Ch.5: Array computing and curve plotting (Part 1)

Joakim Sundnes[1,2] Hans Petter Langtangen[1,2]

Simula Research Laboratory[1]

University of Oslo, Dept. of Informatics[2]

Sep 20, 2017

(Adjusted) Plan for week 38

Wednesday 20 september

- Live programming of ex 4.4, 4.5, 4.6
- Intro to plotting and NumPy arrays

Friday 22 september

- Live programming of ex 4.7, 5.7, 5.9, 5.10, 5.11, 5.13
- Plotting functions with `matplotlib`
- (Making movies and animations from plots) Moved to next week
- (Making your own Python modules) Next week

Small quiz:

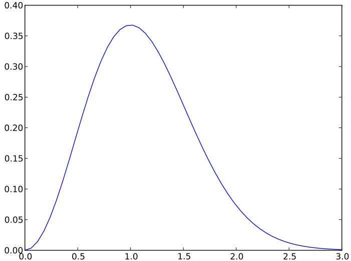
What is output from the following code? Why?

```python
import numpy as np

l = [0,0.25,0.5,0.75,1]
a = np.array(l)

print(l*2)
print(a*2)
```

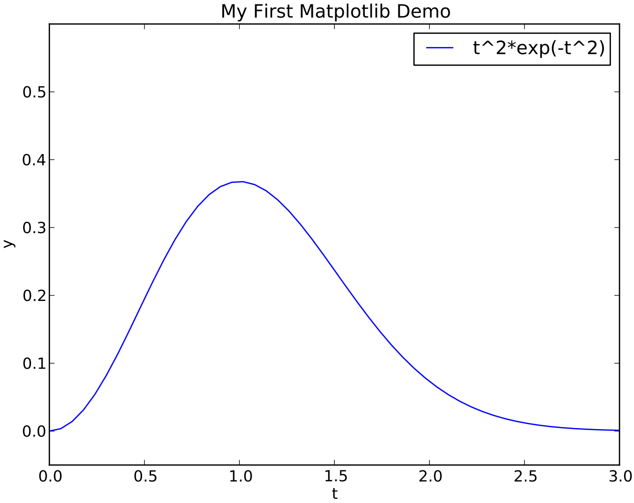
Plotting the curve of a function: the very basics

Plot the curve of $y(t) = t^2 e^{-t^2}$:

```
from matplotlib.pyplot import *  # import and plotting
from numpy import *

# Make points along the curve
t = linspace(0, 3, 51)      # 50 intervals in [0, 3]
y = t**2*exp(-t**2)         # vectorized expression

plot(t, y)           # make plot on the screen
savefig('fig.pdf')   # make PDF image for reports
savefig('fig.png')   # make PNG image for web pages
show()
```



A plot should have labels on axis and a title



The code that makes the last plot

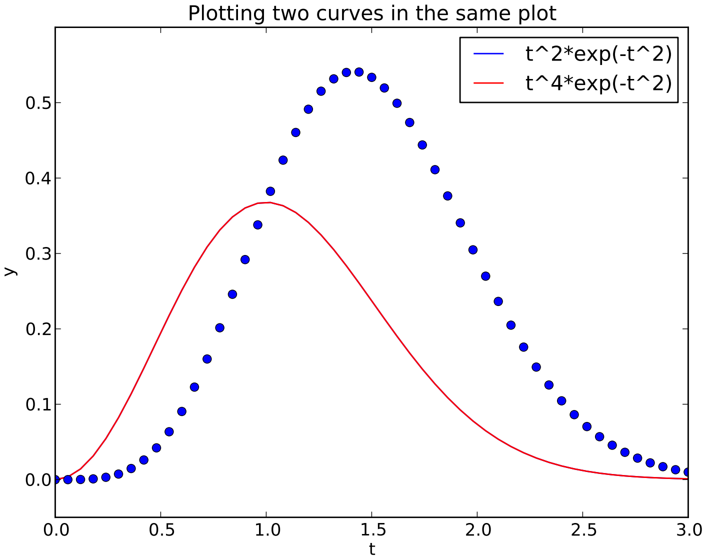
```
from matplotlib.pyplot import *
from numpy import *

def f(t):
    return t**2*exp(-t**2)

t = linspace(0, 3, 51)    # t coordinates
y = f(t)                  # corresponding y values

plot(t, y,label="t^2*exp(-t^2)")

xlabel('t')               # label on the x axis
ylabel('y')               # label on the y axix
legend()                  # mark the curve
axis([0, 3, -0.05, 0.6])  # [tmin, tmax, ymin, ymax]
title('My First Matplotlib Demo')
show()
```

Plotting several curves in one plot

Plot $t^2 e^{-t^2}$ and $t^4 e^{-t^2}$ in the same plot:

```python
from matplotlib.pyplot import *
from numpy import *

def f1(t):
    return t**2*exp(-t**2)

def f2(t):
    return t**2*f1(t)

t = linspace(0, 3, 51)
y1 = f1(t)
y2 = f2(t)

plot(t, y1,  'r-', label = 't^2*exp(-t^2)')
plot(t, y2, 'bo', label = 't^4*exp(-t^2)')

xlabel('t')
ylabel('y')
legend()
title('Plotting two curves in the same plot')
savefig('tmp2.png')
show()
```

The resulting plot with two curves



Controlling line styles

When plotting multiple curves in the same plot, the different lines (normally) look different. We can control the line type and color, if desired:

```
plot(t, y1, 'r-')    # red (r) line (-)

plot(t, y2, 'bo')    # blue (b) circles (o)

# or
plot(t, y1, 'r-', t, y2, 'bo')
```

Documentation of colors and line styles: see the book, Ch. 5, or

```
Unix> pydoc matplotlib.pyplot
```

Quick plotting with minimal typing

A lazy pro would do this:

```
t = linspace(0, 3, 51)
plot(t, t**2*exp(-t**2), t, t**4*exp(-t**2))
```

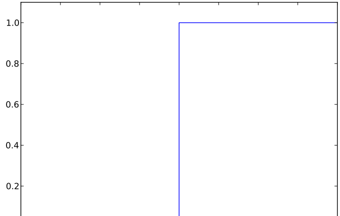
Let's try to plot a discontinuous function

The Heaviside function is frequently used in science and engineering:

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Python implementation:

```
def H(x):
    if x < 0:
        return 0
    else:
        return 1
```



Plotting the Heaviside function: first try

Standard approach:

```
x = linspace(-10, 10, 5)  # few points (simple curve)
y = H(x)
plot(x, y)
```

First problem: `ValueError` error in `H(x)` from `if x < 0`

Let us debug in an interactive shell:

```
>>> x = linspace(-10,10,5)
>>> x
array([-10.,  -5.,   0.,   5.,  10.])
>>> b = x < 0
>>> b
array([ True,  True, False, False, False], dtype=bool)
>>> bool(b)  # evaluate b in a boolean context
...
ValueError: The truth value of an array with more than
one element is ambiguous. Use a.any() or a.all()
```

if x < 0 does not work if x is array

Remedy 1: use a loop over x values

```python
def H_loop(x):
    r = zeros(len(x))  # or r = x.copy()
    for i in range(len(x)):
        r[i] = H(x[i])
    return r

n = 5
x = linspace(-5, 5, n+1)
y = H_loop(x)
```

Downside: much to write, slow code if n is large

`if x < 0` does not work if `x` is array

Remedy 2: use `vectorize`

```
from numpy import vectorize

# Automatic vectorization of function H
Hv = vectorize(H)
# Hv(x) works with array x
```

Downside: The resulting function is as slow as Remedy 1

`if x < 0` does not work if `x` is array

Remedy 3: code the `if` test differently

```
def Hv(x):
    return where(x < 0, 0.0, 1.0)
```

More generally:

```
def f(x):
    if condition:
        x = <expression1>
    else:
        x = <expression2>
    return x

def f_vectorized(x):
    x1 = <expression1>
    x2 = <expression2>
    r = np.where(condition, x1, x2)
    return r
```

if x < 0 does not work if x is array

Remedy 3: code the if test differently

```
def Hv(x):
    return where(x < 0, 0.0, 1.0)
```

More generally:

```
def f(x):
    if condition:
        x = <expression1>
    else:
        x = <expression2>
    return x

def f_vectorized(x):
    x1 = <expression1>
    x2 = <expression2>
    r = np.where(condition, x1, x2)
    return r
```

`if x < 0` does not work if `x` is array

Remedy 3: code the `if` test differently

```
def Hv(x):
    return where(x < 0, 0.0, 1.0)
```

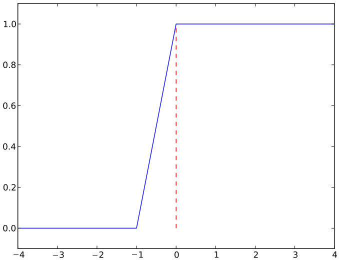
More generally:

```
def f(x):
    if condition:
        x = <expression1>
    else:
        x = <expression2>
    return x

def f_vectorized(x):
    x1 = <expression1>
    x2 = <expression2>
    r = np.where(condition, x1, x2)
    return r
```

Back to plotting the Heaviside function

With a vectorized `Hv(x)` function we can plot in the standard way

```
x = linspace(-10, 10, 5)    # linspace(-10, 10, 50)
y = Hv(x)
plot(x, y, axis=[x[0], x[-1], -0.1, 1.1])
```



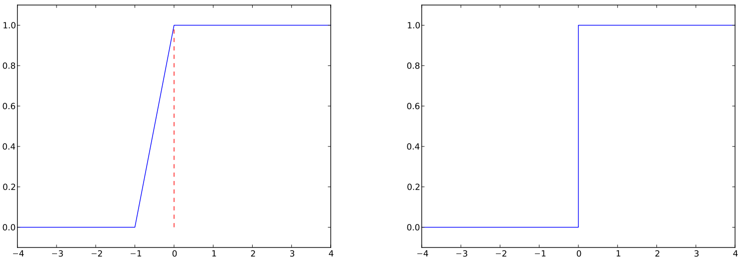
How to make the function look discontinuous in the plot?

- Newbie: use a lot of x points; the curve gets steeper
- Pro: plot just two horizontal line segments
  one from $x = -10$ to $x = 0$, $y = 0$; and one from $x = 0$ to $x = 10$, $y = 1$

```
plot([-10, 0, 0, 10], [0, 0, 1, 1],
     axis=[x[0], x[-1], -0.1, 1.1])
```

Draws straight lines between $(-10, 0)$, $(0, 0)$, $(0, 1)$, $(10, 1)$

The final plot of the discontinuous Heaviside function



Removing the vertical jump from the plot

Question

Some will argue and say that at high school they would draw $H(x)$ as two horizontal lines *without* the vertical line at $x = 0$, illustrating the jump. How can we plot such a curve?

Plot function given on the command line

Task: plot function given on the command line

```
Terminal> python plotf.py expression xmin xmax
Terminal> python plotf.py "exp(-0.2*x)*sin(2*pi*x)" 0 4*pi
```

Should plot $e^{-0.2x}\sin(2\pi x)$, $x \in [0, 4\pi]$. `plotf.py` should work for "any" mathematical expression.

Solution

Complete program:

```
from numpy import *
from matplotlib.pyplot import *

formula = sys.argv[1]
xmin = eval(sys.argv[2])
xmax = eval(sys.argv[3])

x = linspace(xmin, xmax, 101)
y = eval(formula)
plot(x, y, title=formula)
show()
```