DENSITY AND CONTOUR PLOTS

To display three-dimensional data in two dimensions using contours or color-coded regions. There are three Matplotlib functions that can be helpful for this task:

- plt.contour for contour plots,
- plt.contourf for filled contour plots, and
- plt.imshow for showing images.

Visualizing a Three-Dimensional Function

A contour plot can be created with the plt.contour function. It takes three arguments:

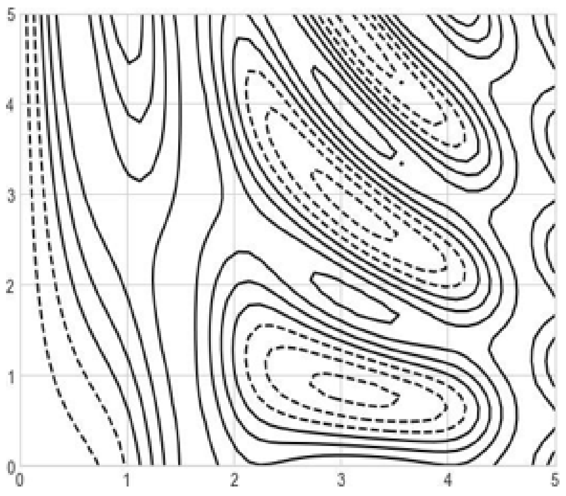
- a grid of x values,
- a grid of y values, and
- a grid of z values.

The x and y values represent positions on the plot, and the z values will be represented by the contour levels.

The way to prepare such data is to use the np.meshgrid function, which builds two-dimensional grids from one-dimensional arrays:

Example

```
def f(x, y):
        return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.contour(X, Y, Z, colors='black');
```

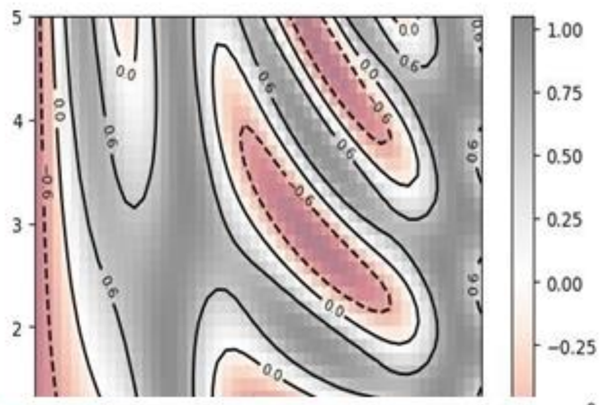


There are a few potential gotchas with imshow().

- plt.imshow() doesn’t accept an x and y grid, so you must manually specify the extent [xmin, xmax, ymin, ymax] of the image on the plot.
- plt.imshow() by default follows the standard image array definition where the origin is in the upper left, not in the lower left as in most contour plots. This must be changed when showing gridded data.
- plt.imshow() will automatically adjust the axis aspect ratio to match the input data; you can change this by setting, for example, plt.axis(aspect='image') to make x and y units match.

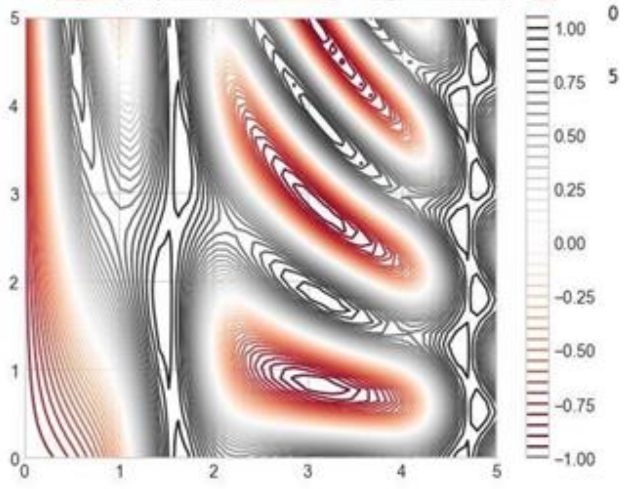
Finally, it can sometimes be useful to combine contour plots and image plots. we'll use a partially transparent background image (with transparency set via the alpha parameter) and over-plot contours with labels on the contours themselves (using the plt.clabel() function):

```
contours = plt.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)
plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
cmap='RdGy', alpha=0.5)
plt.colorbar();
```



Example Program

```
import numpy as np
import matplotlib.pyplot as plt
def f(x, y):
return np.sin(x) ** 10 + np.cos(10 + y * x) *
np.cos(x)
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.imshow(Z, extent=[0, 10, 0, 10],
origin='lower', cmap='RdGy')
plt.colorbar()
```



HISTOGRAMS

Histogram is the simple plot to represent the large data set. A histogram is a graph showing frequency distributions. It is a graph showing the number of observations within each given interval.

Parameters

- plt.hist( ) is used to plot histogram. The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.
- bins - A histogram displays numerical data by grouping data into "bins" of equal width. Each bin is plotted as a bar whose height corresponds to how many data points are in that bin. Bins are also sometimes called "intervals", "classes", or "buckets".
- normed - Histogram normalization is a technique to distribute the frequencies of the histogram over a wider range than the current range.
- x - (n,) array or sequence of (n,) arrays Input values, this takes either a single array or a sequence of arrays which are not required to be of the same length.
- histtype - {'bar', 'barstacked', 'step', 'stepfilled'}, optional

The type of histogram to draw.

- 'bar' is a traditional bar-type histogram. If multiple data are given the bars are arranged side by side.
- 'barstacked' is a bar-type histogram where multiple data are stacked on top of each other.
- 'step' generates a lineplot that is by default unfilled.

- 'stepfilled' generates a lineplot that is by default filled.

Default is 'bar'

- align - {'left', 'mid', 'right'}, optional Controls how the histogram is plotted.
    1. 'left': bars are centered on the left bin edges.
    2. 'mid': bars are centered between the bin edges.
    3. 'right': bars are centered on the right bin edges.

Default is 'mid'

- orientation - {'horizontal', 'vertical'}, optional
  If 'horizontal', barh will be used for bar-type histograms and the bottom kwarg will be the left edges.
- color - color or array_like of colors or None, optional
  Color spec or sequence of color specs, one per dataset. Default (None) uses the standard line color sequence.

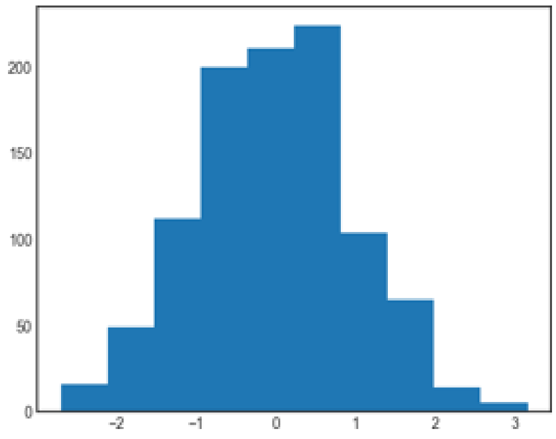
  Default is None
- label - str or None, optional. Default is None

Other parameter

- **kwargs - Patch properties, it allows us to pass a variable number of keyword arguments to a python function. ** denotes this type of function.

Example

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data);
```

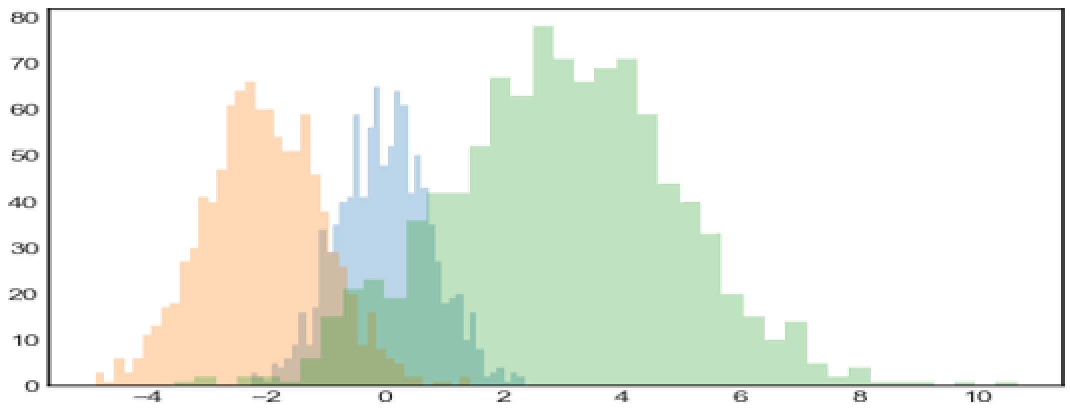


The hist() function has many options to tune both the calculation and the display; here's an example of a more customized histogram.

plt.hist(data, bins=30, alpha=0.5,histtype='stepfilled', color='steelblue',edgecolor='none');

The plt.hist docstring has more information on other customization options available. I find this combination of histtype='stepfilled' along with some transparency alpha to be very useful when comparing histograms of several distributions.

```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)
kwargs = dict(histtype='stepfilled', alpha=0.3, bins=40)
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
```

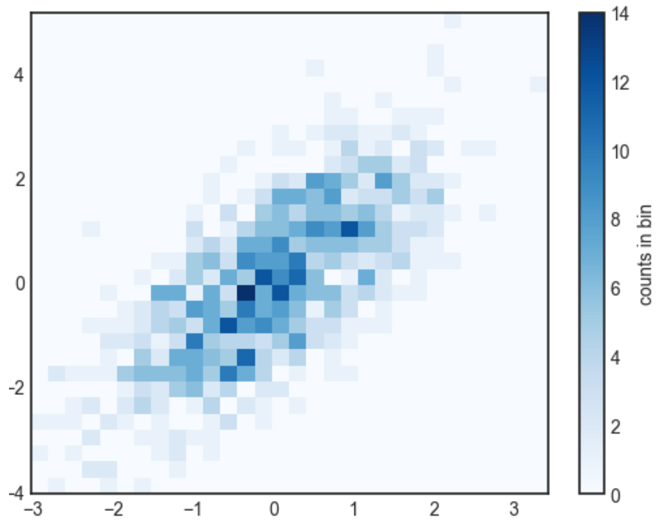


Two-Dimensional Histograms and Binnings

- We can create histograms in two dimensions by dividing points among two dimensional bins.
- We would define x and y values. Here for example We'll start by defining some data—an x and y array drawn from a multivariate Gaussian distribution:
- Simple way to plot a two-dimensional histogram is to use Matplotlib's plt.hist2d() function

Example

```
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 1000).T
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```


counts in bin