第十讲
Matplotlib基础（III）
Lecture 10
Matplotlib Fundamentals (III)

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

缩放
Scales

- 除了上一讲所讲的仿射变换外，matplotlib还提供了其它的高级形式的变换，允许在不修改数据的情况下改变数据的表示。这些变换对应于数据预处理阶段，允许根据数据的性质调整绘制。实际上，我们根据变换的特点，可以将matplotlib中的变换分为两类：一类是独立变化或可分变换，其作用于单个维度，称为缩放；不可分变换或集成变换，其同时作用于两个或多个维度的数据，称为投影。

  Beyond the affine transforms in the previous lecture, matplotlib also offers other advanced transformations that allows to drastically change the representation of your data without ever modifying it. Those transformations correspond to a data preprocessing stage that allows you to adapt the rendering to the nature of your data. As explained in the matplotlib documentation, there are two main families of transforms: separable transformations, working on a single dimension, are called Scales, and non-separable transformations, that handle data in two or more dimensions at once are called Projections.

缩放
Scales

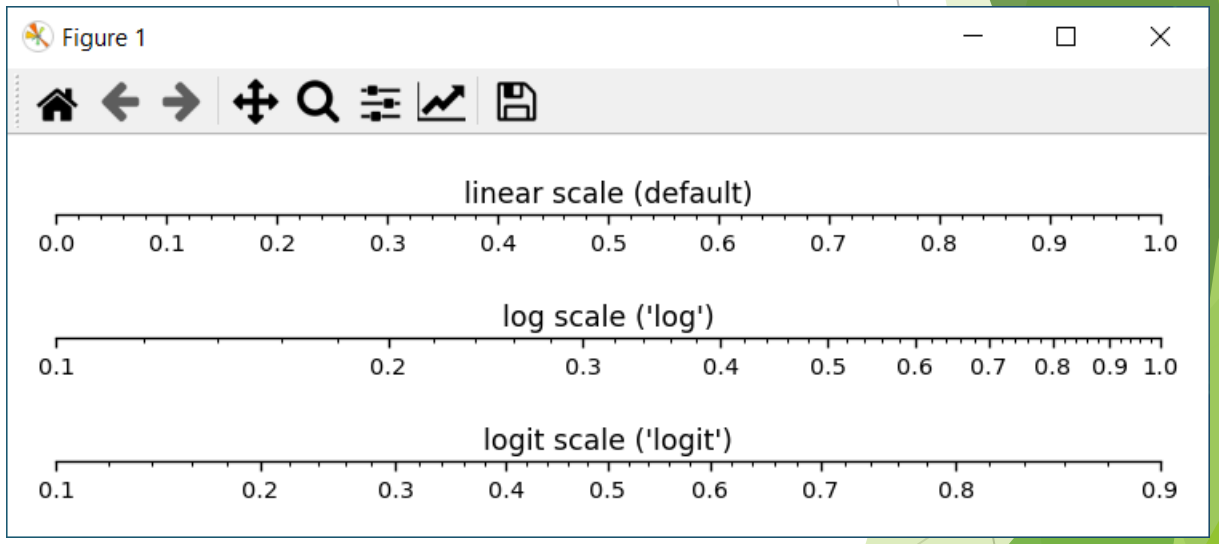
- 缩放提供了数据及其在图中沿给定维度的表示之间的映射机制。Matplotlib提供了四种不同的比例（线性、对数、符号对数和对数），并负责对每一种比例进行修改，例如调整刻度位置和标签。请注意，比例可以仅应用于x轴(set_xscale)、y轴(set_yscale)或两者。默认（和隐式）比例是线性的，因此通常不需要指定任何方式。可以通过比较图形坐标中三个点之间的距离来检查比例是否是线性的，同时，可以通过进一步检查这些点在数据坐标中的差值，与这些点在图形坐标中的差值模去一个值是否一致来确认。

  Scales provide a mapping mechanism between the data and their representation in the figure along a given dimension. Matplotlib offers four different scales (linear, log, symlog and logit) and takes care, for each of them, of modifying the figure such as to adapt the ticks' positions and labels. Note that a scale can be applied to x axis only (set_xscale), y axis only (set_yscale) or both. The default (and implicit) scale is linear, and it is thus generally not necessary to specify anything. You can check if a scale is linear by comparing the distance between three points in the figure coordinates. And this can be further confirmed by double checking whether their difference in data space is the same as in figure space modulo a given factor.

缩放
Scales

- 首先来介绍一下对数刻度(log)，其是一种非线性标度，每个区间不是以相等的增量增加，而是增加对数底的一个因子。对数刻度用于严格为正的值，因为对于负值和空值未定义对数。

  Logarithmic scale (log) is a nonlinear scale where, instead of increasing in equal increments, each interval is increased by a factor of the base of the logarithm (hence the name). Log scales are used for values that are strictly positive since the logarithm is undefined for negative and null values.



缩放
Scales

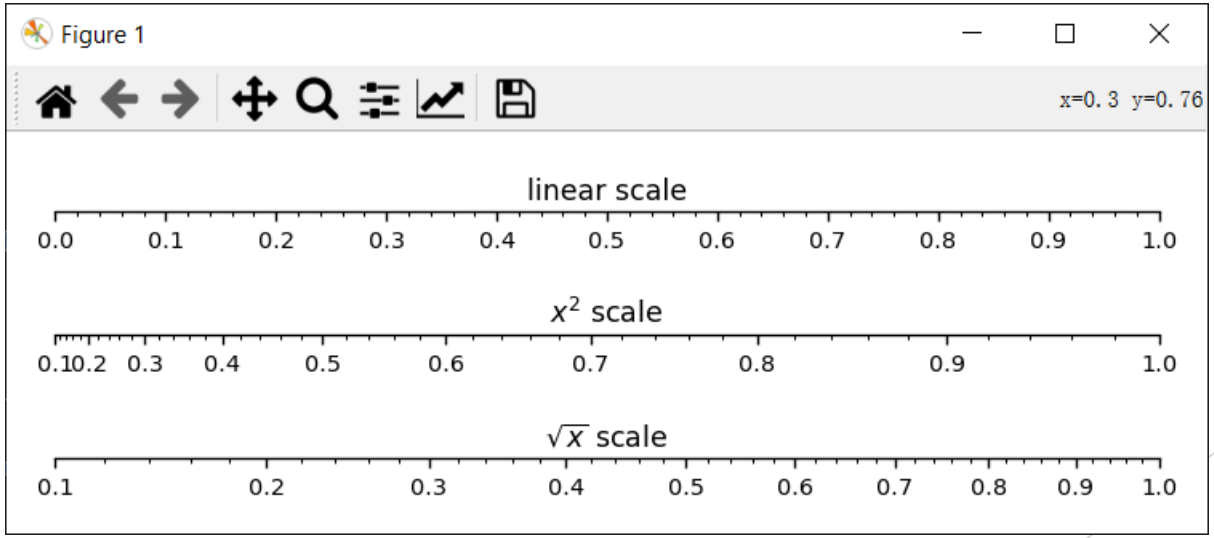
```
# Log axis
# ------------------------------------------------------------------
ax = plt.subplot(3, 1, 2, xlim=[0.1, 1.0], ylim=[0, 1])
ax.patch.set_facecolor("none")
ax.text(
    0.5,
    0.1,
    "log scale ('log')",
    transform=ax.transAxes,
    horizontalalignment="center",
    verticalalignment="bottom",
)
ax.set_xscale("log")
ax.set_yticks([])
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.spines["top"].set_visible(False)
ax.xaxis.set_major_locator(ticker.MultipleLocator(0.10))
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter("%.1f"))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.02))
ax.xaxis.set_minor_formatter(plt.NullFormatter())
```

```
# Logit axis
# ------------------------------------------------------------------
ax = plt.subplot(3, 1, 3, xlim=[0.1, 0.9], ylim=[0, 1])
ax.patch.set_facecolor("none")
ax.text(
    0.5,
    0.1,
    "logit scale ('logit')",
    transform=ax.transAxes,
    horizontalalignment="center",
    verticalalignment="bottom",
)
ax.set_xscale("logit")
ax.set_yticks([])
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.spines["top"].set_visible(False)
ax.xaxis.set_major_locator(ticker.MultipleLocator(0.10))
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter("%.1f"))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.02))
ax.xaxis.set_minor_formatter(plt.NullFormatter())
```

缩放
Scales

- 如果这些比例都不能满足需求，读者仍然可以选择定义自己的自定义比例，在这种情况下，读者必须提供允许转换数据的正向和反向函数。反函数用于在鼠标指针悬停下显示坐标时使用。

  If none of these scales suit the needs, the user still have the option to define your own custom scale, In such case, you have to provide both the forward and inverse function that allows to transform your data. The inverse function is used when displaying coordinates under the mouse pointer.



缩放
Scales

```
# x**2 scale
# ----------------------------------------------------------
ax = plt.subplot(3, 1, 2, xlim=[0.1, 1.0], ylim=[0, 1])
ax.patch.set_facecolor("none")
ax.text(
    0.5,
    0.1,
    "$x^2$ scale",
    transform=ax.transAxes,
    horizontalalignment="center",
    verticalalignment="bottom",
)
ax.set_xscale("function", functions=(forward, inverse))
ax.set_yticks([])
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.spines["top"].set_visible(False)
ax.xaxis.set_major_locator(ticker.MultipleLocator(0.10))
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter("%.1f"))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.02))
ax.xaxis.set_minor_formatter(plt.NullFormatter())
```

```
# sqrt(x)
# ----------------------------------------------------------
ax = plt.subplot(3, 1, 3, xlim=[0.1, 1.0], ylim=[0, 1])
ax.patch.set_facecolor("none")
ax.text(
    0.5,
    0.1,
    "$\sqrt{x}$ scale",
    transform=ax.transAxes,
    horizontalalignment="center",
    verticalalignment="bottom",
)
ax.set_xscale("function", functions=(inverse, forward))
ax.set_yticks([])
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.spines["top"].set_visible(False)
ax.xaxis.set_major_locator(ticker.MultipleLocator(0.10))
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter("%.1f"))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.02))
ax.xaxis.set_minor_formatter(plt.NullFormatter())
```

投影
Projections

- 投影比缩放要复杂一些，但同时功能要强大得多。投影允许读者在将数据呈现为图形之前对数据应用任意转换。只要读者知道如何将数据变换为二维（图形空间）的渲染（同时也明晰反变换），则可以通过定义正向和逆变换，达到满足应用需求的的变换。Matplotlib仅带有几个标准投影，但提供了相关机制来创建新的依赖于域的投影，例如地图投影。读者可能会好奇原生投影为什么如此之少。答案是，对于开发人员来说，实施和维护每个特定领域的预测会太耗时且太难。相反，他们选择将投影限制为最通用的投影，即polar和3d。

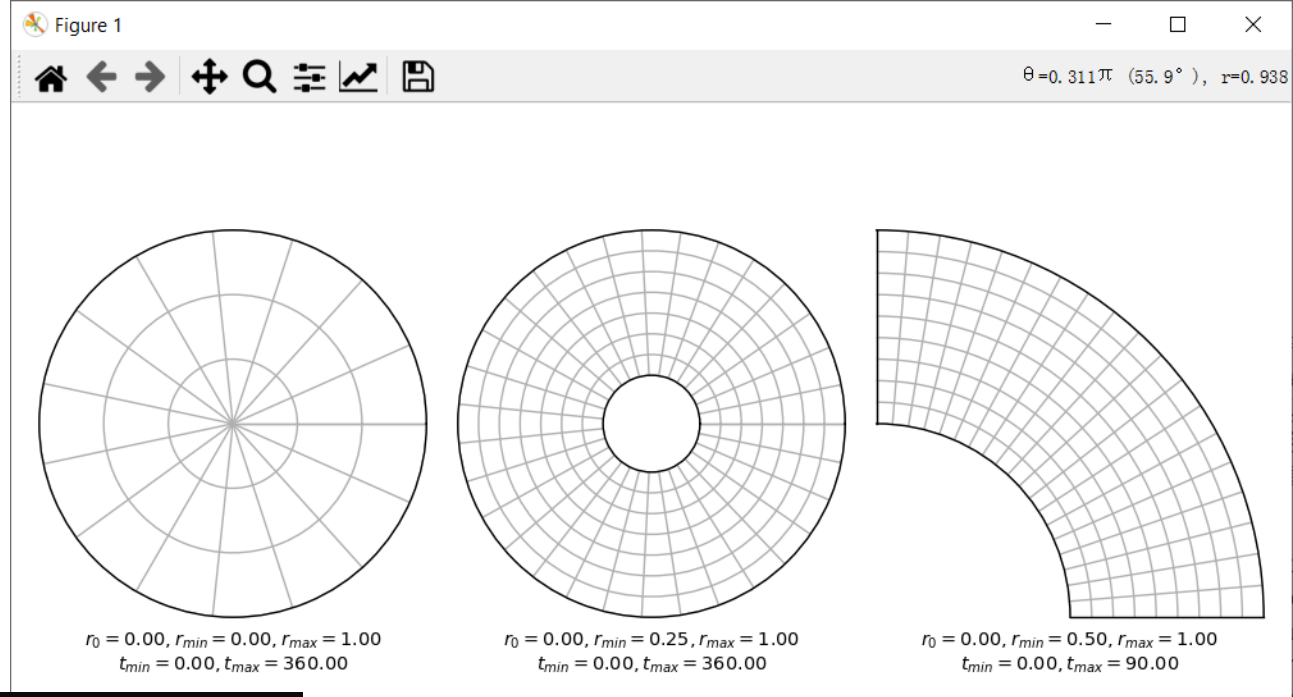
  Projections are a bit more complex than scales but, in the meantime, much more powerful. Projections allows the user to apply arbitrary transformation to data before rendering them in a figure. There is no real limit on the kind of transformation you can apply as long as you know how to transform your data into something that will be 2 dimensional (the figure space) and reciprocally. In other words, you need to define a forward and an inverse transformation. Matplotlib comes with only a few standard projections but offers all the machinery to create new domain-dependent projection such as for example cartographic projection. You might wonder why there are so few native projections. The answer is that it would be too time-consuming and too difficult for the developers to implement and maintain each and every projections that are domain specific. They chose instead to restrict projection to the most generic ones, namely polar and 3d .

投影
Projections

- 极坐标投影在讲不同坐标系统的关系时已经提及。使用极坐标的最简单的方法在创建轴时指定投影方式。当轴具备了极轴投影之后，应用的任何绘图命令都经过预处理，例如（自动）对数据应用正向变换。在极坐标投影的情况下，前向变换必须指定如何从极坐标(ρ, θ)到笛卡尔坐标$(x, y) = (\rho\cos(\theta), \rho\sin(\theta))$。当声明极轴时，有一些专用设置可以像之前所做的那样指定轴的限制，例如set_thetamin、set_thetamax、set_rmin、set_rmax，更具体地说是set_rorigin，这样可以很好地控制实际显示的内容。

  The polar projection is already mentioned in the lecture about relations between different coordinate systems. The most simple and straightforward way to use is to specify the projection when the axes are created. As long as the axis is equipped with a polar projection, any plotting command to be applied is pre-processed such as to apply (automatically) the forward transformation on the data. In the case of a polar projection, the forward transformation must specify how to go from polar coordinates (ρ, θ) to Cartesian coordinates $(x, y) = (\rho\cos(\theta), \rho\sin(\theta))$. When a polar axis is declared, you can specify limits of the axis as we've done previously using some dedicated settings such as set_thetamin, set_thetamax, set_rmin, set_rmax and more specifically set_rorigin. This allows the user to have fine control over what is actually shown.

投影
Projections




```
def polar(ax, r0, rmin, rmax, rticks, tmin, tmax, tticks):
    ax.set_yticks(np.linspace(rmin, rmax, rticks))
    ax.set_yticklabels([])
    ax.set_rorigin(r0)
    ax.set_rmin(rmin)
    ax.set_rmax(rmax)

    ax.set_xticks(np.linspace(np.pi * tmin / 180, np.pi * tmax / 180, tticks))
    ax.set_xticklabels([])
    ax.set_thetamin(tmin)
    ax.set_thetamax(tmax)

    text = r"""$r_{0}=%.2f, r_{min}=%.2f, r_{max}=%.2f$""" % (r0, rmin, rmax)
    text += "\n"
    text += r"""$t_{min}=%.2f, t_{max}=%.2f$""" % (tmin, tmax)

    plt.text(0.5, -0.15, text, size="small", ha="center", va="bottom",
             transform=ax.transAxes)
```

投影
Projections

- matplotlib提供的第二个投影是3d投影，即从3D笛卡尔空间到2D笛卡尔空间的投影。3D投影依赖于与matplotlib一起提供的Axis3D工具包。通过此工具包创建3D轴之后，便可以使用常规的绘图命令，与之前只需提供两个坐标(x,y)的区别是，现在需要提供3个坐标(x,y,z) 。可以通过以下语句创建3D投影：

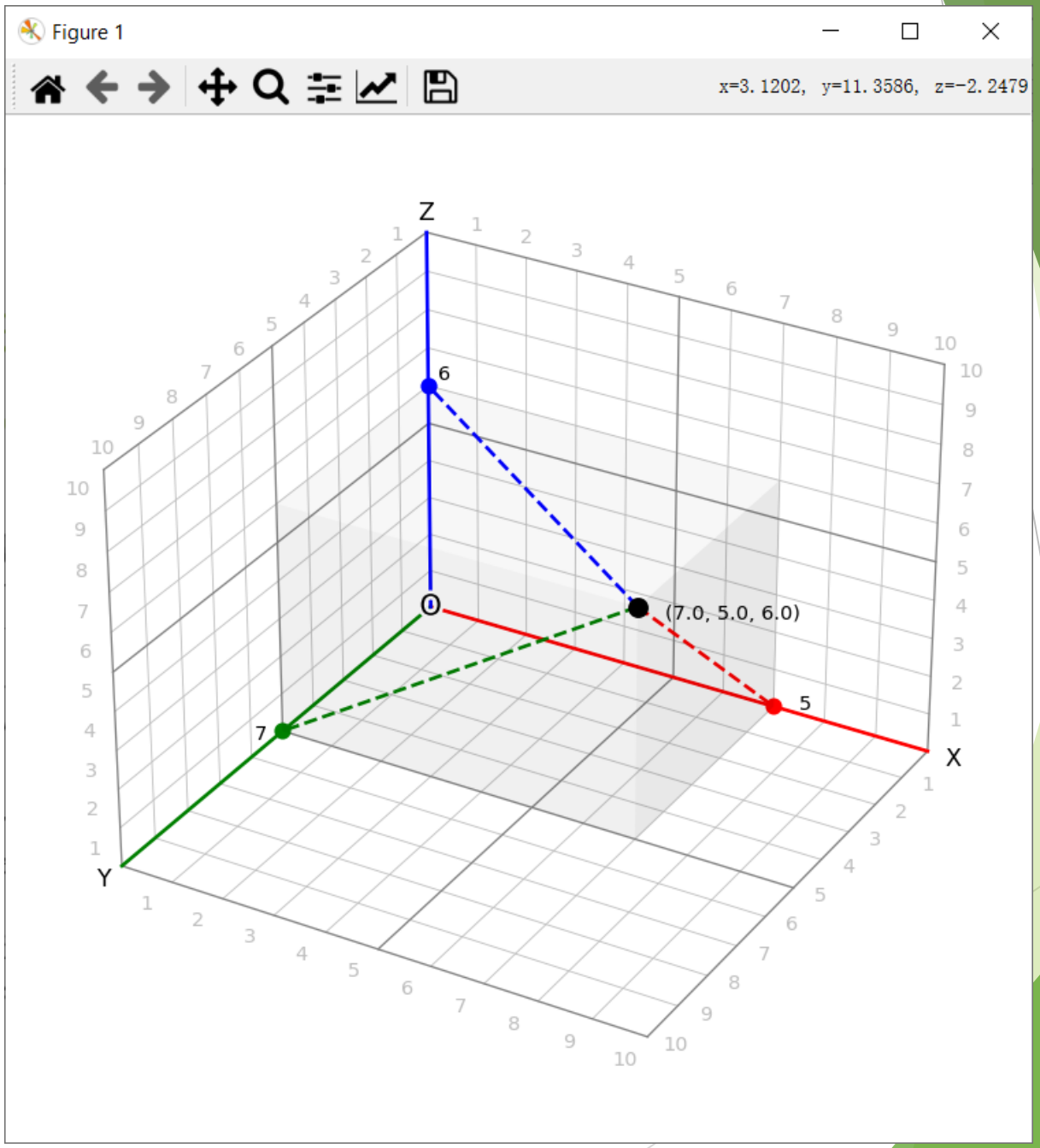
  The second projection that matplotlib offers is the 3d projection, that is the projection from a 3D Cartesian space to a 2D Cartesian space. To start using 3D projection, you'll need to use the Axis3D toolkit that is generally shipped with matplotlib. With this 3D axis, you can use regular plotting commands with a big difference though: you need now to provide 3 coordinates (x,y,z) where you previously provided only two (x,y). The 3D projection can be created via the following statements:

```
from mpl_toolkits.mplot3d import Axes3D
ax = plt.subplot(1, 1, 1, projection='3d')
```

投影
Projections

- 右图展示了从利用3D投影所绘制的图形，读者可以看到一些常见的元素与效果，均能在该图中得到体现。

  The right illustrates the figure draw by 3D projection, demonstrating the commonly-used plotting elements and effects.



投影
Projections

- 对于其他类型的投影，则需要根据打算使用的投影类型安装第三方包。其中：
  - Cartopy是一个Python 包，专为地理空间数据处理而设计，以生成地图和其他地理空间数据分析。
  - Python-ternary是一个绘图库，与matplotlib一起使用，可以在二维单纯形投影到二维平面上制作三元绘图。
  - Matplotlib-3D是一个实验项目，试图为Matplotlib提供更好、更通用的3d轴。

For other type of projections, you'll need to install third-party packages depending on the type of projection you intend to use:

- Cartopy is a Python package designed for geospatial data processing in order to produce maps and other geospatial data analyses.
- Python-ternary is a plotting library for use with matplotlib to make ternary plots in the two-dimensional simplex projected onto a two-dimensional plane.
- Matplotlib-3D is an experimental project that attempts to provide a better and more versatile 3d axis for Matplotlib.

排版
Typography

- 排版是一种排列字体的艺术，以使书面语言在显示时清晰易读和吸引人。然而，对于新手来说，排版主要被理解为屏幕上显示的字符的并列，而对于专家来说，排版意味着字体、脚本、unicode、字形、上升、下降、跟踪、提示、字距调整、整形、加粗、倾斜等等。排版实际上不仅仅是对字形的简单渲染，还涉及许多不同的概念。一般地说，字形渲染是渲染管道的重要组成部分，但如果同时读者对排版有基本的了解更有助于设计出更出色的应用。

  Typography is the art of arranging type to make written language legible, readable, and appealing when displayed. However, for the neophyte, typography is mostly apprehended as the juxtaposition of characters displayed on the screen while for the expert, typography means typeface, scripts, unicode, glyphs, ascender, descender, tracking, hinting, kerning, shaping, weight, slant, etc. Typography is actually much more than the mere rendering of glyphs and involves many different concepts. Generally, glyph rendering is an important part of the rendering pipeline, and it will be helpful if the user has a basic understanding of typography to design more elegant applications.

排版
Typography

- 虽然一个图形往往只有几个地方有文字，但考虑到绘制将在科学期刊上发表的文章中的图形时的要求，因此不能说文字不重要。一般来说，许多科技类期刊都提供一个模板，该模板决定了被接受的文章的布局，以及一个字体堆栈，即主体、参考书目和外围信息的字体选择。此外，如果读者希望一个图形有更好的外观，则需要相应地选择更适配的字体。为此，可以查看系统上安装的字体或浏览在线画廊，例如Font squirrel、dafont.com或Google font。同时，如果系统上安装了新字体，需要重建字体列表缓存使之生效，否则，Matplotlib将忽略新安装的字体。

  Although a scientific figure possesses only a few places with written text, considering a figure you make for inclusion in an article that will be published in a scientific journal, typography is still an important issue. Usually, journals of science or engineering possess a template which dictate the future layout of your accepted article as well as a font stack, that is, a choice of fonts for main body, bibliography and peripheral information. You'll still need to chose your fonts accordingly if you want your figure to have a good appearance. To do that, you can have a look at font installed on your system or browse online galleries such as Font squirrel , dafont.com or Google font. Further, if you install a new font on your system, don't forget to rebuild the font list cache or Matplotlib will just ignore you newly installed font.

排版 Typography

```
from matplotlib.font_manager import findfont, FontProperties
for family in ["serif", "sans", "monospace", "cursive"]:
    font = findfont(FontProperties(family=family))
    print(family, ":" , os.path.basename(font))
```

- Matplotlib字体集合使用四种不同的字体系列定义，即sans、serif、monospace和cursive。默认字体集合基于Bitstream Vera字体的DejaVu字体，默认草书字体是Apple Chancery。但这些只是默认选择，如果未安装默认设置，Matplotlib可以回退到其他字体，或者通过选择一组替代字体系列来设计自己的字体堆栈。下图显示了一些基于 Roboto 和 Source Pro 系列的替代字体堆栈，它们都有serif、sans和monospace 字体，并带有几个粗细。

  The Matplotlib font stack is defined using four different typeface families, namely sans, serif, monospace and cursive. The default font stack is based on the DejaVu fonts that are based on the Bitstream Vera fonts. The default cursive font is Apple Chancery . Note however that these are only the primary default choices and Matplotlib can fall back to other typefaces if the defaults are not installed. You can also design your own own font stack by choosing a set of alternative font families. The following figure shows some alternative font stacks based on the Roboto and Source Pro Family which both have serif, sans and monospace typefaces and comes with several weights..

排版
Typography

- Matplotlib有自己的 TeX 解析器和布局引擎，尽管存在一些缺陷，但仍然是非常不错的。下面展示了用Matplotlib自带的TeX解析器与LaTeX呈现的相同数学表达式的情况，读者可以注意到诸如对齐、权重、线宽等一些明显的差异。如果读者对此不能接受，仍然可以通过设置 usetex 变量来选择使用真正的 TeX 引擎：

  Matplotlib possesses its own TeX parser and layout engine which is quite capable even though it suffers from some imperfections. Here is the same mathematical expression as rendered by LaTeX for comparison. The user can notice some obvious differences such as alignment, weights, line widths, etc. If this is unacceptable in some circumstances, the user still has the option to use the real TeX engine by setting the usetex variable:

$$\frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

mathtext.fontset = "dejavuserif"

$$\frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

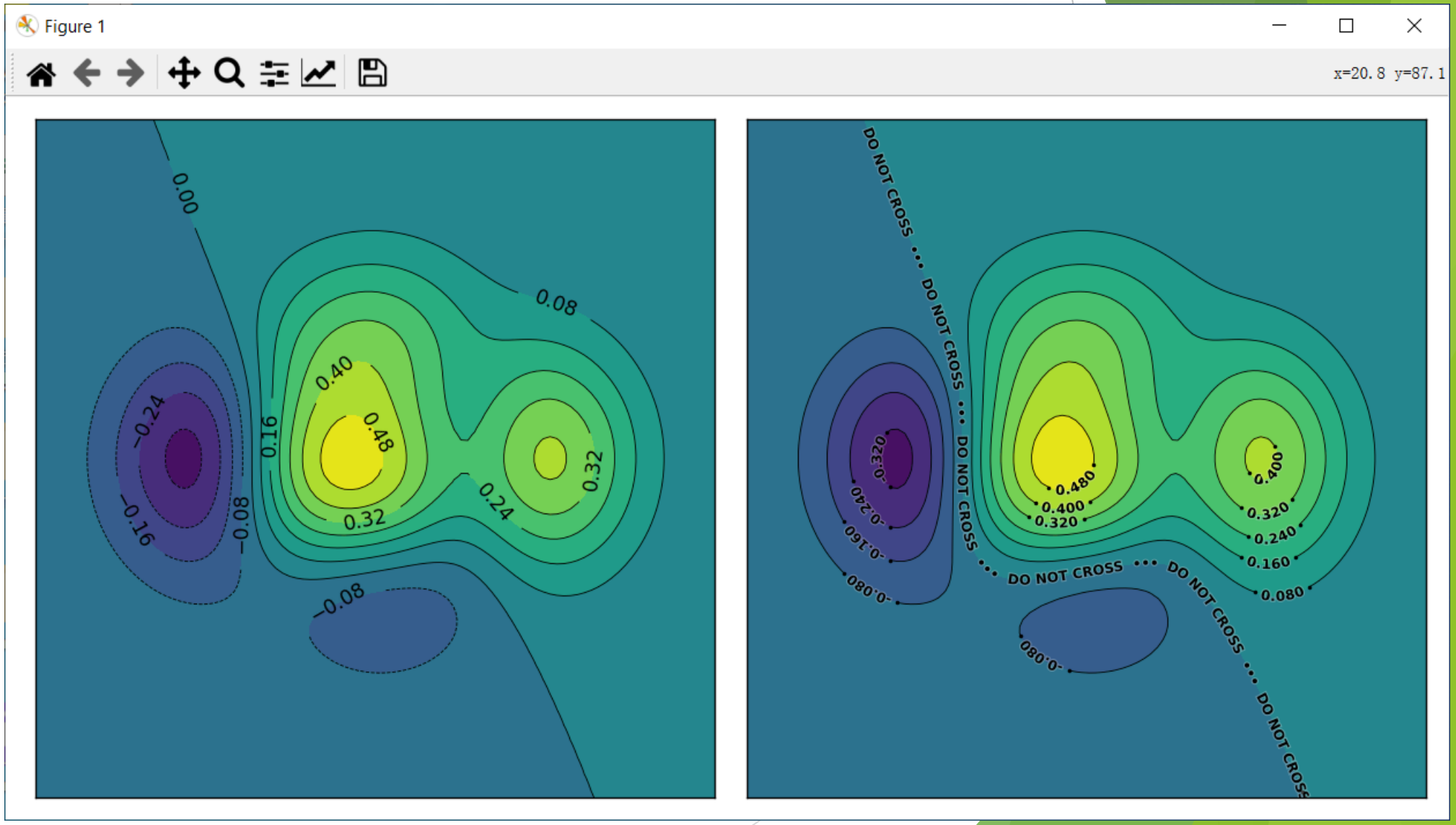
```
import matplotlib as mpl
plt.rcParams.update({"text.usetex": True})
```

排版
Typography

- 对于传统文档而言，文本通常以黑色字体和白色背景呈现，以最大限度地提高可读性。科学可视化的情况有点不同，因为在某些情况下，背景颜色也是结果的一部分，此时用户可能无法控制背景颜色。此时，重要的是要知道Matplotlib提供两种类型的文本对象。第一种是最常用的，用于标签、标题或注释的常规文本。它不支持复杂的转换操作，并且大多数情况下，即使可以自由旋转，文本也会按照单个方向（例如水平或垂直）呈现。另一种类型的文本对象为TextPath，其在可在组成字形的单个顶点级别进行转换。

  For a traditional document, text is usually rendered in black against a white background that maximizes legibility. The case of scientific visualization is a bit different because there are some situations where you cannot control the background color since it is part of your results. At this point, it is important to understand that Matplotlib offers two types of textual object. The first and most commonly used is the regular Text that is used for labels, titles or annotations. It cannot be heavily transformed and most of the time, the text is rendered following a single direction (e.g.horizontal or vertical) even though it can be freely rotated. Another textual object is the TextPath, which can be transformed at the level of individual vertices composing a glyph.

排版
Typography



颜色
Color

- 颜色是一个高度复杂的话题，为了在计算机上表示颜色，大多数时候我们使用颜色模型（如何表示颜色）和颜色空间（可以表示什么颜色）的概念。存在多种颜色模型（RGB、HSV、HLS、CMYK、CIEXYZ、CIELAB 等）和多种颜色空间（Adobe RGB、sRGB、Colormatch RGB等），因此您可以使用不同的颜色模型访问相同的颜色空间。自1996年以来，计算机上使用的标准是sRGB颜色空间，其中s代表标准。此颜色空间使用基于RGB模型的加色模型。这意味着要获得给定的颜色，需要混合不同强度的红光、绿光和蓝光。当这些量都为零时，混合后为黑色；而当这些量强度为最大值时，混合后将获得白色。

  Color is a highly complex topic. To represent a color on a computer, most of the time we use the notion of a color model (how to represent a color) and a color space (what colors can be represented). There exists several color models (RGB, HSV, HLS, CMYK, CIEXYZ, CIELAB, etc.) and several color spaces (Adobe RGB, sRGB, Colormatch RGB, etc.) such that you can access the same color space using different color models. Since 1996, the standard for computers is the sRGB color space where the s stands for standard. This color space uses an additive color model based on the RGB model. This means that to obtain a given color, you need to mix different amounts of red, green, and blue light. When these amounts are all zero, you obtain black and when these amounts are all at full intensity, you obtain white.

颜色
Color

- 注意，当在matplotlib中指定颜色时（例如“#123456”），需要意识到这种颜色是在sRGB颜色模型和空间中隐式编码的。这个事实的结果是，由于sRGB模型不是线性的，如果想尝试使用一种简单的方法在两种颜色之间产生渐变，可能会得到错误的感知结果。

  Consequently, when you specify a color in matplotlib (e.g. "#123456"), you need to realize that this color is implicitly encoded in the sRGB color model and space. This draws immediate consequences that, for example, if you try to produce a gradient between two colors using a naive approach, you’ll get wrong perceptual results because the sRGB model is not linear.

- 另一个流行的模型是 HSV 模型，它代表色相、饱和度和值。它提供了一种替代颜色模型来访问与 sRGB 系统相同的颜色空间。Matplotlib提供了与HSV模型相互转换的方法。

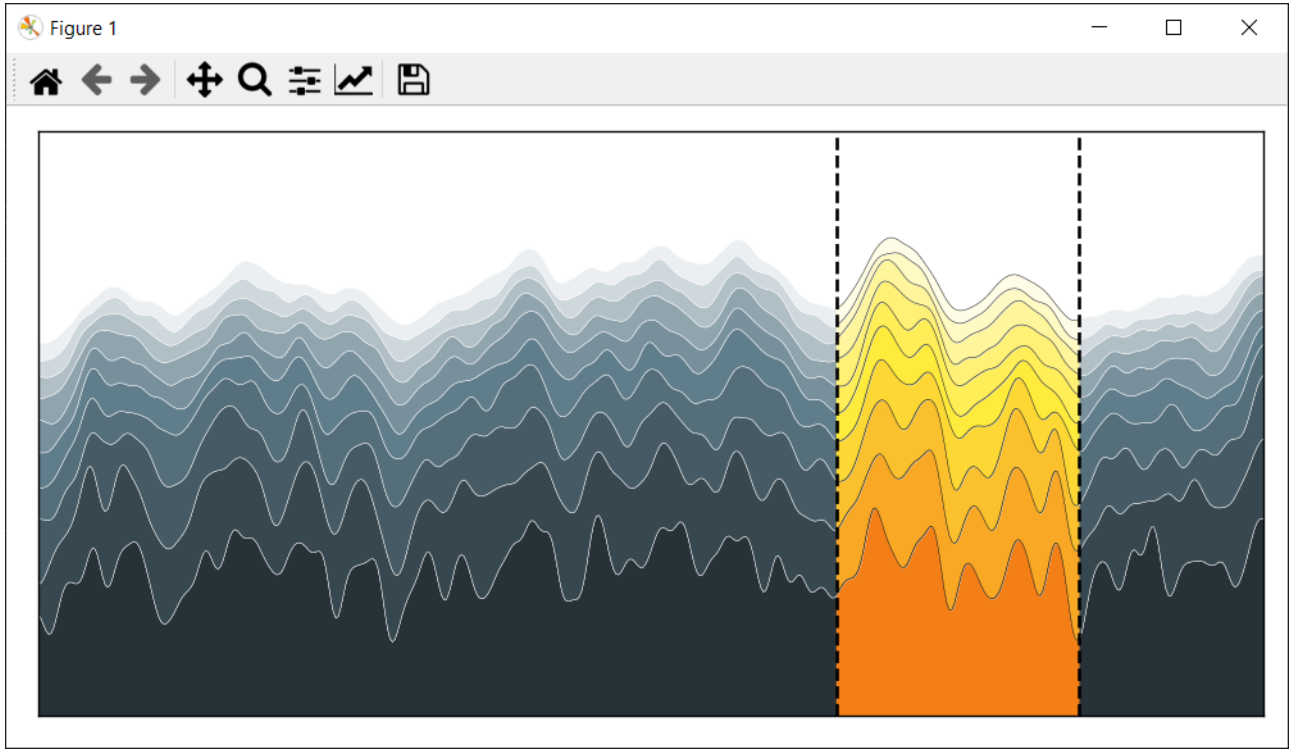
  Another popular model is the HSV model that stands for Hue, Saturation and Value. It provides an alternate color model to access the same color space as the sRGB system. Matplotlib provides methods to convert to and from the HSV model.

颜色
Color

- 通常在绘图时，可以让Matplotlib自动选择颜色。当一次绘制多个图时，读者可能已经注意到这些图使用了几种不同的颜色，这些颜色是从所谓的颜色循环中挑选出来的。同时，这些颜色被设计成足够不同，以减轻视觉差异的感知，同时对眼睛不太敏感（例如，与饱和的纯蓝色、绿色或红色相比）。如果读者需要更多颜色，您首先需要问自己是否真的需要更多颜色。只有当仔细斟酌之后，才建议考虑使用精心设计的调色板。由专家设计的调色板包括开源调色板，材料调色板等。

  The user can delegate to Matplotlib for choosing colors when plotting. When draw several plots at once, the user may have noticed that the plots use several different colors. These colors are picked from what is called a color cycle. These colors have been designed to be sufficiently different such as to ease the visual perception of difference while being not too aggressive on the eye (compared to saturated pure blue, green or red colors for example). If you need more colors, you need first to ask yourself whether you really need more colors. Then, and only then, you might consider using palettes that have been designed with care. Palettes designed by experts include open color palette, the material color palette, etc.

颜色
Color



- 在下图中，两种颜色堆栈（材质调色板中的蓝灰色和黄色）被用来突出显示感兴趣的区域。

  In the following figure, two color stacks (blue grey and yellow from the material palettes) are used to highlight an area of interest.

颜色
Color

- 颜色映射是为每个值定义相应颜色的颜色图，即值到颜色的映射。不同类型的颜色图（顺序的、发散的、循环的等）对应于不同的用例。使用与用户的数据对应的正确类型或颜色图非常重要。对于每种类型，都存在多个颜色图。一些与顺序颜色图相关的选择一般来说比较简单，但发散的颜色图需要特别小心，因为它们实际上是由两个具有特殊中心值的渐变组成的。默认情况下，这个中心值在归一化线性映射中映射到 0.5，只要数据的绝对最小值和最大值相同，它就可以很好地工作。如果我们在没有任何预防措施的情况下使用发散的颜色图，则无法保证我们会获得我们想要的结果。

  Colormapping corresponds to the mapping of values to colors, using a colormap that defines, for each value, the corresponding color. There are different types of colormaps (sequential, diverging, cyclic, etc.) that correspond to different use cases. It is is important to use the right type or colormap that corresponds to your data. For each type, there exist several colormaps. The choices associated with sequential colormaps might be simple. Diverging colormaps needs special care because they are really composed of two gradients with a special central value. By default, this central value is mapped to 0.5 in the normalized linear mapping and this works pretty well as long as the absolute minimum and maximum value of your data are the same. If we use a diverging colormap without any precaution, there’s no guarantee that we’ll obtain the result we want.

颜色
Color

- 考虑如下图所示的情况，这里有一个带有负值的小域和一个带有正值的较大域。理想情况下，我们希望负值映射为蓝色，正值映射为淡黄色。为达到此目标，我们需要告诉matplotlib什么是中心值，并使用双斜率范数而不是线性范数。

  Consider the situation illustrated on the following figure. Here we have a small domain with negative values and a larger domain with positive values. Ideally, it is expected the negative values to be mapped with blueish colors and positive values with yellowish colors. To do it, we thus need to tell matplotlib what is the central value and to use a Two Slope norm instead of a Linear norm .