02_14_plotting_sweeping

Unknown Author

February 17, 2014

Part I

Plotting and coding the algorithms we learnt in the last class.

1 Plotting

Numpy and Scipy *Look at* my webpage *for instructions to install these python packages on your laptop.*

Array/Vectors

Suppose you want to plot $y = x^2, -1 \leq x \leq 1$. From our math knowledge, we know this is a curve in $\mathbb{R}^2$. The way one plots such functions in matlab/mathematica, is to find a lot of points lying on the curve and then joining them. We can use a list to store those points. So a basic python code to generate such lists will be as follows

In [2]:
```
def genxy(f, xmin, xmax, N) :
    """Given a function f, an interval [xmin, xmax] in R and number of points N, This
       of two lists: First an equispaced points on x-axis, and the second one is the v
    step = float(xmax - xmin)/N
    listx = [xmin + i * step for i in range(N+1)]
    listy = [f(x) for x in listx]
    return (listx, listy)
```

In [5]:
```
a = genxy(lambda x : x*x, -1, 1, 10)
azipped = zip(a[0], a[1])
for x, y in azipped :
    print "%6.3f\t%6.3f" % (x, y)
```

```
-1.000   1.000
-0.800   0.640
-0.600   0.360
-0.400   0.160
-0.200   0.040
 0.000   0.000
 0.200   0.040
 0.400   0.160
 0.600   0.360
 0.800   0.640
 1.000   1.000
```

This is not too difficult, but is somewhat tedious. To make tasks simpler, one can use arrays. Arrays are **not** in-built in python. To use them, *you have to install numpy*.

In [9]:
```
# The above code becomes :
from numpy import *
xs = array(a[0])
ys = array(a[1])
type(xs)
```

Out [9]:
```
numpy.ndarray
```

1. In an array all elements must be of the same type. If they are not, they are converted to the same type as is shown below (in the examples).
2. The computations are faster if the size of the array doesn't change during computation.
3. You can mathematical operation on a whole array and that is faster than doing it by traversing a list.

In [27]:
```
# Array elements are all of the same type
v1 = [4, 5, 100, 4, 23.4]
w1 = array(v1)
print w1
```

```
[   4.     5.   100.     4.    23.4]
```

In [28]:
```
v2 = ['eggs', 4.3, 3]
w2 = array(v2)
print w2
```

```
['eggs' '4.3' '3']
```

In [29]:
```
v3 = [3, 2, 5]
w3 = array(v3)
print w3
```

```
[3 2 5]
```

2

In [30]:
```
# doing operations on array
def f(x) :
    return x + x*x

v = array(range(5))
print f(v)
```

```
[ 0  2  6 12 20]
```

In [33]:
```
# One dimensional arrays are like vectors
v = array([1, 4, 5])
w = array([5, 2, 1])
print v + w
print 2 * v
print v * w
```

```
[6 6 6]
[ 2  8 10]
[5 8 5]
```

Back to plotting.

We need two things : 1. We need to generate an array of equidistant points on x-axis. For this we have a numpy function called **linspace** 1. We need to generate another array containing their values.

In [35]:
```
xlist = linspace(-1, 1, 11)
print xlist
```
```
[-1.  -0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8  1. ]
```

In [36]:
```
def square(x) :
    return x*x
ylist = square(xlist)
print ylist
```
```
[ 1.    0.64  0.36  0.16  0.04  0.    0.04  0.16  0.36  0.64  1.  ]
```

Good, it works. Let us now write it as a function.

In [37]:
```
def array_pts(f, xmin, xmax, N) :
    xlist = linspace(xmin, xmax, N+1)
    ylist = f(xlist)
    return (xlist, ylist) # pretty simple, isn't it
```

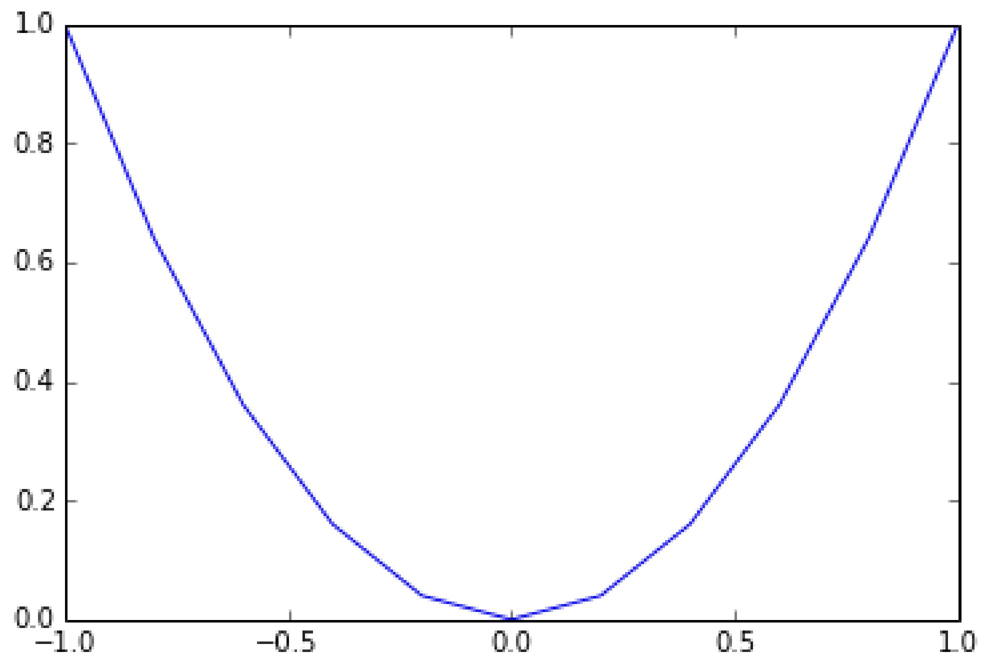
Plotting

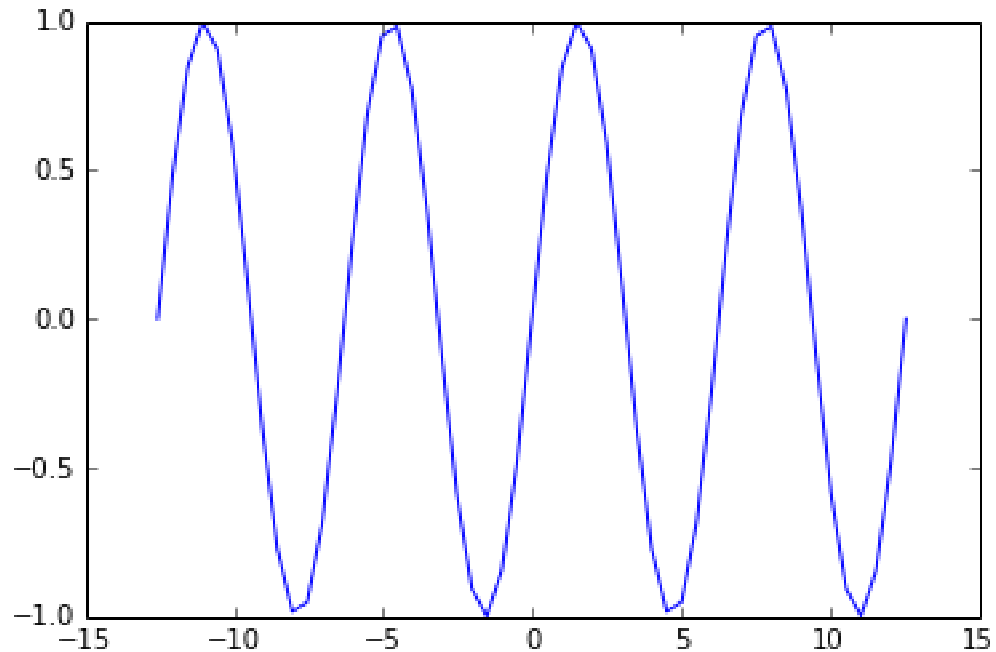
To plot we have a bunch of functions, I like matplotlib. It has a MatLab like syntax.

In [39]:
```
%matplotlib inline
```

In [47]:
```
from matplotlib.pylab import *

a = array_pts(lambda x : x*x, -1, 1, 10)
plot(a[0], a[1])
show()
b = array_pts(sin, -4*pi, 4*pi, 50)
plot(b[0], b[1])
show()
```





Some comments on trigonometric functions

Note that we have not imported math anywhere. The sin we used here is imported from numpy. Actually if we override with the sin from math, we'll get errors :

In [48]:
```
from matplotlib.pylab import *
import math as m

b = array_pts(m.sin, -4*pi, 4*pi, 50)
plot(b[0], b[1])
show()
```

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



        <ipython-input-48-66a6bff9e21c> in <module>()
          2 import math as m
          3
    ----> 4 b = array_pts(m.sin, -4*pi, 4*pi, 50)
          5 plot(b[0], b[1])
          6 show()




        <ipython-input-37-925050db2e56> in array_pts(f, xmin, xmax, N)
          1 def array_pts(f, xmin, xmax, N) :
```

```
          2      xlist = linspace(xmin, xmax, N+1)
    ----> 3      ylist = f(xlist)
          4      return (xlist, ylist) # pretty simple, isn't it
```

```
        TypeError: only length-1 arrays can be converted to Python
scalars
```

3 Accessing array elements

Array elements can be accessed in the same way as list elements x = array(range(1, 10)) print x[5]

In [49]:
```
x = array(range(1, 10))
print x[5]
```
```
6
```

However, you must be careful about assignment. Assignment just reflects the same array, and does not create a new array, just like lists:

In [53]:
```
a = array(range(1, 10))
l = range(1, 10)
x = a
m = l
print a[5], l[5]
x[5] = -1
m[5] = -1
print a[5], l[5]
print type(a), type(l)
```
```
6 6
-1 -1
<type 'numpy.ndarray'> <type 'list'>
```

In [55]:
```
a before : [ 1  2  3  4  5 -1  7  8  9]
y before : [ 1  2  3  4  5 -1  7  8  9]
 a after : [ 1  2  3  4  5 -1  7  8  9]
 y after : [ 1  2 -2  4  5 -1  7  8  9]
```

Note :

For vectors

```
a += b
```

is faster than

```
a = a + b
```

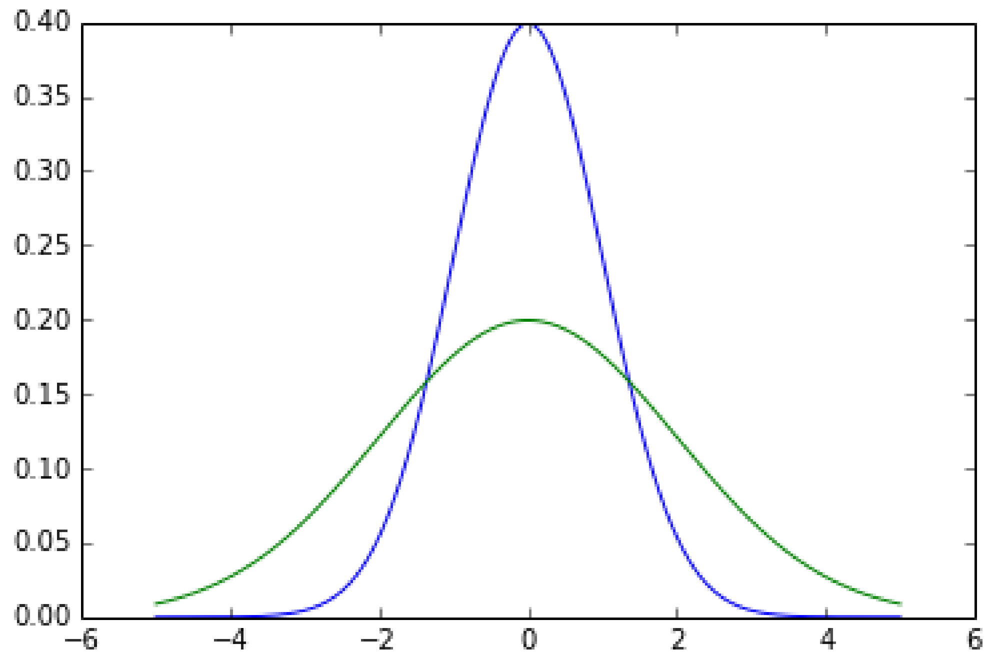
since the later creates a new array to store `a + b` before copying it back to `a`. `a += b` just modifies a directly. Same holds for -=, *= etc.

Putting two graphs in he same picture

One uses the hold funciton.

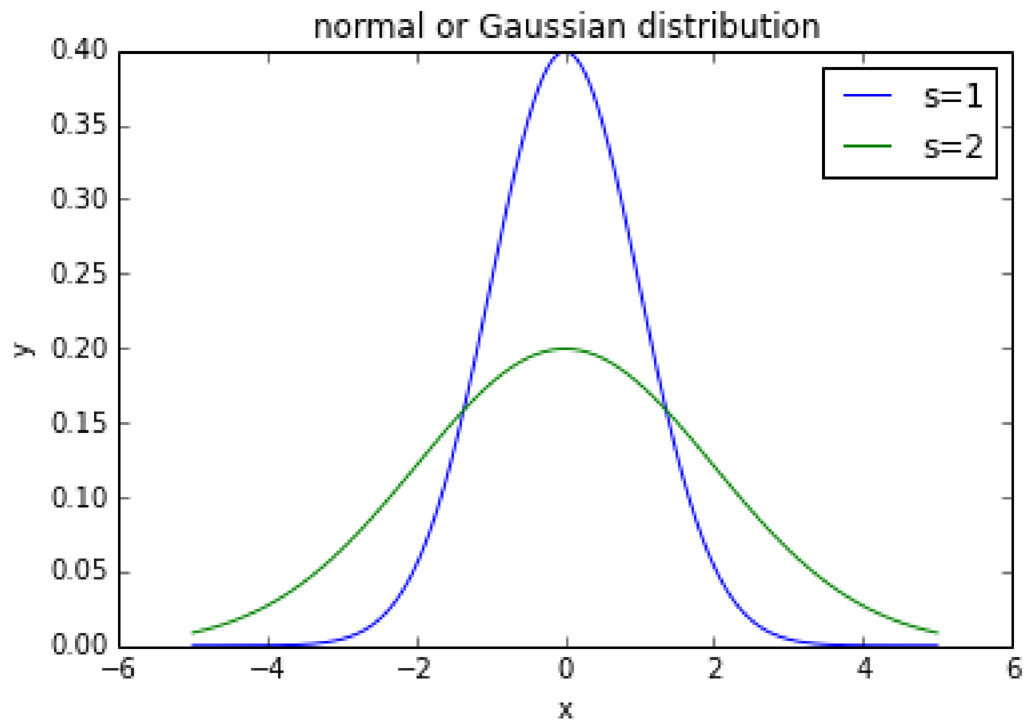
In [59]:
```
def f(x, s):
    return 1 / (sqrt(2 * pi) * s) * exp(- x * x / (2 * s * s))

x = linspace(-5, 5, 1000)
y = f(x, 1)
plot(x, y)
hold('on')
z = f(x, 2)
plot(x, z)
show()
```



Some decorations

In [68]:
```
x = linspace(-5, 5, 1000)
y = f(x, 1)
plot(x, y)
hold('on')
z = f(x, 2)
plot(x, z)
xlabel('x')
ylabel('y')
legend(['s=1', 's=2'])
title('normal or Gaussian distribution')
show()
```


normal or Gaussian distribution
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
y
−6
−4
−2
0
2
4
6
x
s=1
s=2