matplotlib做图中常用的辅助函数

matplotlib可能是Python语言中最常用的绘图库了，使用它可以较为容易的做出印刷品质的专业图形，此外，matplotlib的定制程度也很高，可以满足各式各样的绘图要求，能够限制你做图能力只是你的想像力而已。然而，也正因为matplotlib的定制性，相关API较为底层，所以为了做出复杂的图形，常常需要写很多行代码，显然不够经济。为了节省画图的时间，我们可以把日常工作中经常碰到的某些绘图需求加以定制化抽象化，编写一些辅助函数，这些函数将某些绘图需求打包封装了。这样，下次我们绘图里只需要调用相关函数就可以很快的做出需要的图形了，少写很多代码，也更加符合DRY(Don't Repeat Yourself)的原则。

在这篇文章里，我将对我在使用matplotlib绘图中编写的辅助函数做一个阶段性的总结，如果日后写了新的函数，我再进一步更新。

为了节省我们在绘图时的时间，我一般会在开始做图时，在jupyter notebook中做一些初始化设置，包括导入必须要的库，如numpy,matplotlib,seaborn等；设置图形的大小、风格等。你可以根据自己的需求对这部分配置进行修改，作为自己的初始化配置，下次绘图时可以直接使用。我的初始化配置如下：

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
import pandas as pd
import seaborn as sns
sns.set(style="white",color_codes=True)
plt.rcParams['figure.figsize'] = (15,9.27)
# Set the font set of the latex code to computer modern
matplotlib.rcParams['mathtext.fontset'] = "cm"
```

一、绘图时使用latex

matplotlib支持在绘图时使用latex代码，这样便可以更好的支持在图形中加入数学公式。使用latex代码的方式也很简单，只需要在相应的字符串两边各加上一个$号，如$y=e^x$就可以显示为$y = e^x$。但是每次都要输入$号比较麻烦，我编写了一个辅助函数可以自动为字符串两端加上$，这样只需求调用函数，正常输入latex代码或者普通文本，就可以用latex公式的形式呈现了。如下：

```
def latex(s=''):
    n = len(s.split(' '))
    if n == 1:
        if s == '':
            return '$'+'\ '+'$'
        else:
            return '$'+s+'$'
    else:
        return '$'+'\ '.join(s.split(' '))+'$'
```

二、绘制单个函数图像

在科学计算中，经常会碰到的一个情形是需要绘制函数图像，函数图像可以让我们更加直观地了解到我们关心的函数的性质，比如单调性或者凹凸性等，这为我们之后进一步分析函数性质提供了指导。在matplotlib中为了绘制函数图像，首先需求指定一个自变量的一个取值区间，然后需要调用numpy中的`linspace`函数在区间内产生一系列离散点。使用`numpy.arange`函数也可以产生类似的效果，只需要指定超始值、结束值与步长，如果步长越短，则取的离散点越多，则函数会越平滑。

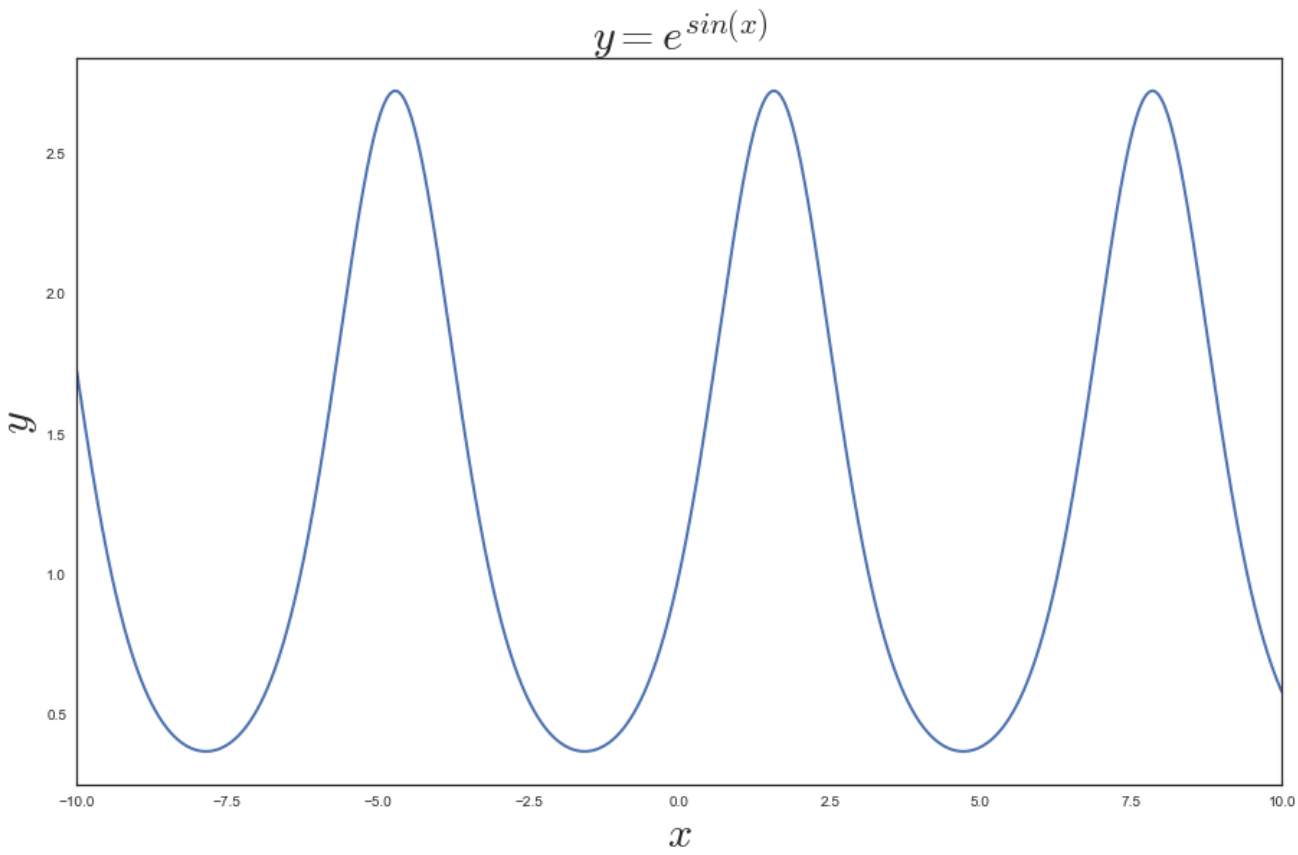
在调用函数时，我们只需要指定想要绘制的函数(通常可以`lambda`表达式去定义)，绘制函数的取值区间。其它设置包括X轴，Y轴与图片标题，字体大小以及线宽，采用默认设置或者自己修改均可。

```
def fplot(func,beg,end,lw=2,xlab='x',ylab='y',title='',fs=30):
    x = np.arange(beg,end+0.001,0.001)
    y = np.array(list(map(func,x)))
    plt.plot(x,y,linewidth=lw)
    plt.xlim(beg,end)
    plt.xlabel(latex(xlab),fontsize=fs)
    plt.ylabel(latex(ylab),fontsize=fs)
    plt.title(latex(title),fontsize=fs)
```

注意*fplot*函数调用了之前定义的*latex*函数，现在我们试试用它来绘制一个函数$y = e^{sin(x)}$：

```
from math import exp,sin

f = lambda x: exp(sin(x))
fplot(f,-10,10,title='y=e^{sin(x)}')
```



可以看到通过一行简单的代码就可以绘制出想要的函数图形，绘制效果也较为满意。你还可以在*fplot*函数之后再加上一些matplotlib中的定制命令，如X与Y轴的取值范围等，从而可以进一步地优化图形。

三、绘制多个函数图像

有时候我们也需要在同一个图形中绘制多个函数图像，一个个的去绘制当然比较麻烦，所以我们也可能编写一个同时绘制多个函数图像的函数。为了更好的区分多个函数，除了不同函数曲线使用不同颜色外，我们也可以使用不同的线型，从而可以看得更清楚，哪怕在黑白打印时也不会影响识别的效果。除此之外，我们还可以为每个函数加上标签，这样才能看出不同颜色与线型分别对应那个函数。

```
def fplots(funcList,beg,end,legendList,loc=2,legendsize=20,lw=2):
    x = np.arange(beg,end+0.001,0.001)
    style = ['solid','dashed','dashdot','dotted']
    styleList = style[:len(funcList)]
    ldlist = ['$'+x+'$' for x in legendList]
    for func,label,linestyle in zip(funcList,ldlist,styleList):
        y = [func(i) for i in x]
        plt.plot(x,y,linewidth=lw,label=label,linestyle=linestyle)
    plt.legend(loc=loc,frameon=True,fontsize=legendsize)
```

*fplots*函数的参数有七个，前四个是必要参数，后三个是可选参数，分别表示的意思如下：

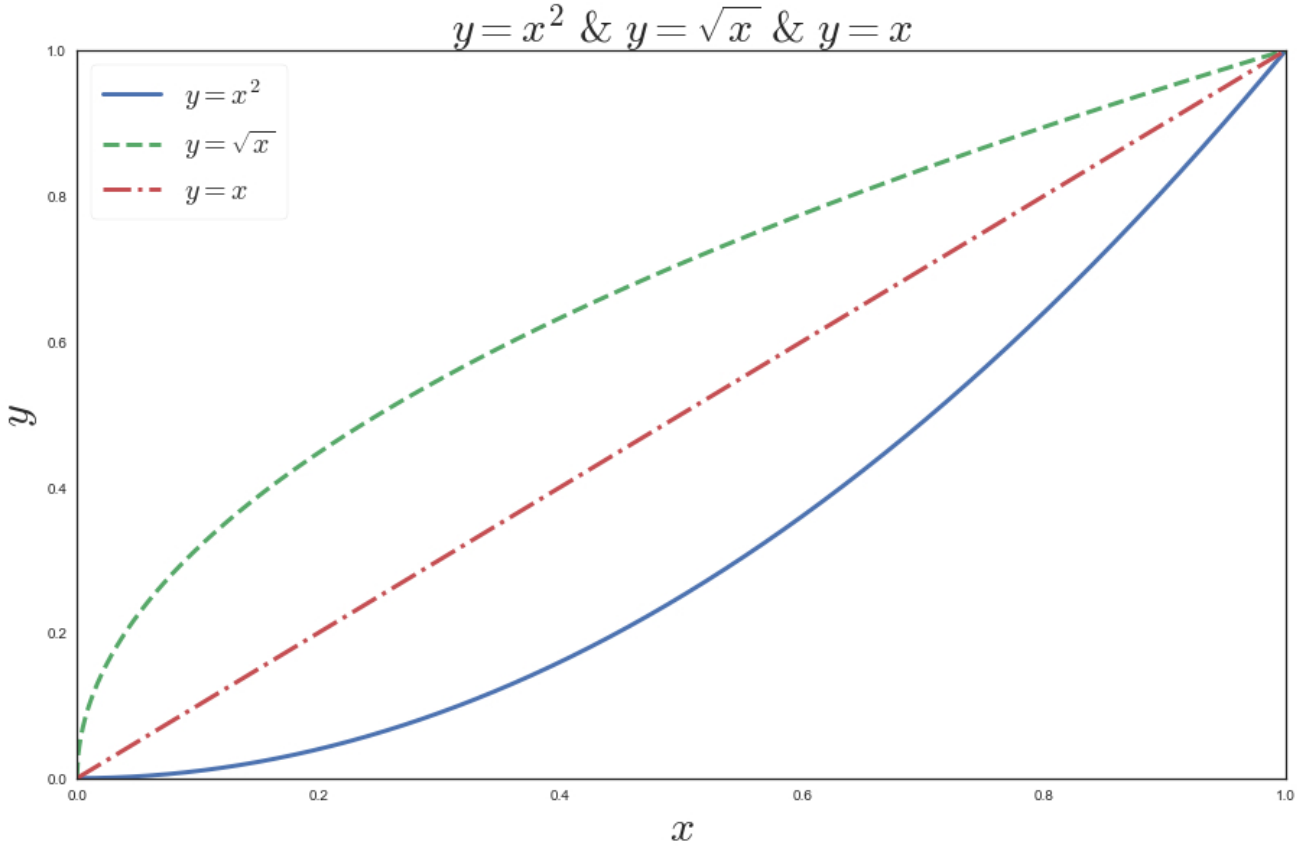
- funcList:需要绘制的函数的列表；
- beg：绘制函数的左端点
- end: 绘制函数的右端点
- legendList: 函数标签的列表，顺序应与函数列表的顺序一致
- loc: 标签显示的位置，默认为2，即左上角；1，3，4分别代表右上角、左下角与右下角
- legendsize: 标签中文字的大小，默认为20
- lw: 线形的宽度，默认为2

下面，我们看一个示例：

```
from math import sqrt

f = lambda x:x**2
g = lambda x:sqrt(x)
h = lambda x:x

fplots([f,g,h],0,1,['y=x^2','y=\sqrt{x}','y=x'],lw=3)
plt.xlim(0,1)
plt.ylim(0,1)
plt.xlabel(latex('x'),fontsize=30)
plt.ylabel(latex('y'),fontsize=30)
plt.title(latex('y=x^2 & y=\sqrt{x} & y=x'),fontsize=30)
```



从以上示例我们可以看到，我们可以绘制函数后再加上其它matplotlib中其它的绘图命令来对函数图形进行修改调整，比如添加标签，定义X与Y轴的取值范围等。

四、为图形添加平滑

在使用matplotlib绘图时有时会碰到图像不够平滑的问题，这虽然不影响绘制的准确性，但有时显得不够美观。因此，我们不影响绘制准确性的的情况下，为图形添加一定的平滑。我本以为matplotlib库中应该有这个选项，正如excel绘图时可以添加平滑性一样，但是怎么找都没有找到。经过搜索网络才知道有很多人也问了类似的问题，但matplotlib库中确实没有这项功能，不过可以通过自己编写函数来较为容易的实现，实现的方法参考了stackoverflow上的这个回答。

实现的原理大致是这样的，使用三次样条插值，它的基本思想是，在由两相邻节点所构成的每一个小区间内用低次多项式来逼近，并且在各结点的连接处又保证是光滑的(即导数连续)。`scipy`库有样条插值的相关函数，直接调用即可，不需要自己去实现，具体见以下代码：

```
def smooth_plot(x,y,area=False,color='b',ls='solid',lw=1):
    from scipy.interpolate import spline
    x_np,y_np = np.array(x),np.array(y)
    xnew = np.linspace(x_np.min(),x_np.max(),300)
    y_smooth = spline(x_np,y,xnew)
    if area == True:
        plt.fill_between(xnew,y_smooth,color=color)
    else:
        plt.plot(xnew,y_smooth,color=color,linestyle=ls,linewidth=lw)
```

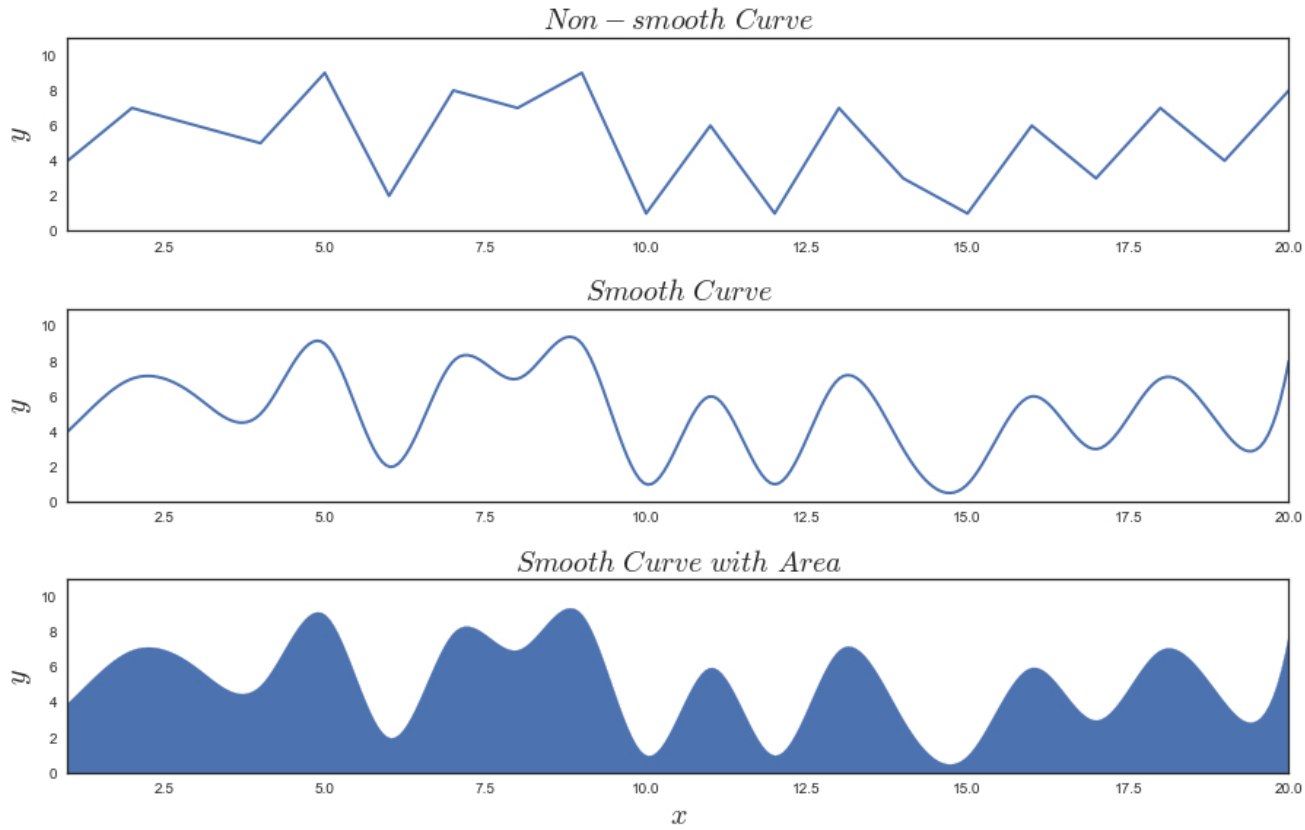
函数包含六个参数，其中两个必选参数，四个可选参数，含义分别如下：

- x: 自变量；
- y: 因变量；
- area: 是否绘制为面积图，默认为False，即不绘制；
- color: 函数图像的颜色，默认为蓝色；
- ls: 线型，默认为实线；
- lw：线宽，默认为1

下面我们看一下示例，分别绘制没有添加平滑和添加平滑后的图形：

```
np.random.seed(1234)
x = np.arange(1,21)
y = np.random.randint(1,10,20)

plt.subplot(311)
plt.plot(x,y,linewidth=2)
plt.xlim(1,20)
plt.ylim(0,11)
plt.ylabel(latex('y'),fontsize=20)
plt.title(latex('Non-smooth Curve'),fontsize=20)
plt.subplot(312)
smooth_plot(x,y,lw=2)
plt.xlim(1,20)
plt.ylim(0,11)
plt.ylabel(latex('y'),fontsize=20)
plt.title(latex('Smooth Curve'),fontsize=20)
plt.subplot(313)
smooth_plot(x,y,area=True)
plt.xlim(1,20)
plt.ylim(0,11)
plt.xlabel(latex('x'),fontsize=20)
plt.ylabel(latex('y'),fontsize=20)
plt.title(latex('Smooth Curve with Area'),fontsize=20)
subplots_adjust(hspace=0.4)
```



显然，第二张图形和第一张图形反映出的数据的趋势或走向是一样的，只是加了一些平滑性，所以看上去更加美观一些。第三张图形和第二张图形是一致的，只是换成了面积图，有时候可以通过这种方式变换一下风格。

五、结语

以上就是我现在主要用到的一些辅助函数，如果之后有添加，我还会继续更新这篇文章。

函数作为一种抽象与封装的方式，可以大大提高我们编写代码的效率，提高程序的健壮性与可维护性，建议大家可以把日常工作中重复使用的一些功能编写成函数，然后通过函数组合来实现更复杂的功能。虽然python并不是一个函数式语言，但这种函数式编程的思想还是可以应用的，这篇短文就是一个简单的示例。