第九讲
Matplotlib基础（II）
Lecture 9
Matplotlib Fundamentals (II)

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

坐标系统
Coordinate Systems

- 在matplotlib图形中，一般会有几个不同的坐标系随时共存。有与图形(FC)相关的，有与轴相关的（AC），也有与绘制(DC)相关的。此外，这些坐标系中还可能存在原始版本(xC)与归一化(NxC)的版本，列举如下：

  In matplotlib figure, there can be several different coordinate systems co-existing. Some is related to the figure (FC), some is related to the axes (AC), some is related to the individual plots (DC). some of these coordinate systems exist in native version (xC) or normalized (NxC) or as listed below:

  - # FC : 图形坐标系统（像素）， Figure coordinates (pixels)
  - # NFC : 归一化图形坐标系统（0 → 1）， Normalized figure coordinates (0 → 1)
  - # DC : 轴坐标系统（像素）， Axes coordinates (pixels)
  - # NAC: 归一化轴坐标系统（0 → 1）， Normalized Axes coordinates (0 → 1)
  - # DC : 数据坐标系统（特定数据单位）， Data coordinates (data units)
  - # NDC : 归一化图形坐标系统（0 → 1）， Normalized data coordinates (0 → 1)

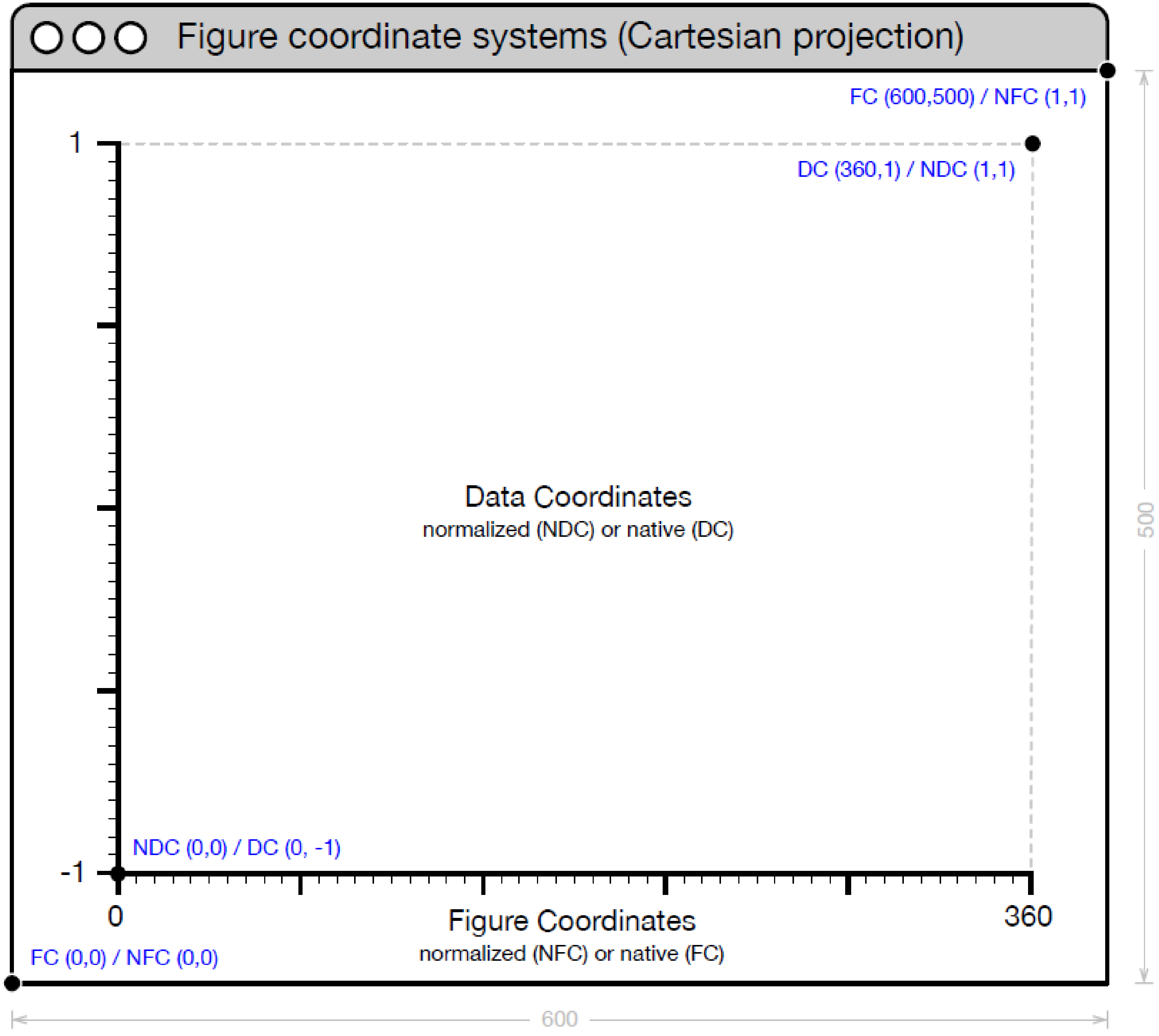
坐标系统
Coordinate Systems

- 为了将坐标从一个系统转换到另一个系统，matplotlib提供了一组变换函数。

  To convert a coordinate from one system to the other, matplotlib provides a set of transform functions.

- 我们先看常见的图形坐标系与数据坐标系共存于使用笛卡尔投影的图形中的情形：

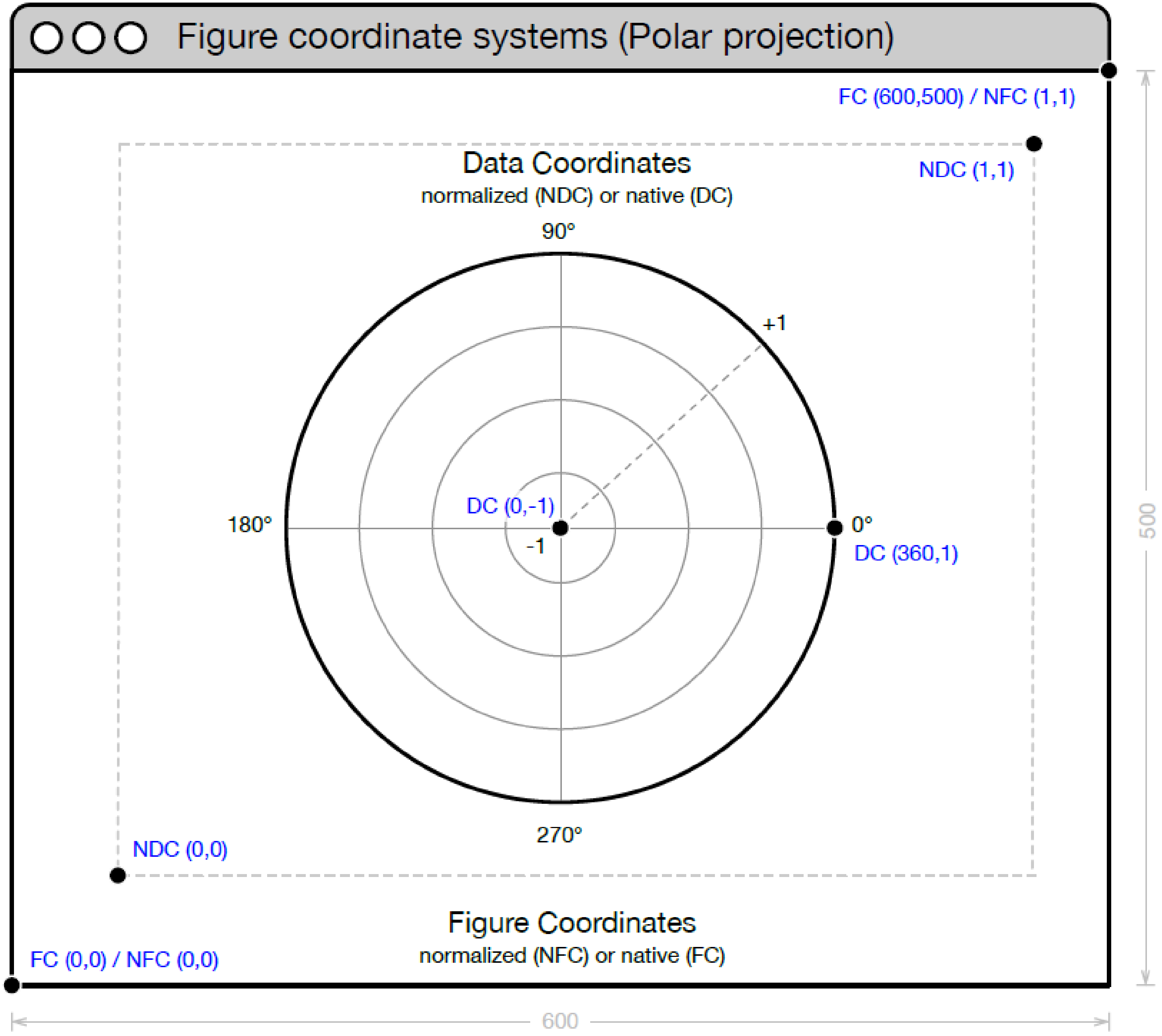
  The co-existence of figure coordinate system and data coordinate system within a figure using Cartesian projection:



坐标系统
Coordinate Systems

- 虽然在绝大多数情况下，都是处理使用笛卡尔坐标的情况，但在某些应用中需要考虑极坐标的情况。右图展示了图形坐标系与数据坐标系共存于使用极坐标投影的图形中的情形：

  In the majority of cases, we only consider Cartesian system for drawing figures, however, there exists applications using the polar system. The co-existence of figure coordinate system and data coordinate system within a figure using Polar projection is shown right:

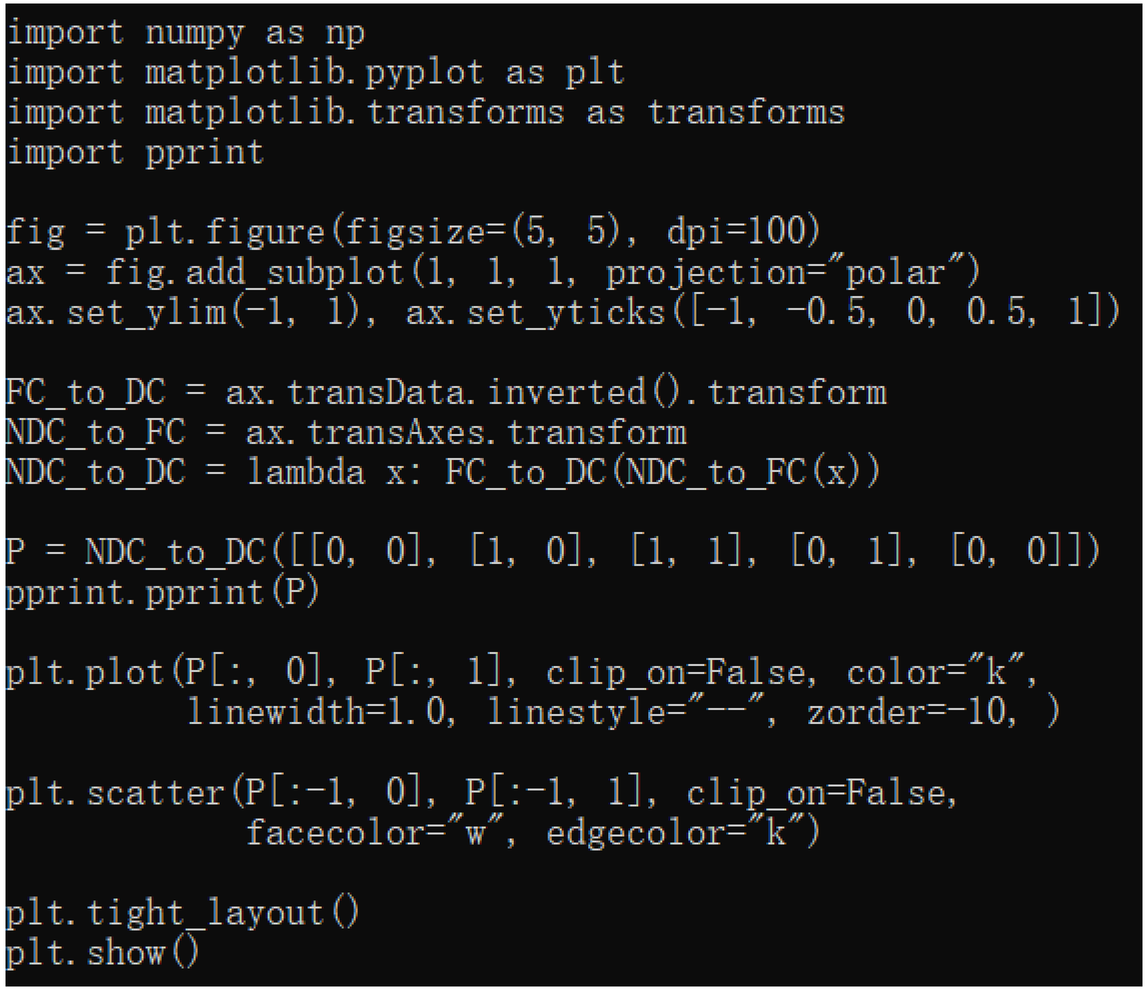


坐标系统
Coordinate Systems

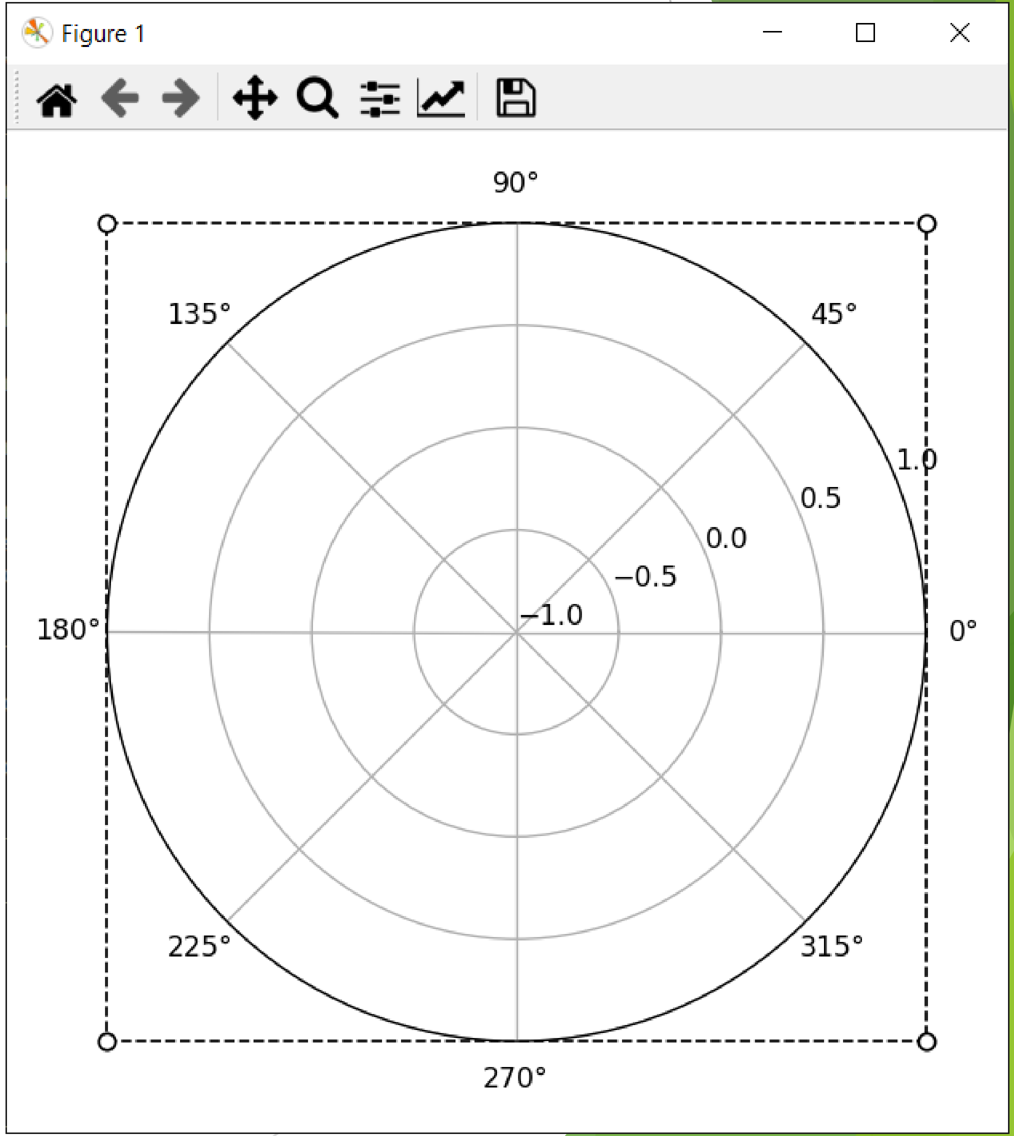
- 使用笛卡尔投影时，归一化数据坐标和原生数据坐标之间的对应关系非常清晰。使用其他类型的投影，即使事情也一样，但看起来却不那么明显。例如，考虑一个要绘制外轴边界的极坐标投影。在归一化数据坐标中，我们知道四个角的坐标为（0,0）、（1,0）、（1,1）和（0,1），然后我们可以将这些归一化的数据坐标转换回原生数据坐标并绘制边框。然而，还有一个额外的困难，因为这些坐标超出了轴的限制，因此我们需要告诉matplotlib，不用施加clip_on参数的限制。

  When using Cartesian projection, the correspondence is quite clear between the normalized and native data coordinates. With other kind of projection, things work just the same even though it might appear less obvious. For example, let us consider a polar projection where we want to draw the outer axes border. In normalized data coordinates, we know the coordinates of the four corners, namely (0,0), (1,0), (1,1) and (0,1). We can then transform these normalized data coordinates back to native data coordinates and draw the border. There is however a supplementary difficulty because those coordinates are beyond the axes limit, and we’ll need to tell matplotlib to not care about the limit using the clip_on arguments.

坐标系统
Coordinate Systems



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

fig = plt.figure(figsize=(5, 5), dpi=100)
ax = fig.add_subplot(1, 1, 1, projection="polar")
ax.set_ylim(-1, 1), ax.set_yticks([-1, -0.5, 0, 0.5, 1])

FC_to_DC = ax.transData.inverted().transform
NDC_to_FC = ax.transAxes.transform
NDC_to_DC = lambda x: FC_to_DC(NDC_to_FC(x))

P = NDC_to_DC([[0, 0], [1, 0], [1, 1], [0, 1], [0, 0]])
pprint.pprint(P)

plt.plot(P[:, 0], P[:, 1], clip_on=False, color="k",
         linewidth=1.0, linestyle="--", zorder=-10, )

plt.scatter(P[:-1, 0], P[:-1, 1], clip_on=False,
            facecolor="w", edgecolor="k")

plt.tight_layout()
plt.show()
```



坐标系统
Coordinate Systems

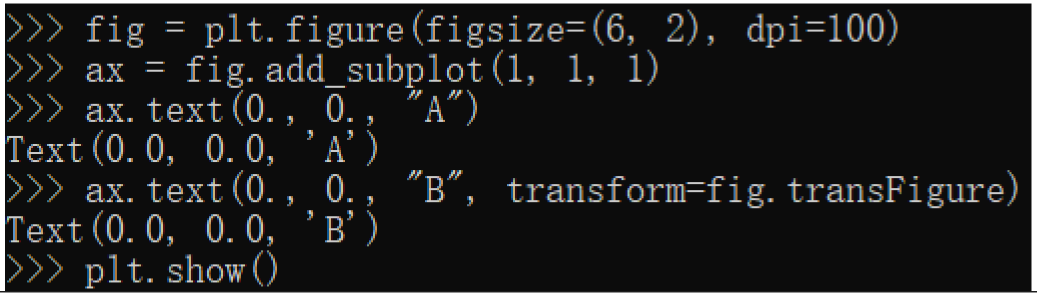
- 读者可能会问在一般情况下， matplotlib是如何考虑这些坐标的。例如，考虑想在特定绘图上添加一些文本例子，它们是以数据坐标为准？ 还是标准化数据坐标？ 还是标准化图形坐标？默认情况下考虑它们以数据坐标表示的。并且，大多数时候，不需要显式使用这些转换函数进行转化，而是隐式使用。因此，使用text函数时只需指定要写入的内容（当然）以及要显示文本的坐标。这时如果想使用不同的系统，则需要在调用函数时显式指定转换即可，而不用自己先行转化，再把转换后的坐标传递给text函数。

  Readers may ask in general, how matplotlib consider these coordinates? For example, consider the case where you want to add some text over a specific plot. Are they expressed in data coordinates? normalized data coordinates? normalized figure coordinates? The default is to consider they are expressed in data coordinates. And most of the time, you won’t need to use these transform functions explicitly but rather implicitly. Therefore, you just need to use the text function and to specify what is to be written (of course) and the coordinates where you want to display the text. If you want to us a different system, you’ll need to explicitly specify a transform when calling the function. You don’t need to convert the coordinate manually and then pass the converted coordinate to the text function.

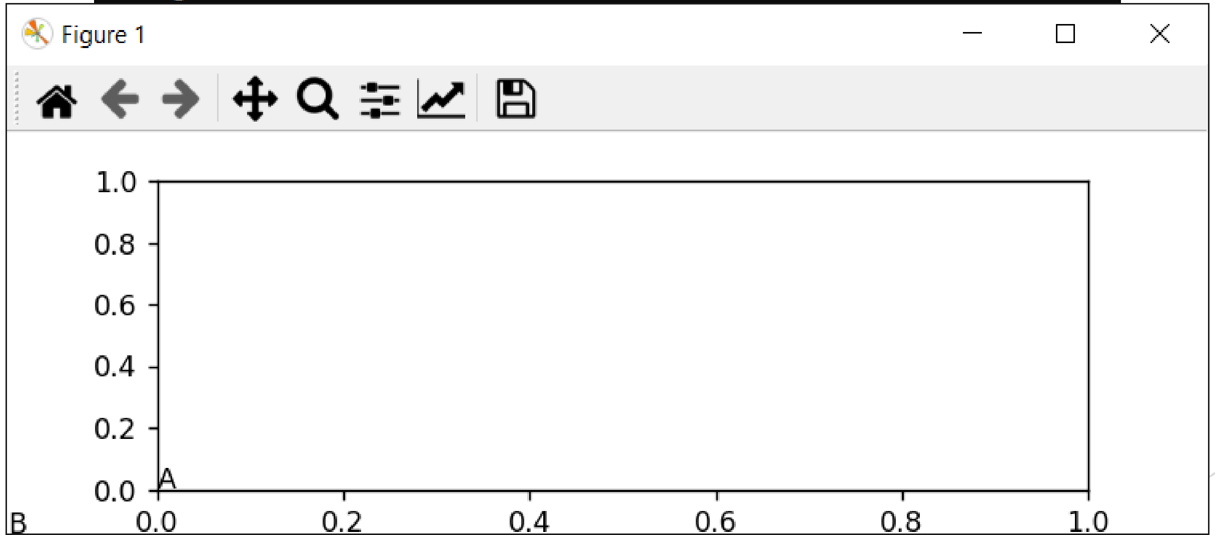
坐标系统
Coordinate Systems

- 下面例子我们展示了想在右下角添加一个字母的情况：

  The following example illustrates what we will do in order to add a letter on the bottom right corner.



```
>>> fig = plt.figure(figsize=(6, 2), dpi=100)
>>> ax = fig.add_subplot(1, 1, 1)
>>> ax.text(0., 0., "A")
Text(0.0, 0.0, 'A')
>>> ax.text(0., 0., "B", transform=fig.transFigure)
Text(0.0, 0.0, 'B')
>>> plt.show()
```

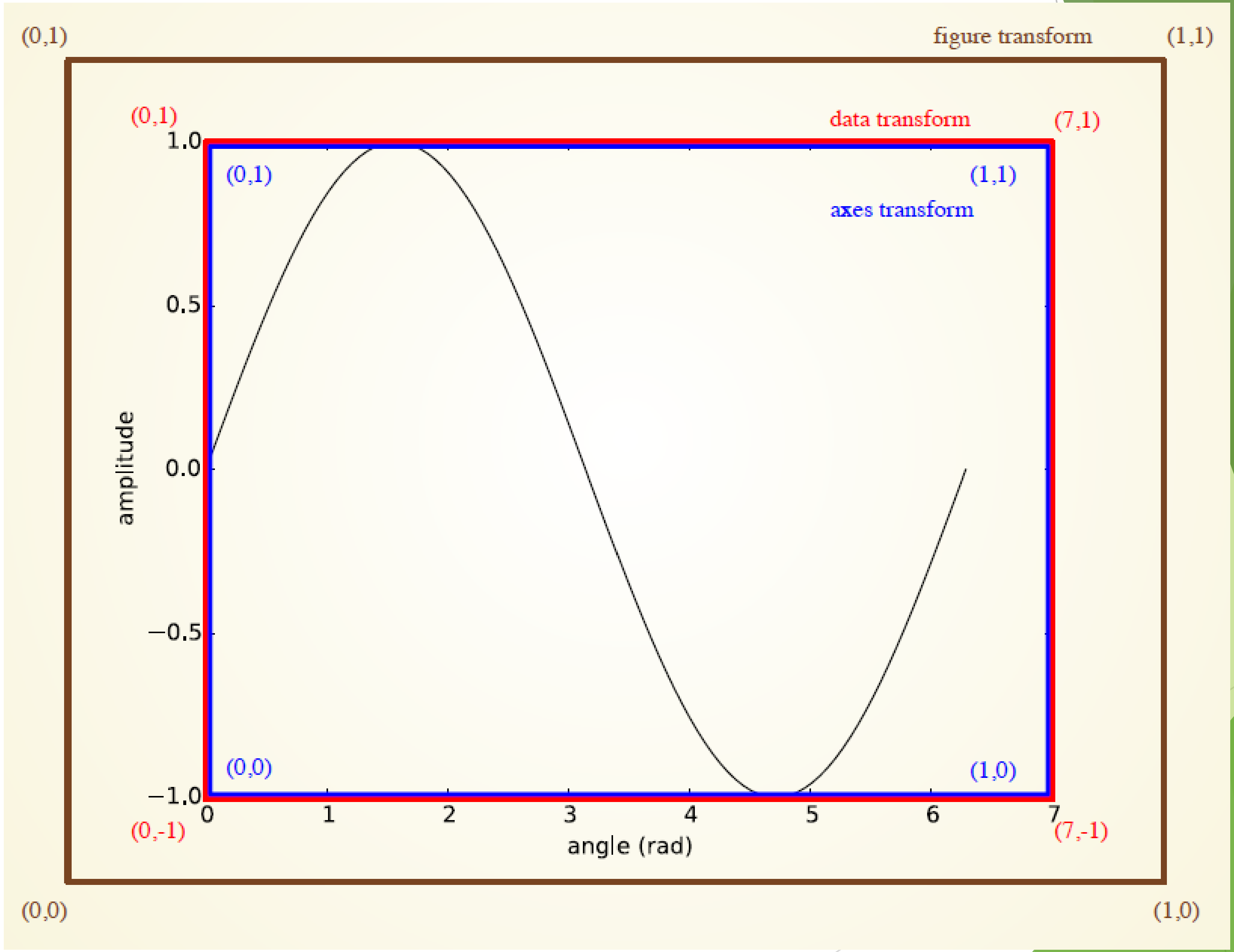


坐标系统
Coordinate Systems

- 下面我们展示轴坐标与其他坐标的关系。因为在绘制之前，不能预期单个坐标轴的刻度范围，因此轴坐标用正则化形式表达更加方便。举个例子，假设对一个某个值的分布图，想高亮在某一区间内值的分布情况，我们不能预期坐标轴的情况，只能假设在纵轴方向，高亮范围始终跟随纵轴变化。因此，我们可以指定高亮区域的高为1，这个1表明要始终与纵轴高度一致。不过，在此之前，先来了解一下如下图中所示的这几个坐标轴之间的关系。

  We continue to dissect the relationship between axes coordinate and other coordinate systems. Usually, it is impractical to predict the range or limits of a particular axis, to use axes coordinates in the normalized form is preferred. For example, suppose we draw a histogram of some value and simultaneously want to create a horizontal span which highlights some region along y-axis and spans across the x-axis regardless of the data limits, pan or zoom level, etc. Due to the difficulty to anticipate the situation of axes, we assume the highlighted range fills the entire height vertically and self-adjust to y-axis harmonically. For this purpose, the height of the range can be set to 1, which indicates it is always equal to the limit of y-axis. But before that, let’s have a glimpse of the relations between various coordinate systems.

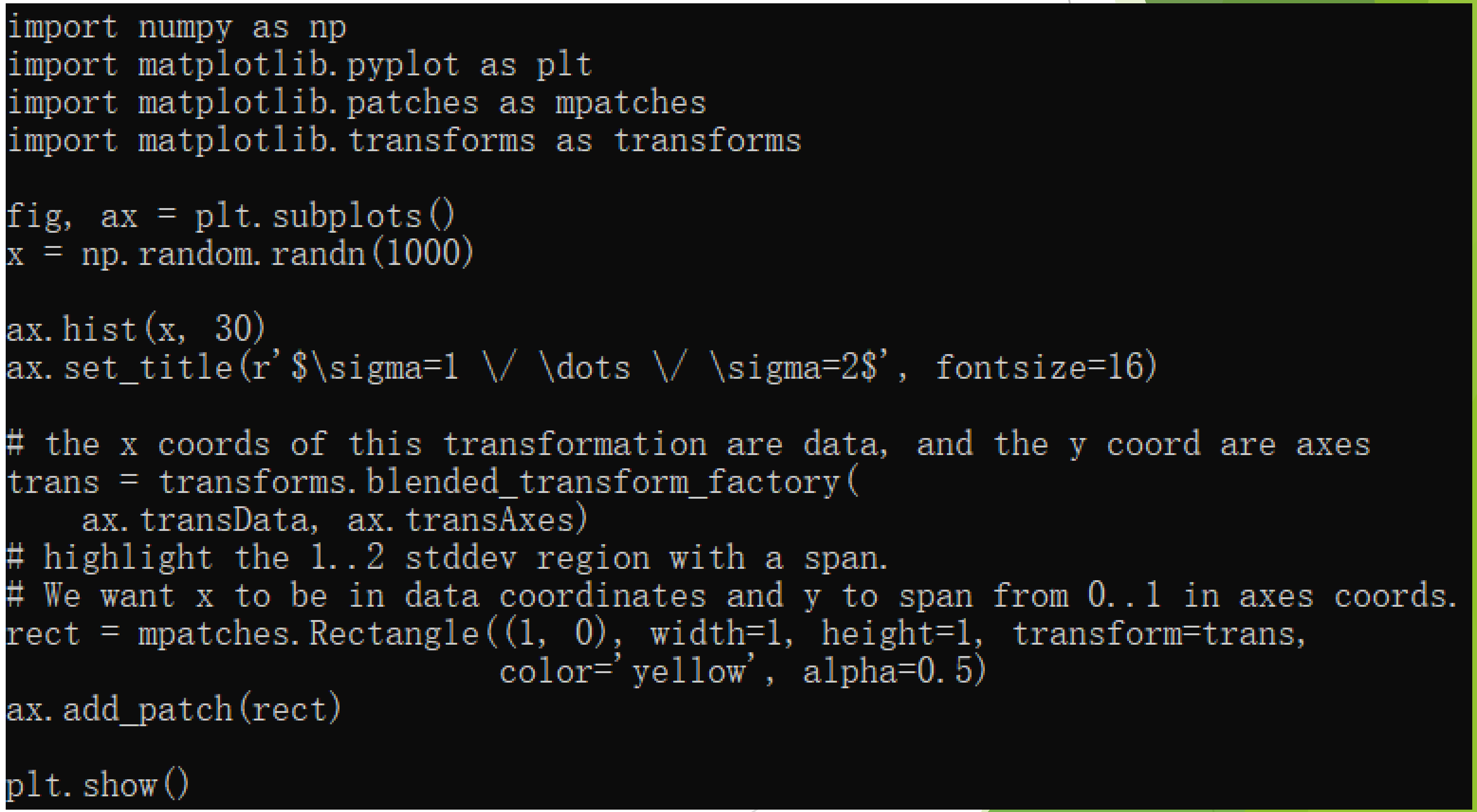
坐标系统 Coordinate Systems



坐标系统
Coordinate Systems

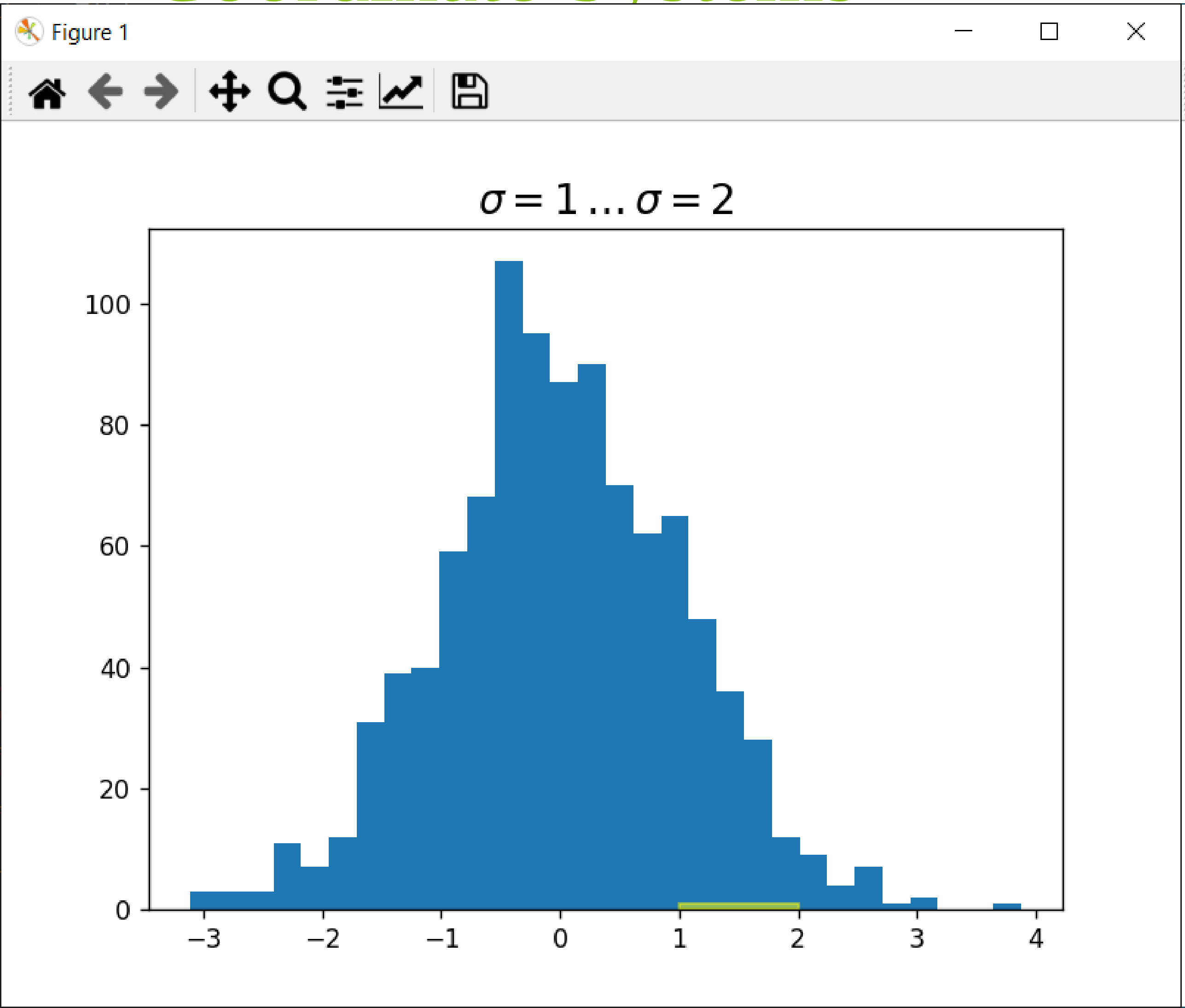
- 下面的代码展示了上面的例子，不过在这个代码中，有一个重要的手段叫混合变换。具体到本例，即*x*坐标的值对应于数据坐标系，而*y*坐标的值对应于轴坐标系。混合不同的坐标空间的方法在实际中非常有用。

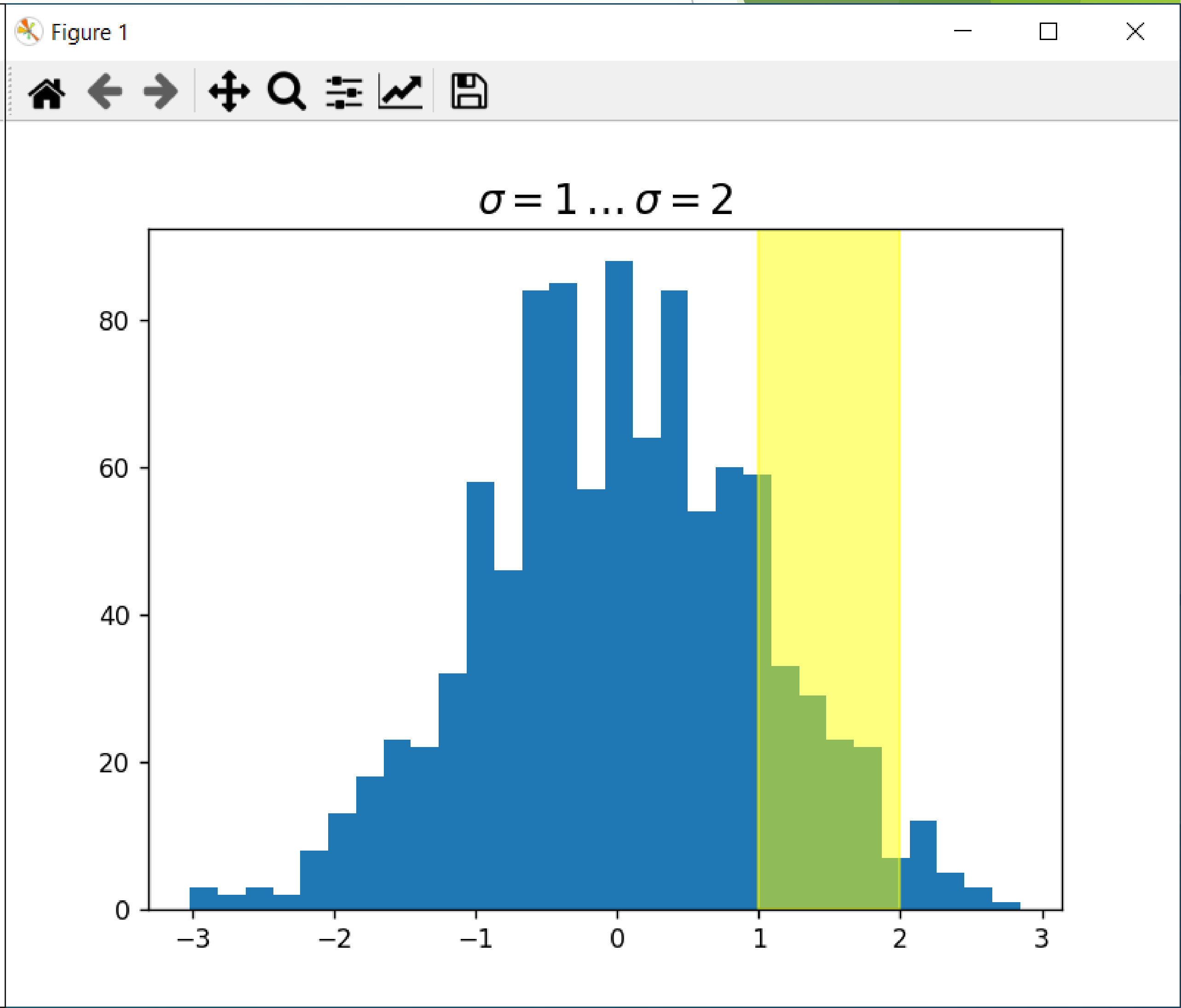
  The following code implements the above example. However, in this example, one important operation is called blended transformation. As for this example, the *x* component corresponds to the data coordinate, and *y* component corresponds to the axes coordinate. Drawing in blended coordinate spaces which mix axes with data coordinates is extremely useful.



```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.transforms as transforms

fig, ax = plt.subplots()
x = np.random.randn(1000)

ax.hist(x, 30)
ax.set_title(r'$\sigma=1 \/ \dots \/ \sigma=2$', 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 = mpatches.Rectangle((1, 0), width=1, height=1, transform=trans,
                          color='yellow', alpha=0.5)
ax.add_patch(rect)

plt.show()
```

坐标系统
Coordinate Systems

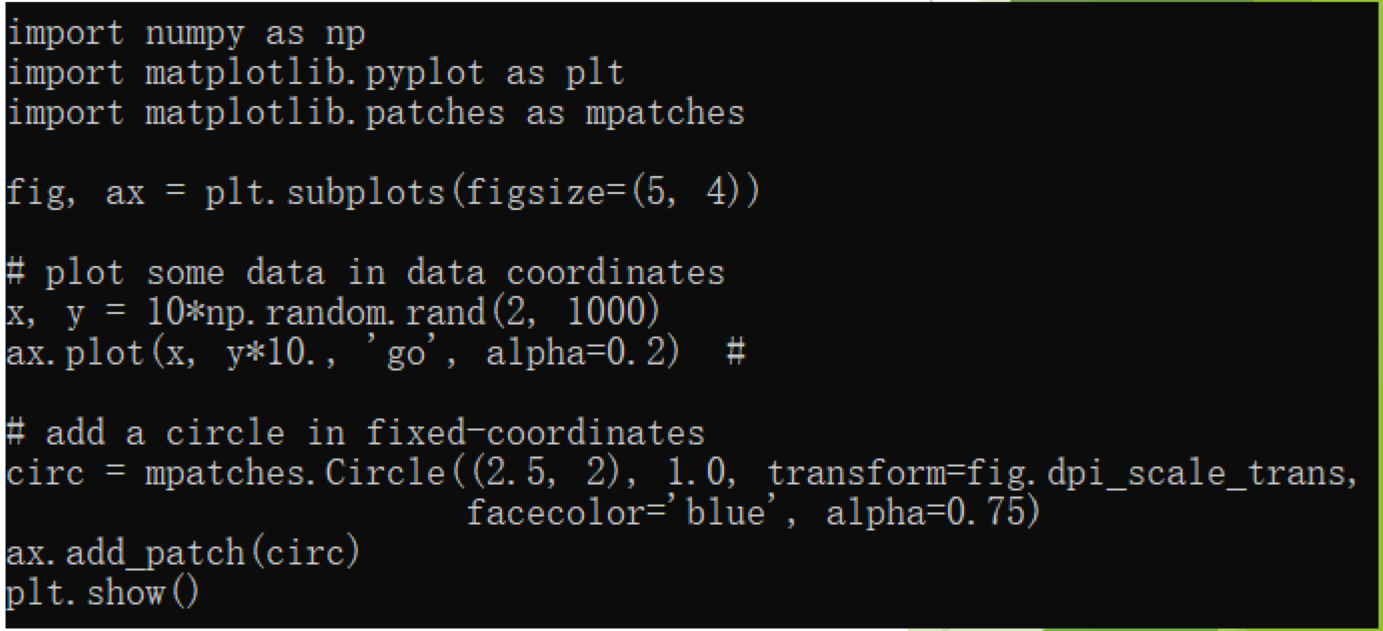




坐标系统 Coordinate Systems

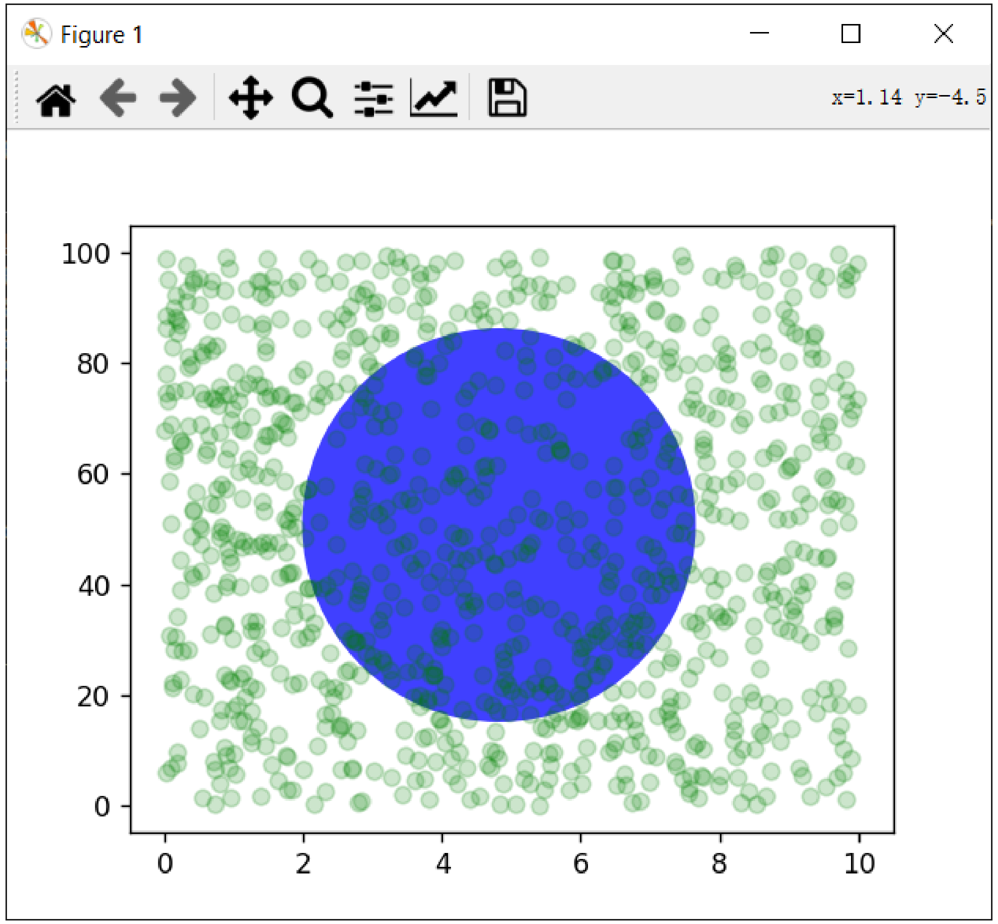
- 有时我们希望对象以一定的物理尺寸绘制。在这个例子中，我们在物理坐标中绘制一个圆。如果以交互方式查看，可以看到更改图形的大小不会更改圆从左下角的偏移量，也不会更改其大小，并且无论轴的纵横比如何，圆仍然是一个圆 。

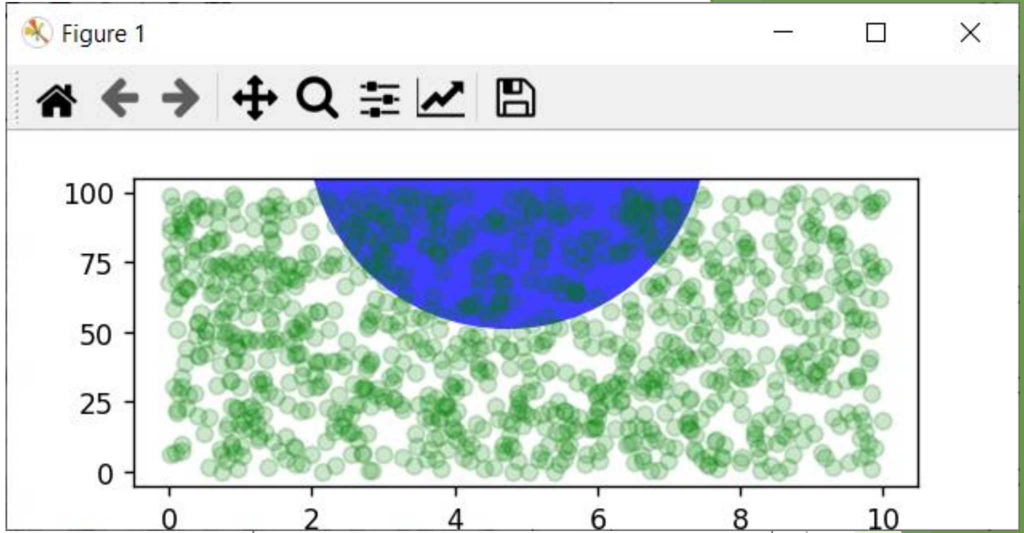
  Sometimes we want an object to be a certain physical size on the plot. Here we draw a circle in physical coordinates. If done interactively, you can see that changing the size of the figure does not change the offset of the circle from the lower-left corner, does not change its size, and the circle remains a circle regardless of the aspect ratio of the axes.

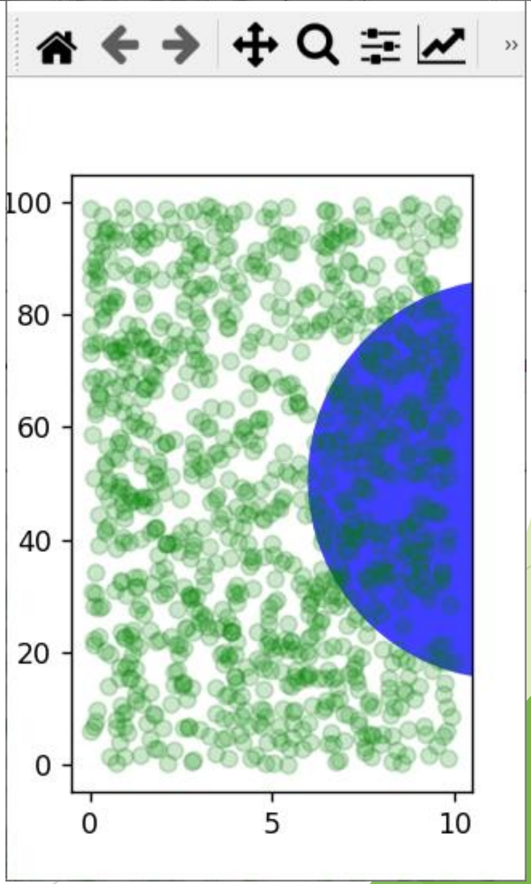


```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

fig, ax = plt.subplots(figsize=(5, 4))

# plot some data in data coordinates
x, y = 10*np.random.rand(2, 1000)
ax.plot(x, y*10., 'go', alpha=0.2)  #

# add a circle in fixed-coordinates
circ = mpatches.Circle((2.5, 2), 1.0, transform=fig.dpi_scale_trans,
                       facecolor='blue', alpha=0.75)
ax.add_patch(circ)
plt.show()
```

坐标系统
Coordinate Systems







坐标系统
Coordinate Systems

- 数据坐标与轴坐标或图形坐标的转换，即不同坐标系的转换，只是线性变换的一种。线性变换还有好多，如平移和旋转等。虽然您在日常应用中不会经常使用变换操作，但变换无疑是非常强大的工具。 在少数情况下，了解它们会带来意想不到的效果。我们下面通过一些例子作进一步介绍，当然读者也可以通过matplotlib 网站上的变换教程进一步了解转换和坐标。

  The transform between data coordinates and figure coordinates, or transform between different coordinates, is just one case of the transformations. Besides that, linear transformations include translation, rotation, etc. Even though you won’t manipulate them too often in your daily applications, transformations are quite powerful tools. So, there are some few cases where you’ll be happy to know about them. Then You can read further on transforms and coordinates with the Transformation tutorial on the matplotlib website.

坐标系统
Coordinate Systems

- 首先我们以一个简单的例子来介绍一下平移，其具有如下形式

  By a simple example, we first introduce the translation, which is defined as below:

  trans = ScaledTranslation(x_t, y_t, scale_trans)

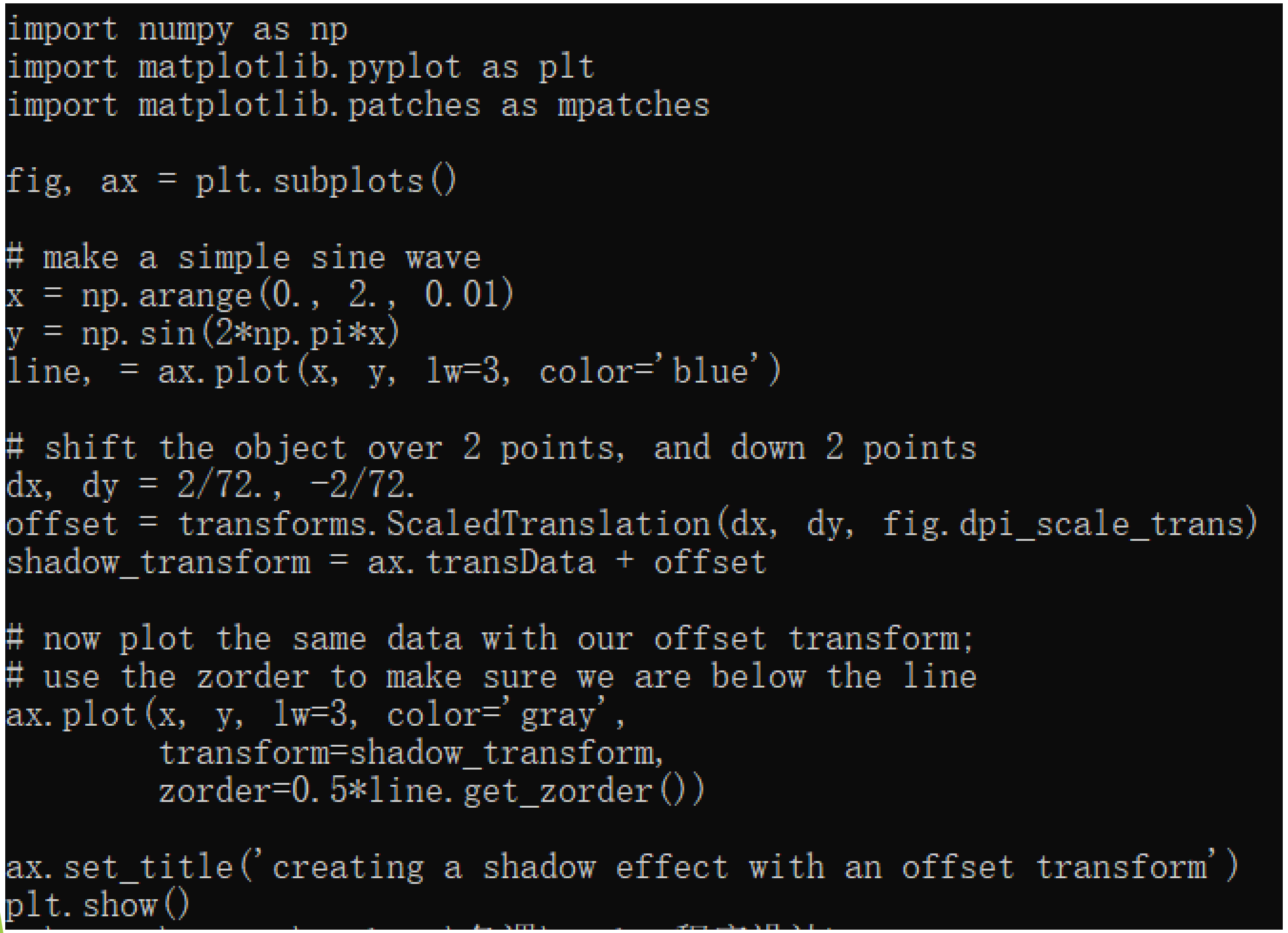
  其中x_t和y_t是平移偏移量，而scale_trans是在应用偏移之前对x_t和y_t进行调整的缩放操作。

  where x_t and y_t are the translation offsets, and scale_trans is a transformation which scales x_t and y_t at transformation time before applying the offsets.

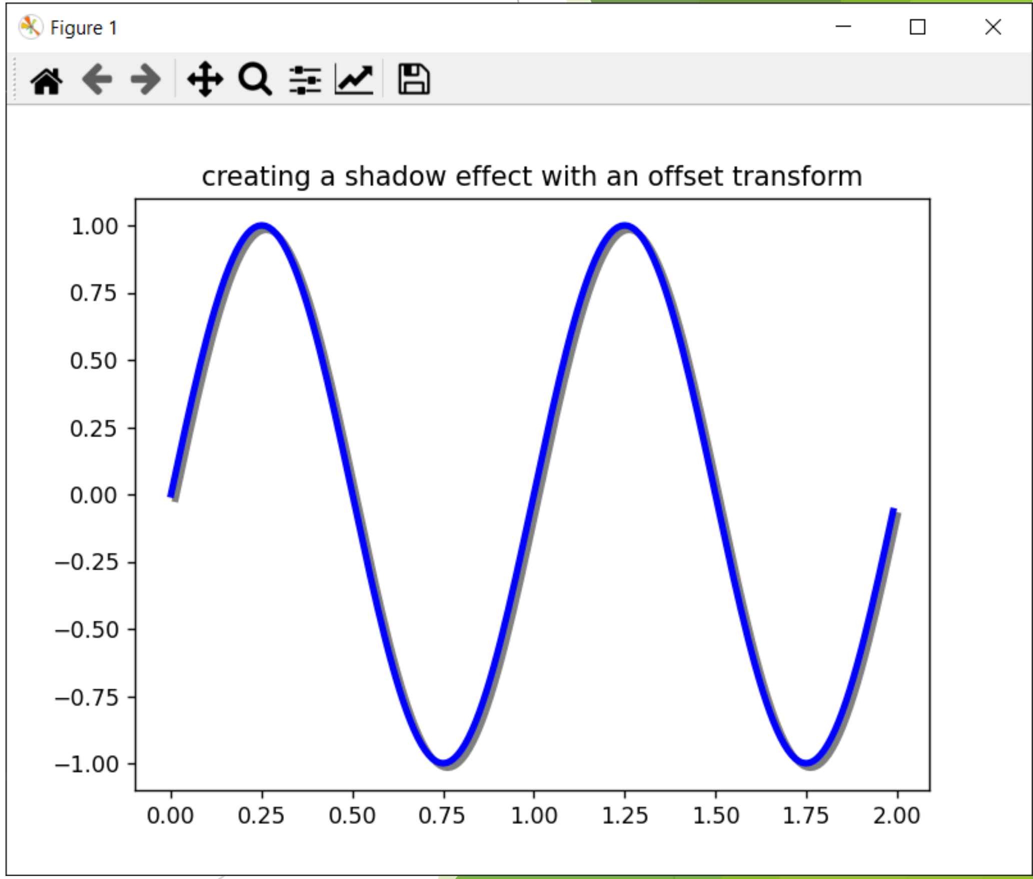
- 我们要考虑的例子是给所绘制的一条曲线添加阴影效果。一个简单的做法即将原曲线平移合适的位置重绘即可。

  The example is to add shadow effect to a plotted curve. A simple solution is to offset the original plot to some place and redraw.

坐标系统
Coordinate Systems



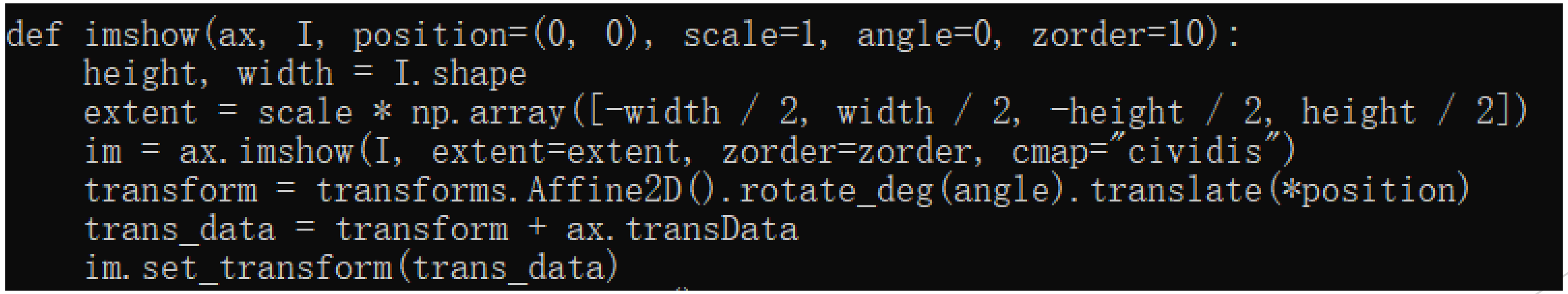
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

fig, ax = plt.subplots()

# 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()
```



坐标系统
Coordinate Systems

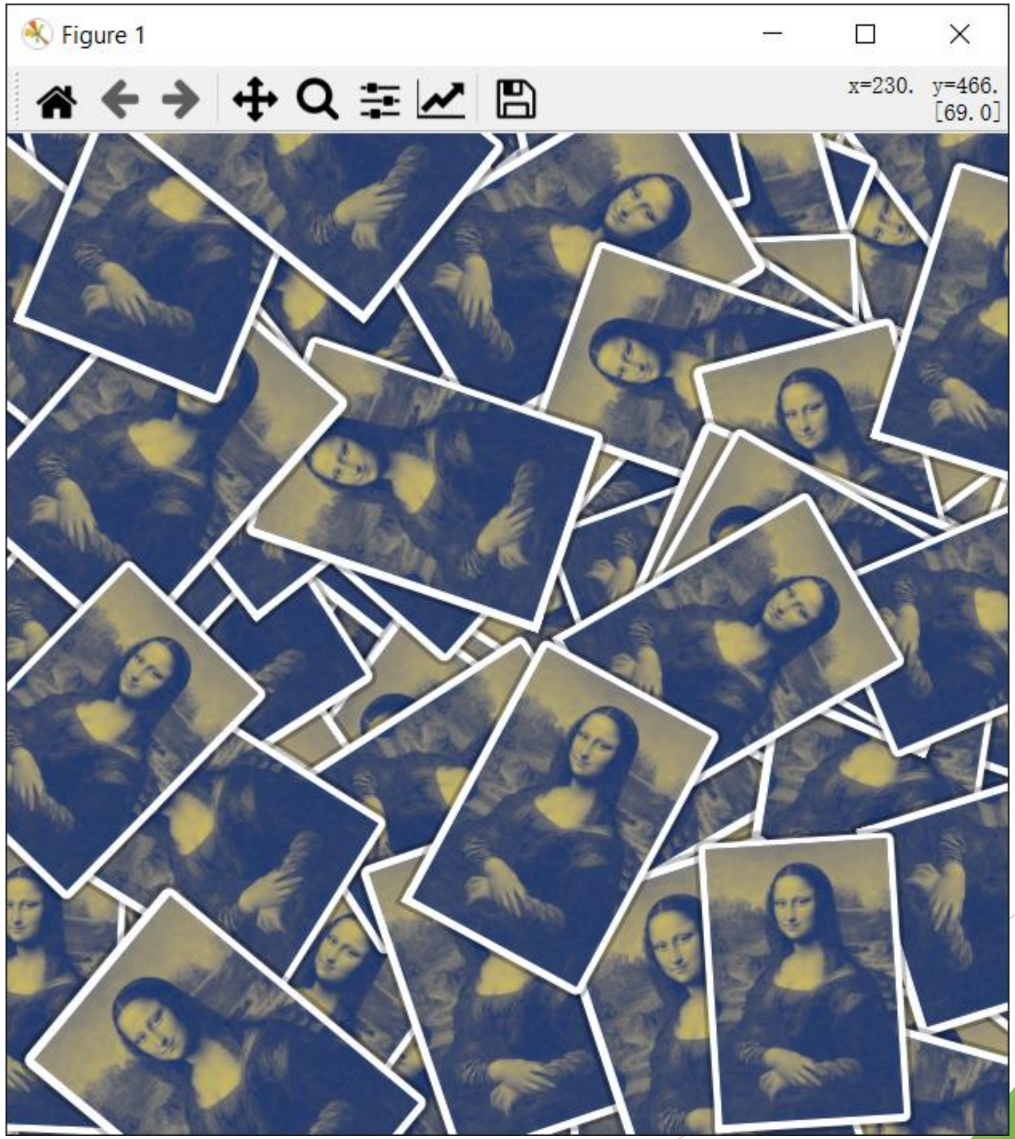
- 对于旋转变换，matplotlib中将其作为仿射变换类的一个方法。实例化一个仿射变换对象，调用该方法并直接指定需要旋转的角度即可。下例展示了如何利用旋转与平移，实现图片的堆叠效果:

  For rotation, matplotlib provide it as a method of the affine transformation class. The user just needs to instantiate an affine transform object to call the rotation method with specified degree. The following example illustrates how to combine rotation and translation to realize the collage effect of an image.



```
def imshow(ax, I, position=(0, 0), scale=1, angle=0, zorder=10):
    height, width = I.shape
    extent = scale * np.array([-width / 2, width / 2, -height / 2, height / 2])
    im = ax.imshow(I, extent=extent, zorder=zorder, cmap="cividis")
    transform = transforms.Affine2D().rotate_deg(angle).translate(*position)
    trans_data = transform + ax.transData
    im.set_transform(trans_data)
```

坐标系统
Coordinate Systems



坐标系统
Coordinate Systems

- 虽然您在日常应用中不会经常使用转换操作，但转换操作无疑是非常强大的工具。在少数情况下，了解它们会带来意想不到的效果。可以通过 matplotlib 网站上的转换教程进一步了解转换和坐标。

  Transformations are quite powerful tools even though you won’t manipulate them too often in your daily life. But there are some few cases where you’ll be happy to know about them. You can read further on transforms and coordinates with the Transformation tutorial on the matplotlib website.