第 5 次课 数组计算和曲线绘图

Python 科学计算

周吕文

宁波大学，机械工程与力学学院

2024 年 9 月 1 日

提要

向量

中学数学的概念

- (x, y) 表示平面上的一个点，(x, y, z) 表示立体空间的一个点。

向量的数学定义

- 数学上，向量就是一个 n 元组：$v = (v_0, \cdots, v_{n-1})$

计算机中的向量

- 向量可以使用列表（`list`）或数组（`array`）存储，v_i 存储为 `v[i]`。
- 数组 $\approx$ 列表，但比列表更高效。因此，应尽可能使用数组来表示向量。

数组

- 数组可以看做是向量的一般形式，可能会有多个索引：$A_{i,j}$，$A_{i,j,k}$
- 例如表格（矩阵），有行索引和列索引

$$\begin{bmatrix} 1 & 9 & 9 & 4 \\ 0 & 7 & 2 & 9 \\ 3 & 0 & 2 & 9 \\ 9 & 3 & 8 & 4 \end{bmatrix} \qquad A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{bmatrix}$$

- 索引的数量是数组的维度，向量是一维数组，A 是二维数组
- 在 Python 中，使用 Numerical Python 数组替代嵌套的列表，更高效

使用数组存储由曲线点坐标构成的列表

使用列表存储

```
>>> f = lambda x: x**3
>>> n = 5                    # 点的个数
>>> dx = 1/(n-1)             # 步长
>>> xlist = [i*dx for i in range(n)]
>>> ylist = [f(x) for x in xlist]
```

转为数组存储

```
>>> import numpy as np  # 要使用数组，需要载入 numpy 库
>>> x = np.array(xlist)
>>> y = np.array(ylist)
>>> x
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
>>> y
array([0.      , 0.015625, 0.125   , 0.421875, 1.      ])
```

直接使用数组

使用循环计算 y

```
>>> x = np.linspace(0, 1, n)
>>> y = np.zeros(n)
>>> for i in range(n):
...     y[i] = f(x[i])
...
>>> x
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
>>> y
array([0.      , 0.015625, 0.125   , 0.421875, 1.      ])
```

直接计算 y

```
>>> y = f(x)
>>> y
array([0.      , 0.015625, 0.125   , 0.421875, 1.      ])
```

数组和列表的差别

- 列表可以包含任意类型的对象，一个数组只能包含同一类型的对象
- 数组的维度和尺寸不可更改，在创建时就必需确定
- 如果数组的元素类型是整数、浮点数或复数时，存储和计算非常高效
- 数组不是标准 Python 的一部分，需要额外的库 numpy
- 使用 numpy 可直接对数组进行数学运算，避免循环，这称为向量化

数组在索引、切片、修改上与列表类似

```
>>> import numpy as np
>>> r = [1, 1.5, 2, 2.5]
>>> a = np.array(r)
>>> a
array([1. , 1.5, 2. , 2.5])
>>> c = np.zeros_like(r)
>>> c
array([0., 0., 0., 0.])
```

```
>>> a[0]
1.0
>>> b = a[1:-1]
>>> b
array([1.5, 2. ])
>>> a[1] = 0.1
>>> a
array([1. , 0.1, 2. , 2.5])
```

可以一次性操作整个数组 → 向量化

使用列表和 math 库中函数

```
>>> from math import pi, sin
>>> x = [pi, pi/2, pi/3]; y = [1, 2, 3]
>>> [sin(xi) for xi in x]
[1.2246467991473532e-16, 1.0, 0.8660254037844386]
>>> [x[i] + y[i] for i in range(len(x))]
[4.141592653589793, 3.5707963267948966, 4.047197551196597]
```

使用 numpy 库及其函数：底层计算和循环使用 C 语言实现，更高效

```
>>> import numpy as np
>>> x = np.array(x); y = np.array(y)
>>> np.sin(x)
array([1.22464680e-16, 1.00000000e+00, 8.66025404e-01])
>>> x + y
array([4.14159265, 3.57079633, 4.04719755])
```

向量化：程序更短、可读性更好、运行更快，更接近数学

$y = \sin(x)$ 的向量化加速 — np_speed_test.py

```
import numpy as np; from math import sin
from timeit import timeit
x = np.linspace(0, 2*np.pi, 100001); y = np.zeros(len(x))

t1 = timeit('for i in range(len(x)): y[i] = sin(x[i])',\
            globals=globals(), number=10)
t2 = timeit('for i in range(len(x)): y[i] = np.sin(x[i])',\
            globals=globals(), number=10)
t3 = timeit('y = np.sin(x)', globals=globals(), number=10)

print('loop: %2.4f, np loop:%2.4f, np: %2.4f' % (t1,t2,t3))
```

```
>>>
loop: 0.2339, np loop:0.7173, np: 0.0101
```

作用于单个数值的函数 f(x) 通常对适用于数组

```
>>> from numpy import sin, exp, linspace
>>> f = lambda x: x**3 + sin(x)*exp(-3*x)
>>> f(1.2)
1.7534667772949966
>>> x = linspace(0, 2, 3)
>>> f(x)
array([0.        , 1.04189437, 8.00225392])
```

math 用于数值计算，numpy 用于数组计算

```
>>> import math, numpy
>>> x = numpy.linspace(0, 1, 3)
>>> math.sin(x[1])                    # 注意，math.sin(x) 会报错
0.479425538604203
>>> numpy.sin(x)
array([0.        , 0.47942554, 0.84147098])
```

数组运算被分解为一系列的单目/双目数组操作

考虑计算 y = f(x), 其中 f(x) = x**3 + sin(x)*exp(-3)

1. `r1 = x**3`
   `for i in range(len(x)): r1[i] = x[i]**3` (由 C 语言实现)
2. `r2 = sin(x)` (由 C 语言实现)
3. `r3 = -3*x`
4. `r4 = exp(r3)`
5. `r5 = r3*r4`
6. `r6 = r1 + r5`
7. `y = r6`

注意

- 注意：数组运算的过程与在计算器上对单个数字 x 执行的操作相同

例子：曲线上的点坐标的向量化计算

$$f(x) = x^2 \mathrm{e}^{-x/2} \sin\left(x - \frac{1}{3}\pi\right), \quad x \in [0, 4\pi]$$

计算函数 $f(x)$ 曲线上 $n+1$ 个点 vec_cmpt_xy.py

```
from numpy import *
n = 5
x = linspace(0, 4*pi, n+1)
y = x**2*exp(-x/2)*sin(x-pi/3)
print('x =' + ' '.join(['%7.4f'%i for i in x]))
print('y =' + ' '.join(['%7.4f'%i for i in y]))
```

```
>>>
x =  0.0000  2.5133  5.0265  7.5398 10.0531 12.5664
y = -0.0000  1.7879 -1.5209  0.2725  0.2697 -0.2554
```

向量化

- 标量：一个数
- 向量或数组：数的序列（数学中的向量或矩阵）
- 标量计算：一次算一个数
- 向量计算：一次算一个数组，不使用 python 循环
- 向量化：将循环操作数组的算法转化为直接对整个数组进行操作的过程
- 在 Python 中，不含条件判断的数学函数可以自动处理标量和向量（数组）参数（即程序员无需手动进行向量化）

例子：阶跃函数的向量化

阶跃函数常用于工程和科学领域

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

```
>>> def H(x):
...     return (0 if x < 0 else 1)
...
>>> H(1)
1
>>> x = np.linspace(-10, 10, 5)
>>> H(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in H
ValueError: The truth value of an array with more than one ele
```

例子：阶跃函数的向量化

vectorize_heaviside_fun.py（接下页）

```
import numpy as np; from timeit import timeit

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

def H_loop(x):                  # 当 len(x) 很大时，会很慢
    r = np.zeros_like(x, dtype=int)
    for i in range(len(x)):
        r[i] = H(x[i])
    return r

H_vec1 = np.vectorize(H) # 当 len(x) 很大时，也很慢

def H_vec2(x):
    return np.where(x < 0, 0.0, 1.0)
```

例子：阶跃函数的向量化

vectorize_heaviside_fun.py（接上页）

```
x = np.linspace(-100, 100, 201)
t1 = timeit('y = H_loop(x)', globals=globals(), number=100)
t2 = timeit('y = H_vec1(x)', globals=globals(), number=100)
t3 = timeit('y = H_vec2(x)', globals=globals(), number=100)
print('loop: %2.4f, vec1: %2.4f, vec2: %2.4f' % (t1,t2,t3))
```

```
>>>
loop: 0.0044, vec1: 0.0028, vec2: 0.0004
```

含判断的一般向量化方法

```
y = <表达式 1> if <条件> else <表达式 2>
```

↓

```
y = np.where(<条件>, <表达式 1>, <表达式 2>)
```

数组赋值不是数组元素的拷贝：避免修改可以使用拷贝

```
>>> import numpy as np
>>> x = np.array([1, 3, 2, 4, 6])
>>> x
array([1, 3, 2, 4, 6])

>>> a = x      # 与列表相似
>>> a[-1] = 9
>>> x
array([1, 3, 2, 4, 9])

>>> b = x.copy()
>>> b[0] = 7
>>> b
array([7, 3, 2, 4, 9])

>>> x
array([1, 3, 2, 4, 9])
```

```
>>> a = x[2:] # 切片不同于列表
>>> a[-1] = 0
>>> x
array([1, 3, 2, 4, 0])

>>> b = x[2:].copy()
>>> b[-1] = 8
>>> b
array([2, 4, 8])

>>> x
array([1, 3, 2, 4, 0])
```

确保某个对象是数组（如果不是，把它变成数组）

```
>>> x = np.array([1, 2])
>>> a = np.asarray(x)
>>> a[1] = 0
>>> a
array([1, 0])
>>> x
array([1, 0])
```

```
>>> y = [1, 2]
>>> b = np.asarray(y)
>>> b[1] = 0
>>> b
array([1, 0])
>>> y
[1, 2]
```

```
>>> c = np.asarray(x, dtype=float)
>>> c
array([1., 0.])
>>> type(x)
<class 'numpy.ndarray'>
>>> isinstance(x, np.ndarray)
True
```

例子：常函数的向量化

vectorize_constant_fun.py

```
import numpy as np

def fs(x): return 2

def fv(x): return np.zeros(x.shape, x.dtype) + 2

def f(x):
    if isinstance(x, (float, int)): return fs(x)
    elif isinstance(x, np.ndarray): return fv(x)
    else: raise TypeError( 'x 必需数字或数组')

print(fs(12))
print(fv(np.array([1,2,3])))
print(f(12))
print(f(np.array([1,2,3])))
```

```
>>>
2
[2 2 2]
2
[2 2 2]
```

数组下标的批量操作

```
>>> a = linspace(1,7,7)
>>> a
array([1., 2., 3., 4., 5., 6., 7.])

>>> a[[1,3,5]]
array([2., 4., 6.])
>>> a[[1,3,5]] = 10
>>> a
array([ 1., 10.,  3., 10.,  5., 10.,  7.])

>>> a[2:6:2]        # 等价于 a[range(2,6,2)]
array([3., 5.])
>>> a[2:6:2] = -2 # 等价于 a[range(2,6,2)] = -2
>>> a
array([ 1., 10., -2., 10., -2., 10.,  7.])
```

使用布尔表达式作为数组下标

```
>>> a
array([ 1., 10., -2., 10., -2., 10.,  7.])

>>> b = a<0
>>> b
array([False, False,  True, False,  True, False, False])
>>> a[b]
array([-2., -2.])

>>> a[a<0]
array([-2., -2.])

>>> a[a<0] = a.max()
>>> a
array([ 1., 10., 10., 10., 10., 10.,  7.])
```

数组的一些方法和函数

array(ld)	将列表 ld 的数据复制建立 numpy 数组
asarray(d)	使用 d 创建数组，如果 d 是数组，则不拷贝
zeros(n)、zeros(n, int)	创建包含 n 个浮点、整数 0 的数组
zeros((m,n))	创建 m×n 的 0 数组
zeros_like(x)	创建数组，维度和数据都和 x 一样
linspace(a,b,m)	返回从 a 到 b（含）的 m 个等间距样本的数组
arange(a,b,d)	返回从 a 到 b（不含）固定步长 d 的排列数组
len(a)、a.shape、a.dtype	数组 a 的长度、维度（元组）、元素类型
a.reshape(3,2)	将 a 中的数据组织成 3×2 的数组
a[i]	访问序号为 i 的元素
a[i,j]	二维数组元素的访问
a[1:8:3]	切片：1,4,7
b = a.copy()	数组拷贝
sin(a), exp(a), ...	可作用于数组的 numpy 函数
c = concatenate((a, b))	连接 a 和 b，赋值给 c
c_[a,b]、r_[a,b]	按列或行增加拼接两个数组 a 和 b
c = where(cond, a1, a2)	c[i] = a1[i] if cond[i] else a2[i]
isinstance(a, ndarray)	如果 a 是 numpy 数组，则返回真

课堂练习

1 数组切片 文件名：array_slicing.py

- 创建一个值为 0, 0.1, 0.2, ···, 1.5 的数组 w。
- 打印出 w[:]、w[:-2]、w[::5]、w[2:-2:3]，并确保搞清楚输出的每种切片的具体含意。

2 填充数组 文件名：fill_arrays.py

在区间 $[-4, 4]$ 上等间隔地取 11 个坐标值，存储在数组 x 中。根据下式

$$h(x) = \frac{1}{2\pi} \mathrm{e}^{-\frac{1}{2}x^2}$$

计算相应的 $h(x)$ 存储在数组 y 中。使用以下两种方式实现以上操作：

- 使用 for 循环计算 x 和 y 数组的每一个元素。
- 使用 Numpy 包中的 linspace 函数产生 x，y 则使用向量参数 x 计算。

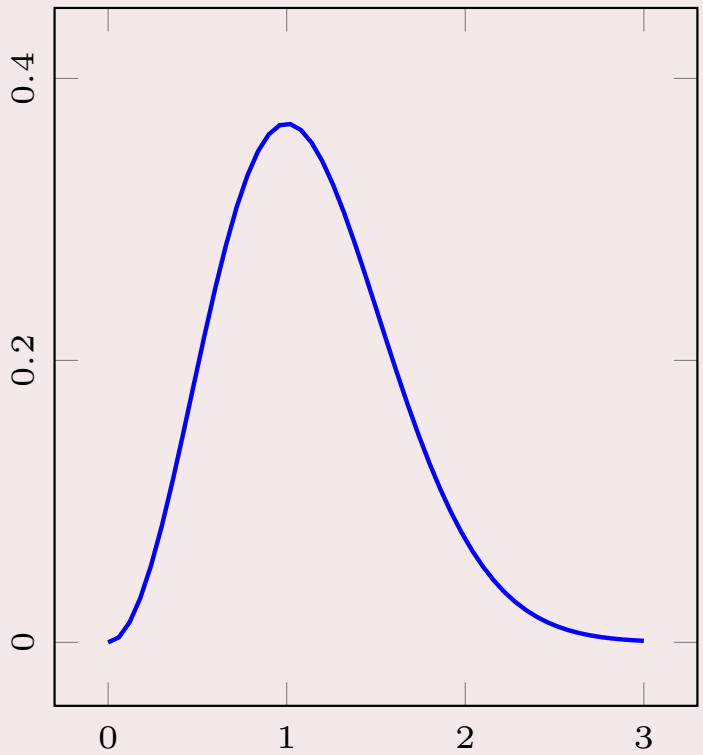
提要

绘制曲线图

pyplot_basic.py

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

t = np.linspace(0, 3, 51)
y = t**2*np.exp(-t**2)

plt.plot(t, y)

# 可保成图像为指定格式
plt.savefig('fig.pdf')
plt.savefig('fig.png')
plt.savefig('fig.eps')

plt.show()
```

$$y(t) = t^2 \mathrm{e}^{-t^2}$$

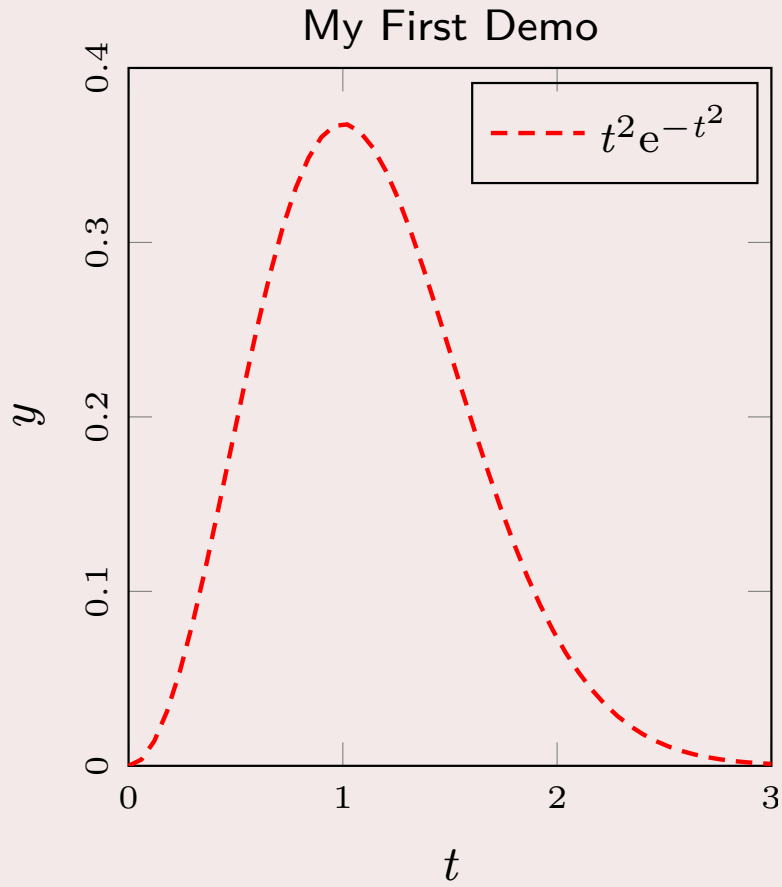


装饰图形

pyplot_decorate.py

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

t = np.linspace(0, 3, 51)
y = t**2*exp(-t**2)

plt.plot(t, y, '--r', \
         label='$t^2 e^{-t^2}$')

plt.axis([0,3, 0,0.4])
plt.title('My First Demo')
plt.xlabel('t'); plt.ylabel('y')
plt.legend()
plt.show()
```

$$y(t) = t^2 \mathrm{e}^{-t^2}$$



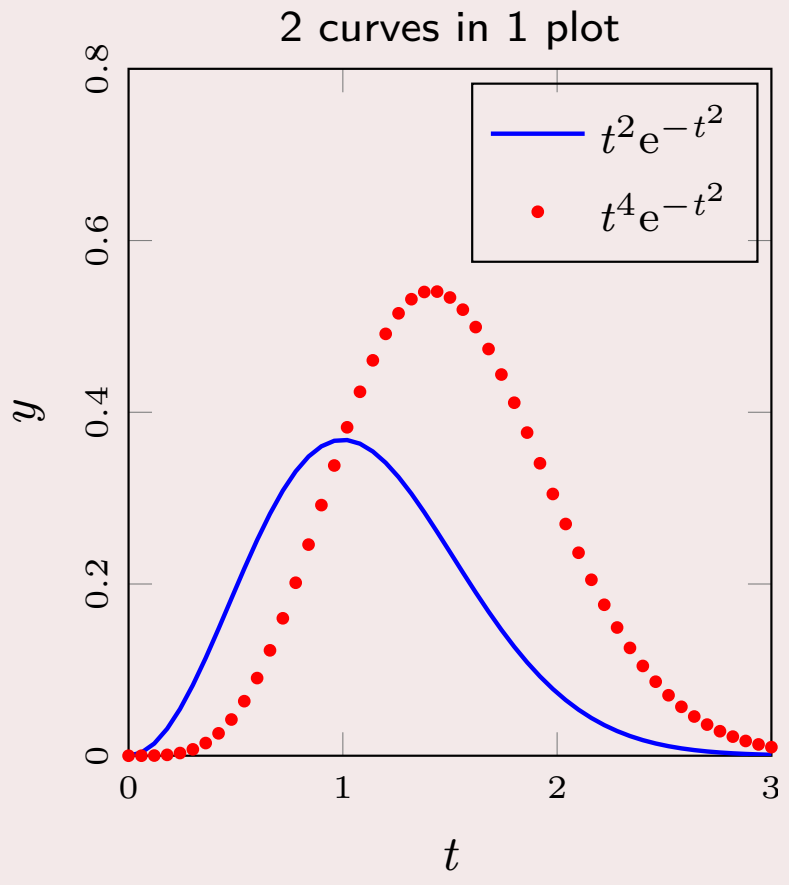
在一个图中画多条曲线

pyplot_2curves.py

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

f = lambda t: t**2*np.exp(-t**2)
h = lambda t: t**4*np.exp(-t**2)
t = np.linspace(0, 3, 51)

plt.plot(t, f(t), 'b-', \
         label='$t^2 e^{-t^2}$')
plt.plot(t, h(t), 'ro', \
         label='$t^4 e^{-t^2}$')
plt.axis([0,3,0,0.8])
plt.title('2 curves in 1 plot')
plt.xlabel('t'); plt.ylabel('y')
plt.legend(); plt.show()
```

$$y_1 = f(t) = t^2 \mathrm{e}^{-t^2}$$

$$y_2 = h(t) = t^4 \mathrm{e}^{-t^2}$$



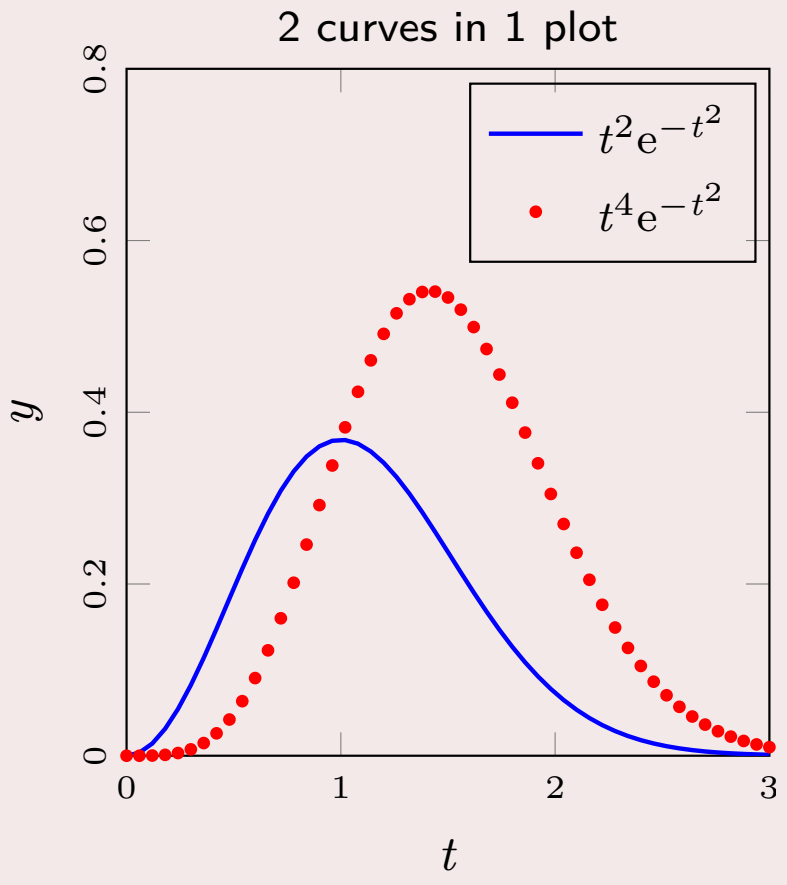
一条 plot 命令同时画两条曲线

pyplot_2curves2.py

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

f = lambda t: t**2*np.exp(-t**2)
h = lambda t: t**4*np.exp(-t**2)
t = np.linspace(0, 3, 51)

plt.plot(t, f(t), 'b-',\
         t, h(t), 'ro')
plt.legend(['$t^2 e^{-t^2}$',\
            '$t^4 e^{-t^2}$'])
plt.axis([0,3,0,0.8])
plt.title('2 curves in 1 plot')
plt.xlabel('t'); plt.ylabel('y')
plt.show()
```

$$y_1 = f(t) = t^2 \mathrm{e}^{-t^2}$$

$$y_2 = h(t) = t^4 \mathrm{e}^{-t^2}$$



课堂练习

3 绘制公式　　文件名：plot_ball.py

绘制函数图像：

$$y(t) = v_0 t - \frac{1}{2} g t^2, \quad g = 9.8, \quad t \in [0, 2v_0/g]$$

具体要求如下：

- 编写带参数的函数实现上式，并使用数组向量化计算曲线上的点坐标。
- 用不同的颜色分别绘制 $v_0 = 10$ 和 $v_0 = 12$ 的两条曲线。
- 为两条曲线添加相应的图例。
- 将坐标轴标签设置为“`time (s)`”和“`height (m)`”。
- 将坐标轴的显示范围调整到合适的区间。
- 添加图名为“`Position of the ball`”。

提要

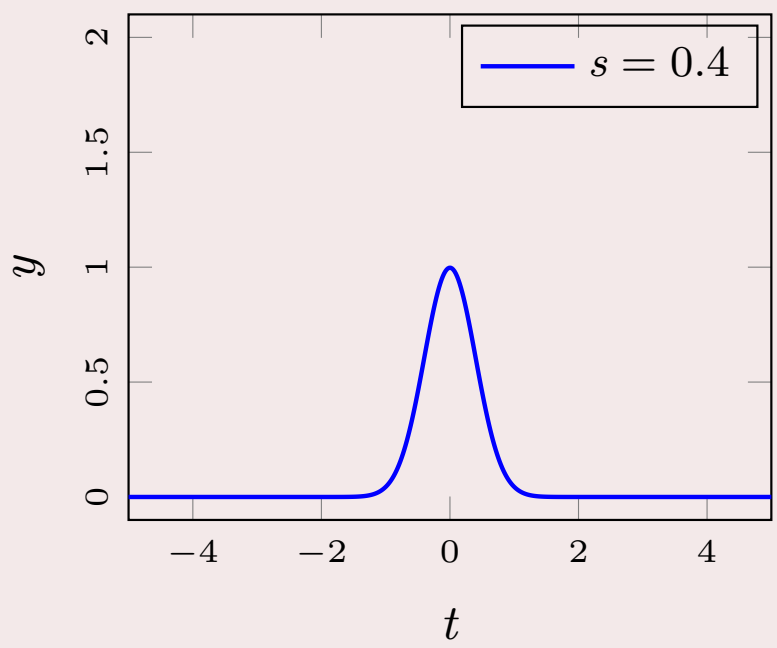
制作动画

高斯/贝尔函数：m 对称轴，s 函数的宽度

$$f(x, m, s) = \frac{1}{\sqrt{2\pi}s} \exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right]$$

动画是由一组图像构成

- **目的：** 制作动画，演示当 s 变小时，函数 f 的变化
- **方法：** 绘制 s 取不同值时的一组图像
- **实现：** s 从 2 循环到 0.2，为每个值绘制一幅图像，组合起来就是动画
- **注意：** 坐标轴范围不要变，否则会有影响效果



制作动画：生成每一帧图像

movie_mpl.py

```
import numpy as np; import matplotlib.pyplot as plt
def f(x, s, m=0):
    return 1/(2*np.pi)**0.5/s * np.exp(-1/2*((x-m)/s)**2)

S = np.linspace(2, 0.2, 31)
x = np.linspace(-5, 5, 1000)

lines = plt.plot(x, f(x,2))
plt.axis([-5, 5, -0.1, 2.1])
for i, s in enumerate(S):
    lines[0].set_ydata(f(x,s)) # set_data(x, f(x,s))
    plt.legend(['s = %4.2f' % s])
    plt.savefig('tmp_%04d.png' % i)
    plt.pause(0.1)
```

制作动画：将一组图像组合起来就是动画

Windows 系统：使用系统自带的 Movie Maker

- 导入所有图片（Home – Add Videos and Photos）
- 设置每个图片的播放时间，如 0.3 秒（Edit – Duration）
- 存储动画文件（File – Save Movie – For computer）

Linux 系统：使用 convert 和 ffmpeg 命令

```
nbu@ubuntu:~# convert -delay 20 tmp_*.png movie.gif
nbu@ubuntu:~# ffmpeg -framerate 5 -i tmp_%4d.png movie.mp4
```

在线工具：ezgif.com/maker

- 上传所有图片（Upload images – Choose Files – Upload files!）
- 设置每个图片的播放时间（Delay time）
- 生成动画文件（Make a GIF!）

制作动画：另一种方法 – 使用 FuncAnimation

movie_FuncAnimation.py

```
import matplotlib.animation as animation
fig = plt.figure()
lines = plt.plot(x, f(x,2))
plt.axis([-5, 5, -0.1, 2.1])

def frame(args): # 返回动画中的一帧图像
    i, s, x, lines = args
    y = f(x, s)
    lines[0].set_data(x, y)
    return lines

args = [(i, s, x, lines) for i, s in enumerate(S)]
m = animation.FuncAnimation(fig, frame, args, interval=150)
m.save('movie.gif', fps=5)
plt.show()
```

提要

二维数组的表示

```
>>> import numpy as np
>>> A = np.zeros((3,4))
>>> A
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
>>> A[0,0] = -1
>>> A[1,0] = 1
>>> A[2,0] = 10
>>> A[0,1] = -5
>>> A[2,3] = -100  # 也可以像访问嵌套列表一样: A[2][3] = -100
>>> A
array([[  -1.,   -5.,    0.,    0.],
       [   1.,    0.,    0.,    0.],
       [  10.,    0.,    0., -100.]])
```

将嵌套列表转换为二维数组

嵌套列表

```
>>> Cdegrees = [-30 + i*10 for i in range(3)]
>>> Fdegrees = [9/5*C + 32 for C in Cdegrees]
>>> table = [[C, F] for C, F in zip(Cdegrees, Fdegrees)]
>>> print(table)
[[-30, -22.0], [-20, -4.0], [-10, 14.0]]
```

二维数组

```
>>> arr2d = np.array(table)
>>> print(arr2d)
[[-30. -22.]
 [-20.  -4.]
 [-10.  14.]]
>>> arr2d.shape
(3, 2)
```

循环遍历二维数组

```
>>> for i in range(arr2d.shape[0]):          # 使用二重循环
...     for j in range(arr2d.shape[1]):
...         print('arr2d[%d,%d] = %g' % (i, j, arr2d[i,j]))
...
arr2d[0,0] = -30
...
arr2d[2,1] = 14
```

```
>>> for i, value in np.ndenumerate(arr2d): # 使用单重循环
...     print(' 索引 %s 值为 %g' % (i, value))
...
索引 (0, 0) 值为 -30
...
索引 (2, 1) 值为 14
```

获取二维数组的切片：对每一维分别使用 start:stop:inc

```
>>> arr2d
array([[-30., -22.],
       [-20.,  -4.],
       [-10.,  14.]])
>>> arr2d[0:arr2d.shape[0], 1] # = arr2d[0:,1] 或 arr2d[:,1]
array([-22.,  -4.,  14.])
>>> t = np.linspace(1, 30, 30).reshape(5, 6)
>>> t
array([[ 1.,  2.,  3.,  4.,  5.,  6.],
       [ 7.,  8.,  9., 10., 11., 12.],
       [13., 14., 15., 16., 17., 18.],
       [19., 20., 21., 22., 23., 24.],
       [25., 26., 27., 28., 29., 30.]])
>>> t[1:-1:2, 2:] # 对第一维取 1:-1:2, 对第二维取 2:
array([[ 9., 10., 11., 12.],
       [21., 22., 23., 24.]])
```

课堂练习

4 绘制双列文件中的数值　　文件名：read_2cols.py

文件 xy.dat 包含两列数据，对应曲线的 x、y 从标。编写程序，用以下两种方式将数据读入二维数组，并绘制相应的图像：

- 使用之前学过方式的读取文件。
- 函数 numpy.loadtxt 可读取表格数据文件，并返回二维数组。

5 用多项式拟合数据　　文件名：density_air_polyfit.py

文件 density_water.dat 包含水的温度和密度，请根据以下找到水的密度与温度的简单数学多项式关系：

- Numpy 中的函数 polyfit(x, y, n) 可以找到阶数为 n 的多项式

$$y = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$$

  最大程度（最小二乘）地拟合数组 x 和 y 中的数据。
- y = polyval(a, x) 则可以根据多项式系数 a 和数组 x 求相应的 y。

提要

逆、行列式和特征值

```
>>> import numpy as np
>>> A = np.array([[2, 0], [0, 5]], dtype=float)
>>> np.linalg.inv(A) # 逆矩阵
array([[0.5, 0. ],
       [0. , 0.2]])

>>> np.linalg.det(A) # 行列式
9.999999999999998

>>> eig_vals, eig_vecs = np.linalg.eig(A) # 特征值和特征向量
>>> eig_vals
array([2., 5.])
>>> eig_vecs
array([[1., 0.],
       [0., 1.]])
```

乘积和范数

```
>>> a = np.array([4, 0])
>>> b = np.array([0, 1])

>>> np.dot(A, a)       # 矩阵点积向量
array([8., 0.])
>>> np.dot(a, b)       # 向量点积向量
0

>>> B = np.ones((2, 2))
>>> np.dot(A, B)       # 矩阵点积矩阵
array([[2., 2.],
       [5., 5.]])

>>> np.linalg.norm(A) # 范数
5.385164807134504
```

求和与极值

```
>>> B = np.array([[1, 2], [3, -4]])
>>> np.sum(B)           # 求和，等价于 B.sum()
2
>>> np.sum(B, axis=0) # 沿着第 0 维索（行）引求和
array([ 4, -2])
>>> np.sum(B, axis=1) # 沿着第 1 维索（列）引求和
array([ 3, -1])

>>> np.max(B)           # 最大值，等价于 B.max()
3
>>> np.max(B, axis=0) # 沿着第 0 维索（行）引最大值
array([3, 2])
>>> np.min(B)           # 最小值，等价于 B.min()
-4
>>> np.min(B, axis=1) # 沿着第 1 维索（列）引最小值
array([ 1, -4])
```

课堂练习

6 验证线性代数结果 文件名：verify_linalg.py

生成三个 $n \times n$ 的随机矩阵 $\mathbf{A}$、$\mathbf{B}$ 和 $\mathbf{C}$，然后验证以下线性代数等式：

- $(\mathbf{A}+\mathbf{B})\mathbf{C} = \mathbf{AC}+\mathbf{BC}$
- $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$
- $\det(\mathbf{AB}) = \det\mathbf{A}\,\det\mathbf{B}$

7 求解线性方程组 文件名：linear_eqs.py

用 numpy 库中矩阵求逆的方式求解以下线性方程组：

$$\begin{cases} x+ \ y+ \ z= \ 6 \\ \quad 2y+5z=-4 \\ 2x+5y- \ z=27 \end{cases}$$

The End!