff(sp.tan(x),x)
print sp.limit((sp.tan(x+y)-sp.tan(x))/y,y,0)

cos(x)
2*cos(2*x)
```

```
tan(x)**2 + 1  
tan(x)**2 + 1
```

In [11]: *"higher derivatives"*

```
print sp.diff(sp.sin(2*x), x, 1)  
print sp.diff(sp.sin(2*x), x, 2)  
print sp.diff(sp.sin(2*x), x, 3)  
  
2*cos(2*x)  
-4*sin(2*x)  
-8*cos(2*x)
```

0.1.3 Taylor series expansion

In [12]: `print sp.series(sp.cos(x), x, n=10)`

```
print sp.series(1/sp.cos(x), x)  
  
1 - x**2/2 + x**4/24 - x**6/720 + x**8/40320 + O(x**10)  
1 + x**2/2 + 5*x**4/24 + O(x**6)
```

0.1.4 integration

In [13]: `print sp.integrate(6*x**5, x)`

```
print sp.integrate(sp.sin(x), x)  
print sp.integrate(sp.log(x), x)  
print sp.integrate(2*x + sp.sinh(x), x)  
  
print sp.integrate(sp.exp(-x**2)*sp.erf(x), x)  
  
x**6  
-cos(x)  
x*log(x) - x  
x**2 + cosh(x)  
sqrt(pi)*erf(x)**2/4
```

In [14]: `print sp.integrate(x**3, (x, -1, 1))`

```
print sp.integrate(sp.sin(x), (x, 0, sp.pi/2))  
print sp.integrate(sp.cos(x), (x, -sp.pi/2, sp.pi/2))
```

```
0  
1  
2
```

```
In [15]: print sp.integrate(sp.exp(-x), (x, 0, sp.oo))  
  
print sp.integrate(sp.exp(-x**2), (x, -sp.oo, sp.oo))  
  
1  
sqrt(pi)
```

0.2 Equation solving

```
In [16]: a = sp.solve(x**4 - 1, x)  
print type(a)  
print a  
  
<type 'list'>  
[-1, 1, -I, I]  
  
In [17]: b = sp.solve([x + 5*y - 2, -3*x + 6*y - 15], [x,y])  
print type(b)  
print b  
  
<type 'dict'>  
{x: -3, y: 1}
```

```
In [18]: print sp.solve(sp.exp(x)+1, x)  
  
[I*pi]
```

```
In [19]: """ polynomial factorization """  
  
f = x**4 - 3*x**2 + 1  
print sp.factor(f)  
  
(x**2 - x - 1)*(x**2 + x - 1)
```

0.3 Linear Algebra

```
In [20]: a = sp.Matrix([[1,0], [0,1]])  
print type(a)  
print a
```

```

<class 'sympy.matrices.dense.MutableDenseMatrix'>
Matrix([[1, 0], [0, 1]])

In [21]: """ unlike to Numpy Array, you can also put Symbols in the Matrix """
         

x = sp.Symbol('x')
y = sp.Symbol('y')

a = sp.Matrix([[1,x],[y,1]])
print a

print a**2

Matrix([[1, x], [y, 1]])
Matrix([[x*y + 1, 2*x], [2*y, x*y + 1]])

```

0.3.1 ordinary differential equations

```

In [22]: from sympy.abc import x

f = sp.Function('f')
print f(x).diff(x,x) + f(x)

print sp.ode.dsolve(f(x).diff(x,x) + f(x), f(x))

f(x) + Derivative(f(x), x, x)
Eq(f(x), C1*sin(x) + C2*cos(x))

```