Topic for the class-Visualizing errors, density and contour Plots

Unit _2 : Title-Digital data – an Imprint

Date & Time : 12.8.24 11.00 AM – 11.50 AM

Dr. Bhramaramba Ravi
Professor
Department of Computer Science and Engineering
GITAM School of Technology (GST)
Visakhapatnam – 530045
Email: **bravi@gitam.edu**

Unit2-syllabus

- **UNIT 2 Digital Data-An Imprint 9 hours, P - 2 hours** Type of data analytics (Descriptive, diagnostic, perspective, predictive, Prescriptive.) Exploratory Data Analysis (EDA), EDA-Quantitative Technique, EDA - Graphical Technique. Data Types for Plotting, Data Types and Plotting, Simple Line Plots, Simple Scatter Plots, Visualizing Errors, Density and Contour Plots, Histograms, Binnings, and Density, Customizing Plot Legends, Customizing Color bars, Multiple Subplots, Text and Annotation, Customizing Ticks.
- https://www.coursera.org/learn/data-visualization-r

Visualizing errors

- For any scientific measurement, accurate accounting for errors is nearly as important, if not more important, than accurate reporting of the number itself.
- Now are the values consistent? That is a question that can be quantitatively answered.
- In visualization of data and results, showing these errors effectively can make a plot convey much more complete information.
- **Basic Errorbars**
- A basic errorbar can be created with a single Matplotlib function call (Figure 4-27):
- In[1]: %matplotlib inline
- **import matplotlib.pyplot as plt**
- plt.style.use('seaborn-whitegrid')
- **import numpy as np**
- In[2]: x = np.linspace(0, 10, 50)
- dy = 0.8
- y = np.sin(x) + dy * np.random.randn(50)
- plt.errorbar(x, y, yerr=dy, fmt='.k');

Fig. Errorbar example

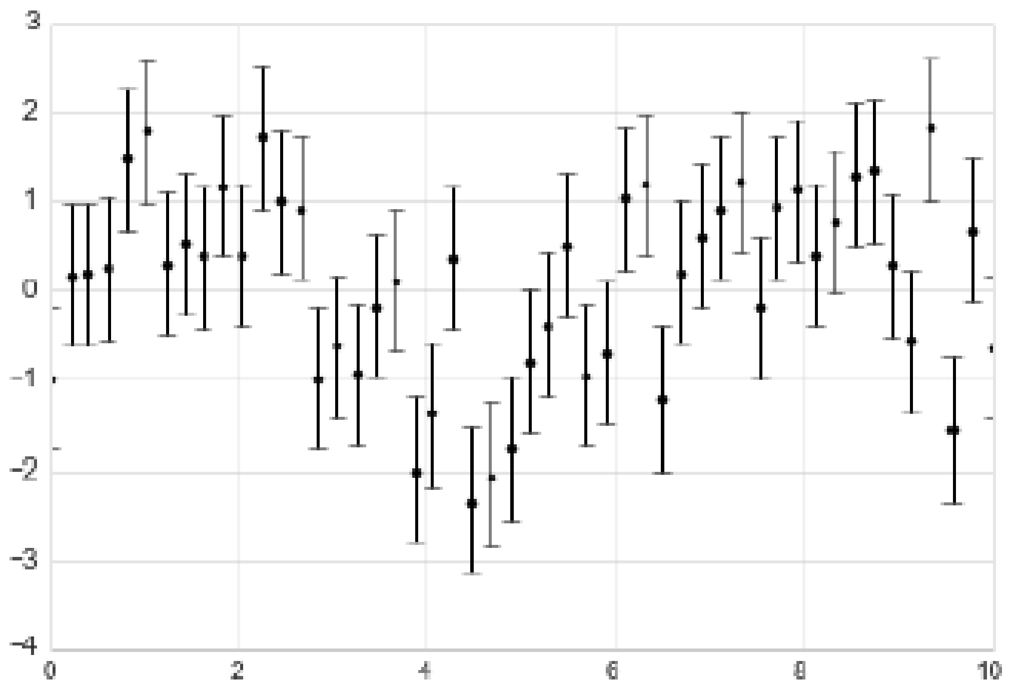

Figure 4-27. An errorbar example

Errorbar

- Here the fmt is a format code controlling the appearance of lines and points, and hasthe same syntax as the shorthand used in plt.plot
- In addition to these basic options, the errorbar function has many options to finetune the outputs. Using these additional options you can easily customize the aesthetics of your errorbar plot.
- In[3]: plt.errorbar(x, y, yerr=dy, fmt='o', color='black’, ecolor='lightgray', elinewidth=3, capsize=0);

Fig. Customizing errorbars

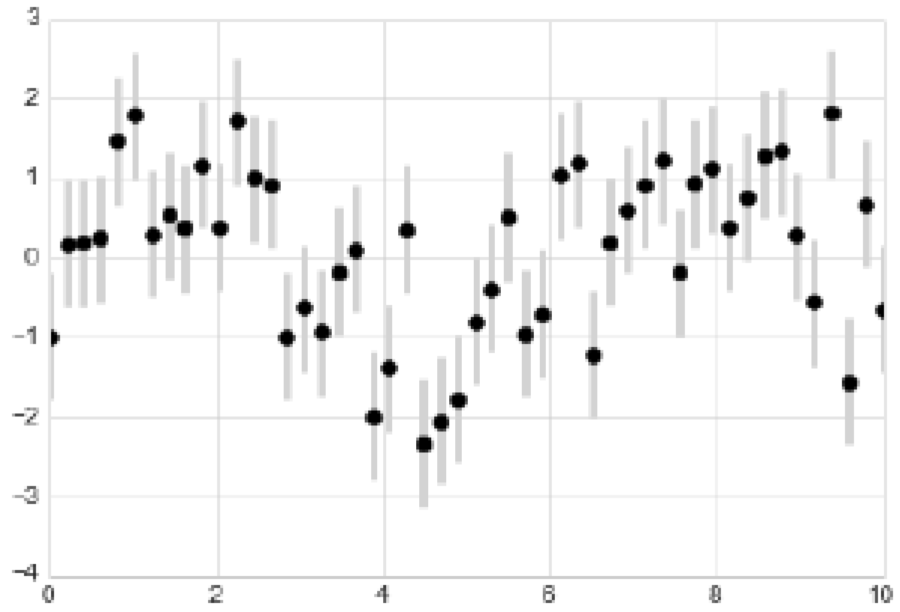

Figure 4-28. Customizing errorbars

Customizing errorbars

- In addition to these options, you can also specify horizontal errorbars (xerr), onesided errorbars, and many other variants

Continuous errors

- Matplotlib does not have a built-in convenience routine for this type of application,
- it’s relatively easy to combine primitives like plt.plot and plt.fill_between for a useful result.
- Here we’ll perform a simple *Gaussian process regression* (GPR), using the Scikit-Learn API.
- This is a method of fitting a very flexible nonparametric function to data with a continuous measure of the uncertainty.
- Here we will focus on how you might visualise a continuous error measurement.

Continuous errors contd.

- In[4]: **from** sklearn.gaussian_process **import** GaussianProcess
- *# define the model and draw some data*
- model = **lambda** x: x * np.sin(x)
- xdata = np.array([1, 3, 5, 6, 8])
- ydata = model(xdata)
- *# Compute the Gaussian process fit*
- gp = GaussianProcess(corr='cubic', theta0=1e-2, thetaL=1e-4, thetaU=1E-1,
- random_start=100)
- gp.fit(xdata[:, np.newaxis], ydata)
- xfit = np.linspace(0, 10, 1000)
- yfit, MSE = gp.predict(xfit[:, np.newaxis], eval_MSE=True)
- dyfit = 2 * np.sqrt(MSE) *# 2*sigma ~ 95% confidence region*

Continuous errors contd.

- We now have xfit, yfit, and dyfit, which sample the continuous fit to our data. We
- could pass these to the plt.errorbar function as above, but we don’t really want to
- plot 1,000 points with 1,000 errorbars. Instead, we can use the plt.fill_between
- function with a light color to visualize this continuous error (Figure 4-29):
- In[5]: *# Visualize the result*
- plt.plot(xdata, ydata, 'or')
- plt.plot(xfit, yfit, '-', color='gray')
- plt.fill_between(xfit, yfit - dyfit, yfit + dyfit,
- color='gray', alpha=0.2)
- plt.xlim(0, 10);

Continuous errors contd.

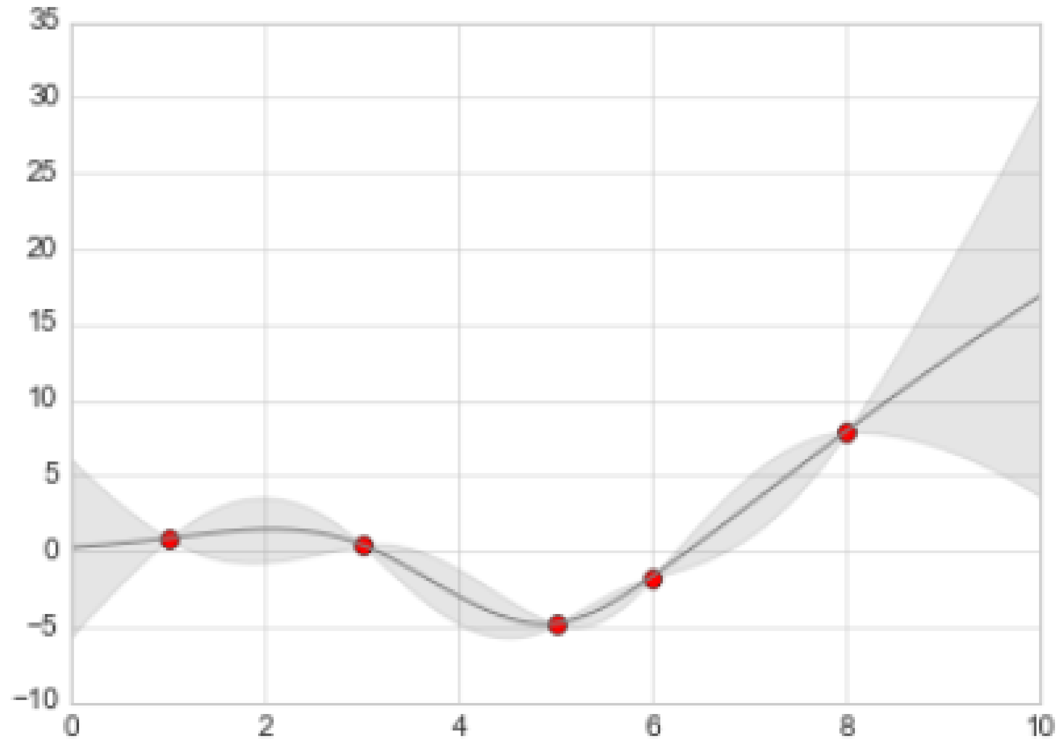

Figure 4-29. Representing continuous uncertainty with filled regions

Continuous errors contd.

- Note what we've done here with the fill_between function: we pass an x value, then the lower y-bound, then the upper y-bound, and the result is that the area between these regions is filled.
- The resulting figure gives a very intuitive view into what the Gaussian process regression algorithm is doing: in regions near a measured data point, the model is strongly constrained and this is reflected in the small model errors. In regions far from a measured data point, the model is not strongly constrained, and the model errors increase.
- Seaborn package, which has a more streamlined API for visualizing this type of continuous errorbar.

Density and contour plots

- Sometimes it is useful 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
- In[1]: %matplotlib inline
- **import matplotlib.pyplot as plt**
- plt.style.use('seaborn-white')
- **import numpy as np**

Visualizing a Three-Dimensional Function

- demonstrating a contour plot using a function *z* = *f x*, *y* , using the following particular choice for *f*
- In[2]: **def** f(x, y):
- **return** np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)

Visualizing a three-dimensional function contd.

- 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.
- Perhaps the most straightforward way to prepare such data is to use the np.meshgrid function, which builds two-dimensional grids from one-dimensional
- arrays:
- In[3]: x = np.linspace(0, 5, 50)
- y = np.linspace(0, 5, 40)
- X, Y = np.meshgrid(x, y)
- Z = f(X, Y)
- Now let's look at this with a standard line-only contour plot (Figure 4-30):
- In[4]: plt.contour(X, Y, Z, colors='black');

Visualizing a three-dimensional function contd

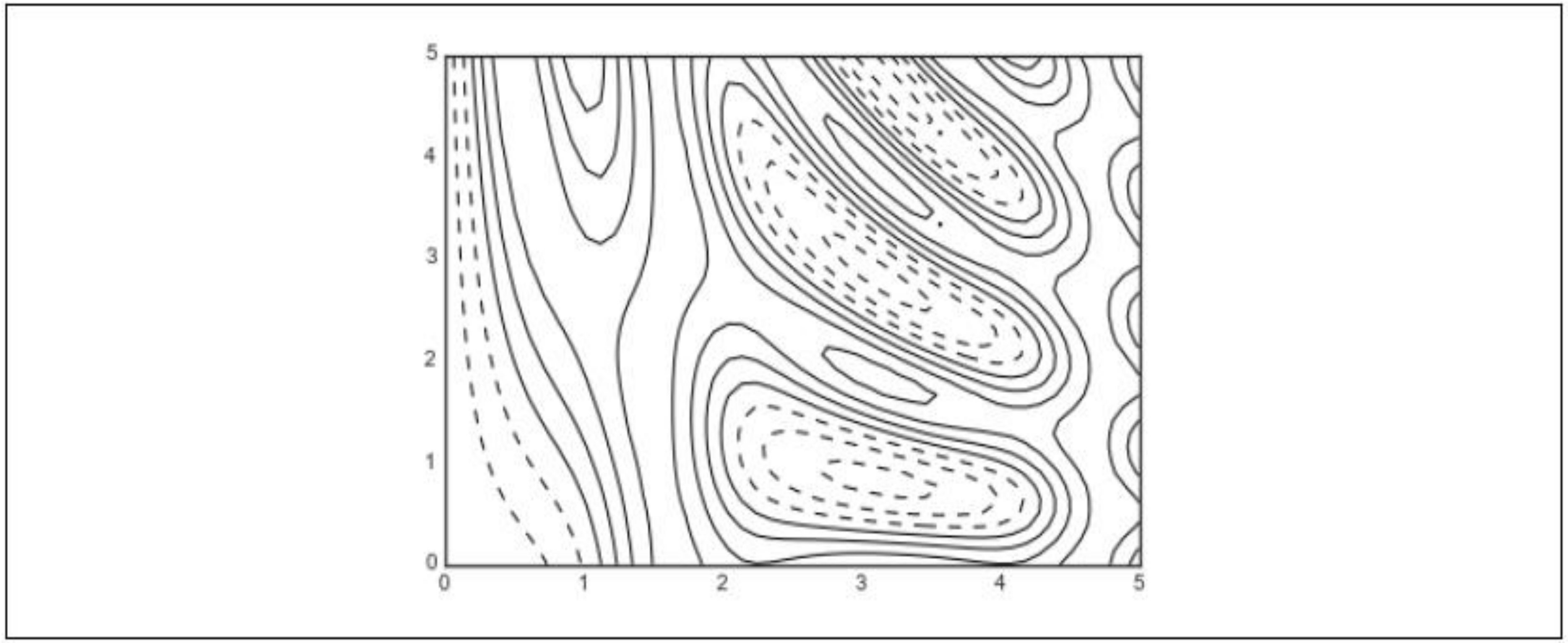

Figure 4-30. Visualizing three-dimensional data with contours

Visualizing a three-dimensional function contd.

- Notice that by default when a single color is used, negative values are represented by dashed lines, and positive values by solid lines.
- Alternatively, you can color-code the lines by specifying a colormap with the cmap argument.
- Here, we'll also specify that
- we want more lines to be drawn—20 equally spaced intervals within the data range
- (Figure 4-31):
- In[5]: plt.contour(X, Y, Z, 20, cmap='RdGy');

Visualizing a three-dimensional function contd.

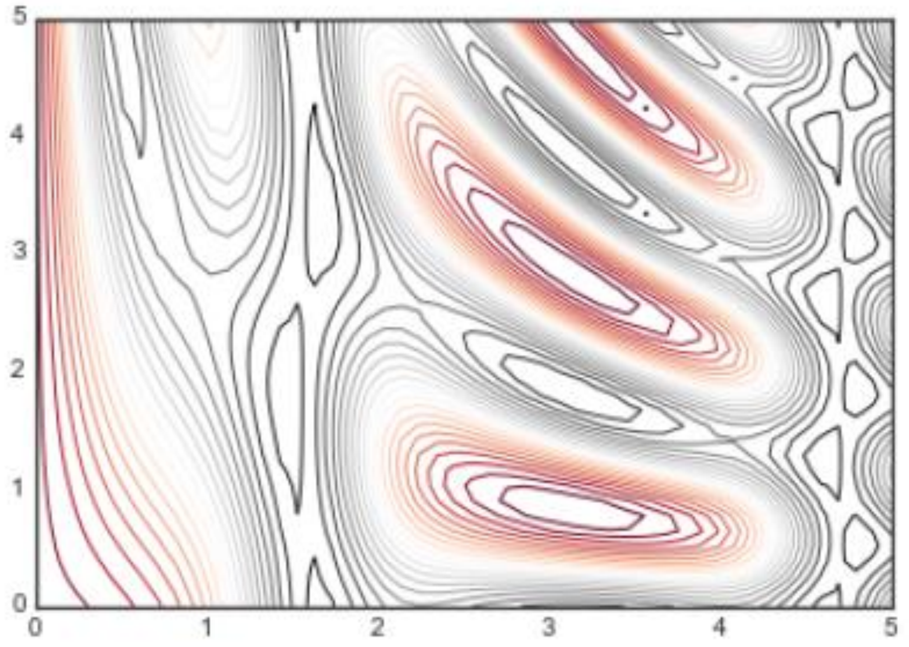

Figure 4-31. Visualizing three-dimensional data with colored contours

Visualizing a three dimensional function contd.

- Here we chose the RdGy (short for *Red-Gray*) colormap, which is a good choice for centered data.
- Matplotlib has a wide range of colormaps available, which you can easily browse in IPython by doing a tab completion on the plt.cm module:
- plt.cm.<TAB>
- Our plot is looking nicer, but the spaces between the lines may be a bit distracting.
- We can change this by switching to a filled contour plot using the plt.contourf() function (notice the f at the end), which uses largely the same syntax as plt.contour().

Visualizing a three-dimensional function contd.

- Additionally, we'll add a plt.colorbar() command, which automatically creates an
- additional axis with labeled color information for the plot (Figure 4-32):
- In[6]: plt.contourf(X, Y, Z, 20, cmap='RdGy')
- plt.colorbar();

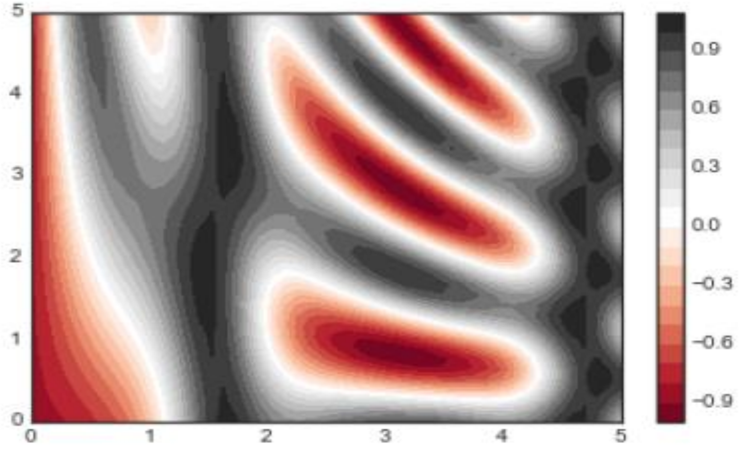

Figure 4-32. Visualizing three-dimensional data with filled contours

Visualizing a three dimensional function contd.

- The colorbar makes it clear that the black regions are “peaks,” while the red regions are “valleys.”
- One potential issue with this plot is that it is a bit “splotchy.” That is, the color steps are discrete rather than continuous, which is not always what is desired.
- You could remedy this by setting the number of contours to a very high number, but this results in a rather inefficient plot:
- Matplotlib must render a new polygon for each step in the level. A better way to handle this is to use the plt.imshow() function, which interprets a two-dimensional grid of data as an image.
- Figure 4-33 shows the result of the following code:
- In[7]: plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
- cmap='RdGy')
- plt.colorbar()
- plt.axis(aspect='image');

Visualizing a three dimensional function contd.

- There are a few potential gotchas with imshow(), however:
- • 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. T
- This must be changed when showing gridded data.

Visualizing a three dimensional function contd.

- 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.

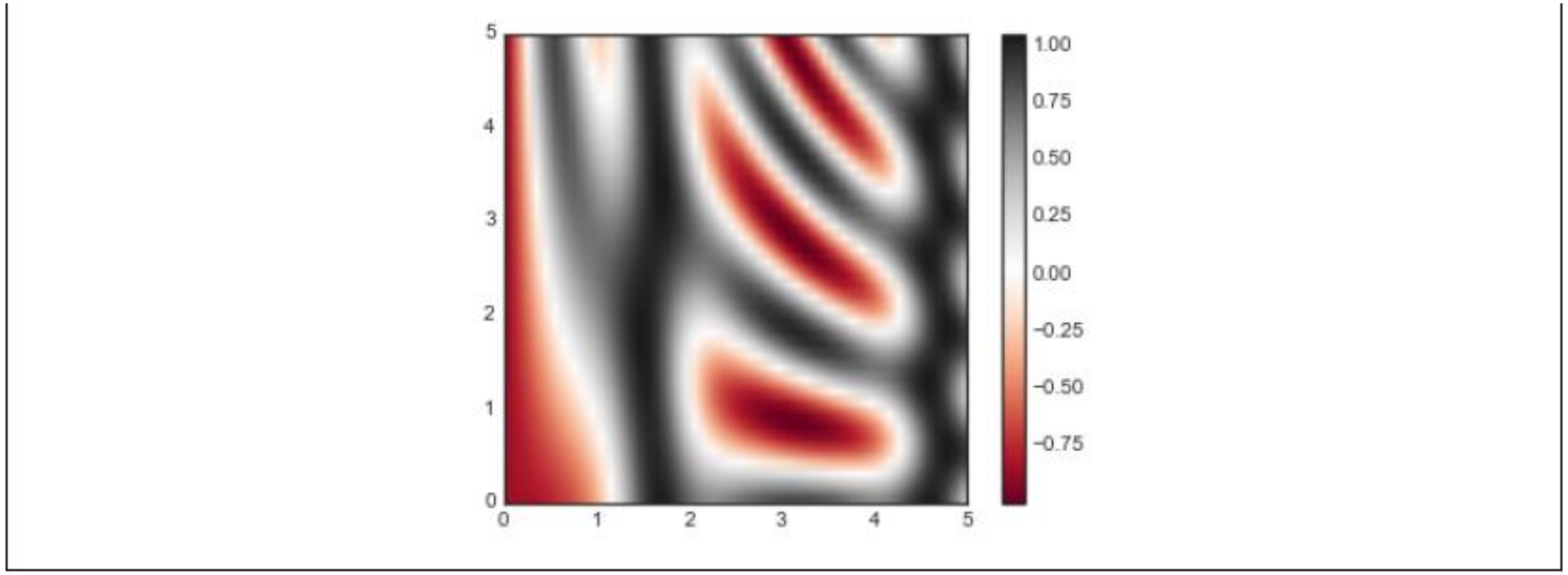

Figure 4-33. Representing three-dimensional data as an image

Visualizing a three dimensional function contd.

- Finally, it can sometimes be useful to combine contour plots and image plots. For example, to create the effect shown in Figure 4-34, 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):
- In[8]: 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();

Visualizing a three dimensional function contd.

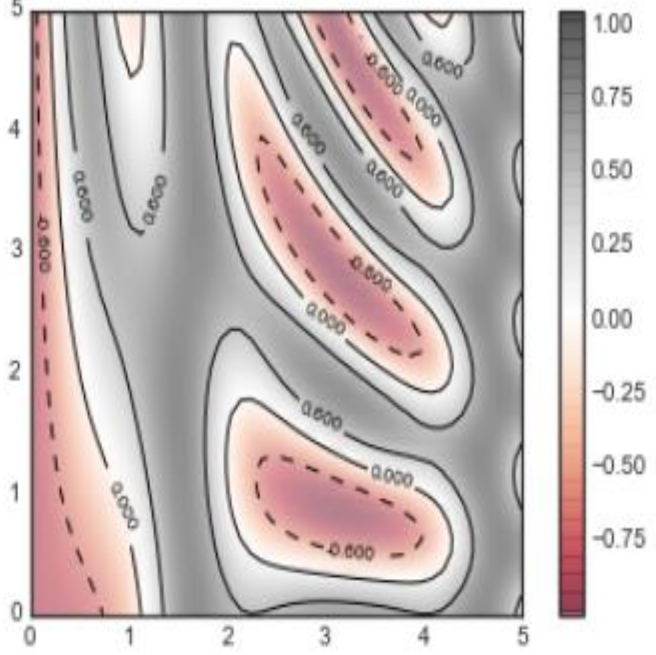

Figure 4-34. Labeled contours on top of an image

Visualizing a three dimensional function contd.

- The combination of these three functions—plt.contour, plt.contourf, and plt.imshow—gives nearly limitless possibilities for displaying this sort of threedimensional data within a two-dimensional plot.

THANK YOU