



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Working with 2D arrays



Reminder: NumPy arrays

- Homogeneous in type
- Calculations all at once
- Indexing with brackets:
 - $A[\textit{index}]$ for 1D array
 - $A[\textit{index0}, \textit{index1}]$ for 2D array



Reminder: slicing arrays

- Slicing: 1D arrays: $A[\textit{slice}]$, 2D arrays: $A[\textit{slice0}, \textit{slice1}]$
- Slicing: $\textit{slice} = \textit{start}:\textit{stop}:\textit{stride}$
 - Indexes from \textit{start} to $\textit{stop}-1$ in steps of \textit{stride}
 - Missing \textit{start} : implicitly at *beginning* of array
 - Missing \textit{stop} : implicitly at *end* of array
 - Missing \textit{stride} : implicitly $\textit{stride } 1$
- Negative indexes/slices: count from *end of array*

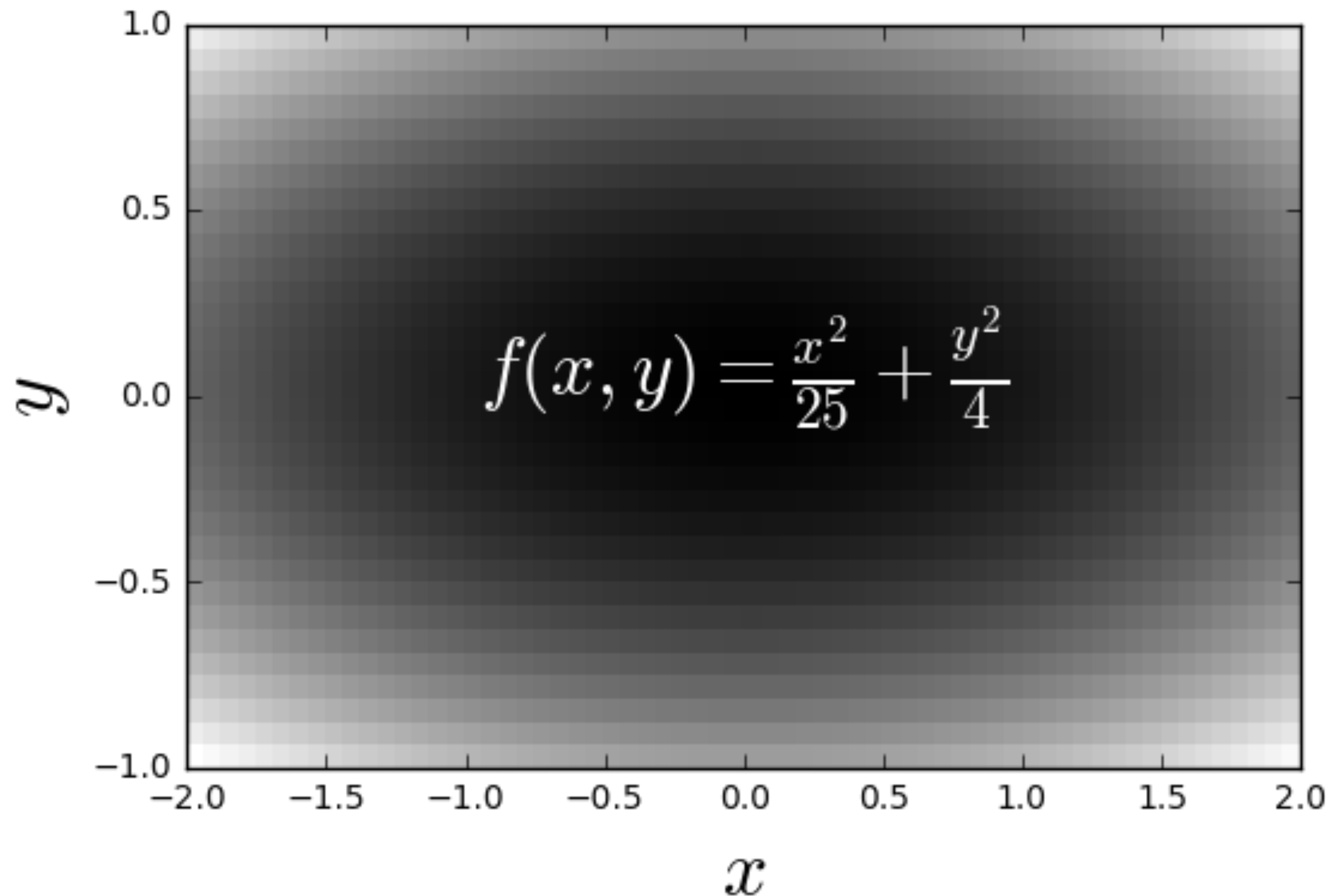
2D arrays & images

0.434	0.339	0.337	0.367	...
0.434	0.421	0.404	0.395	...
0.350	0.388	0.340	0.340	...
0.328	0.384	0.308	0.308	...
...





2D arrays & functions





Using meshgrid()



meshgrids.py

```
import numpy as np
```

```
u = np.linspace(-2, 2, 3)
```

```
v = np.linspace(-1, 1, 5)
```

```
X,Y = np.meshgrid(u, v)
```

```
Z = X**2/25 + Y**2/4
```



Using meshgrid()

X:

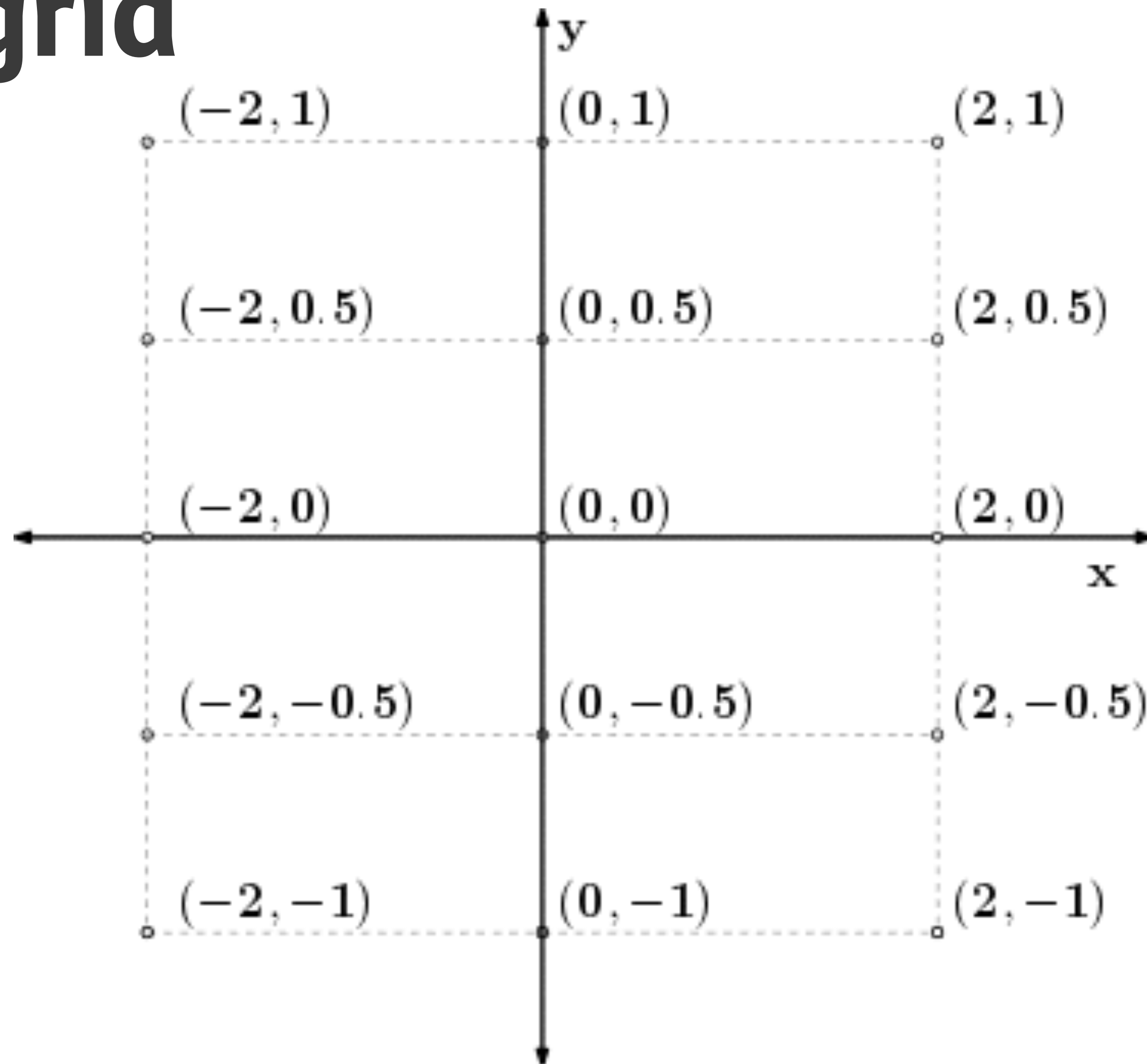
```
[[-2.  0.  2.]  
 [-2.  0.  2.]  
 [-2.  0.  2.]  
 [-2.  0.  2.]  
 [-2.  0.  2.]]
```

Y:

```
[[-1.  -1.  -1. ]  
 [-0.5 -0.5 -0.5]  
 [ 0.   0.   0. ]  
 [ 0.5  0.5  0.5]  
 [ 1.   1.   1. ]]
```




Meshgrid





Sampling on a grid



meshgrids.py

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

```
u = np.linspace(-2, 2, 3)
v = np.linspace(-1, 1, 5)
X,Y = np.meshgrid(u, v)
Z = X**2/25 + Y**2/4
```

```
print('Z:\n', Z)
plt.set_cmap('gray')
plt.pcolor(Z)
plt.show()
```

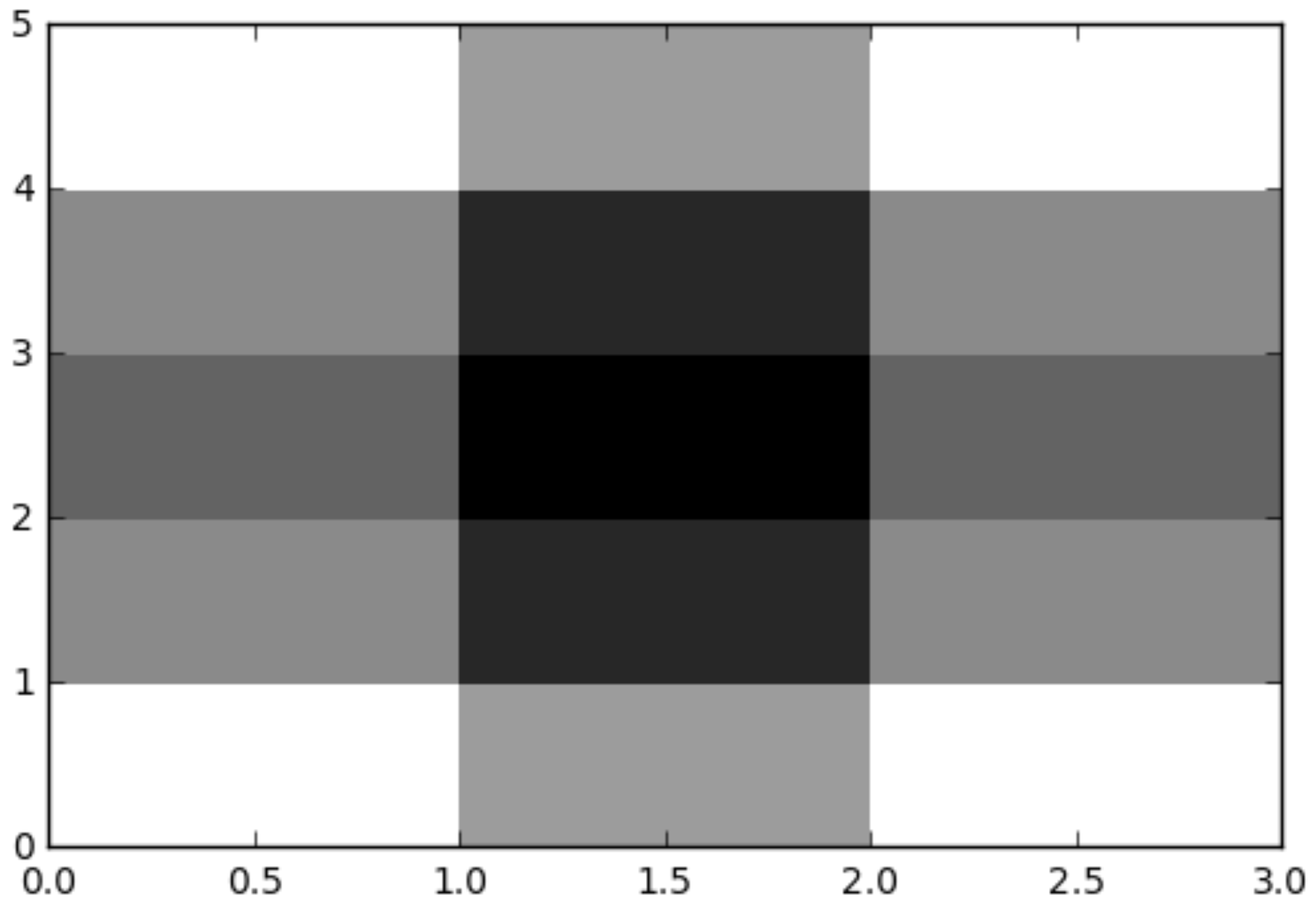


Sampling on a grid

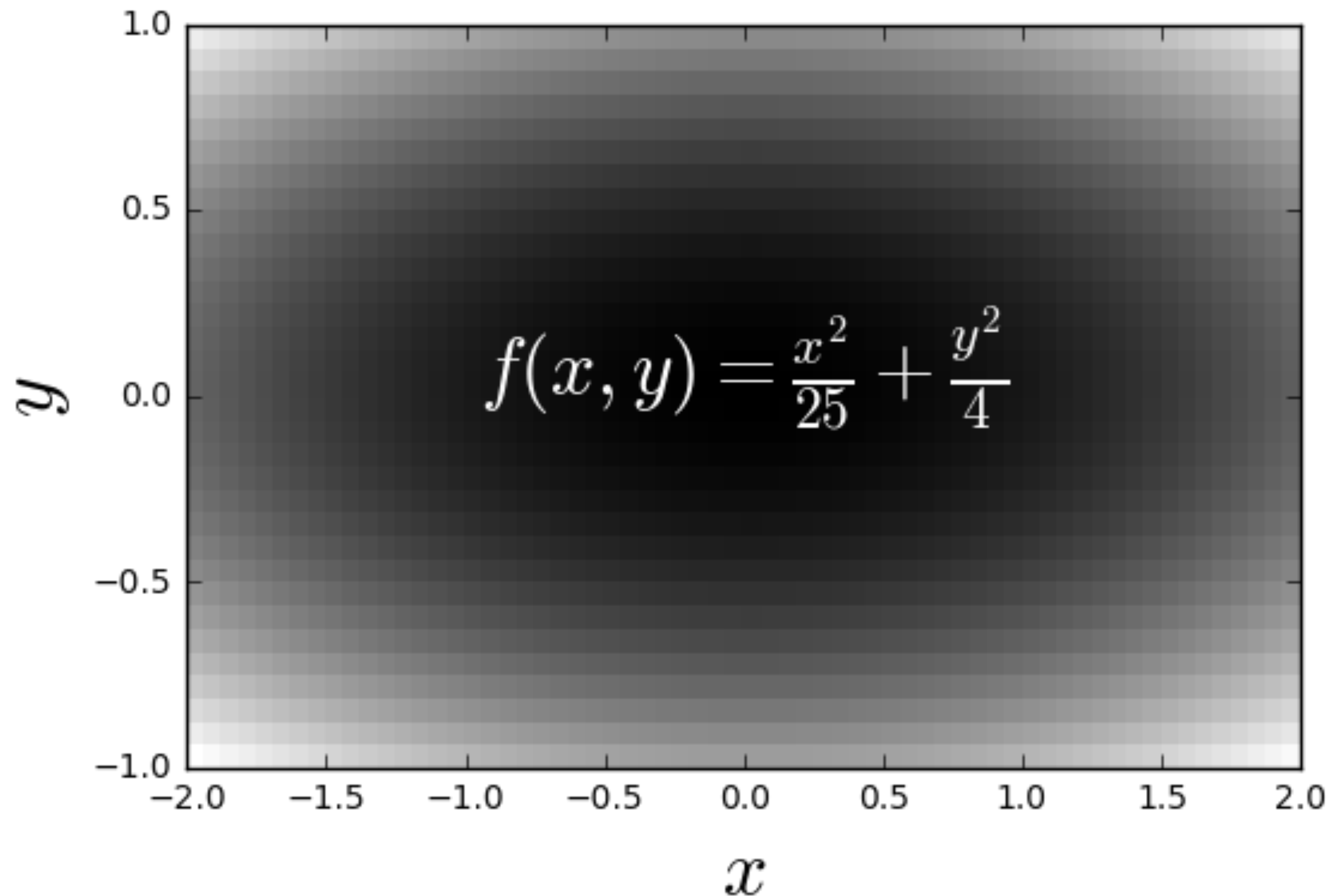
```
Z:
[[ 0.41    0.25    0.41   ]
 [ 0.2225  0.0625  0.2225]
 [ 0.16     0.     0.16   ]
 [ 0.2225  0.0625  0.2225]
 [ 0.41    0.25    0.41   ]]
```



Sampling on a grid



Sampling on a grid





Orientations of 2D arrays & images



orientation.py

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

Z = np.array([[1, 2, 3], [4, 5, 6]])
print('Z:\n', z)

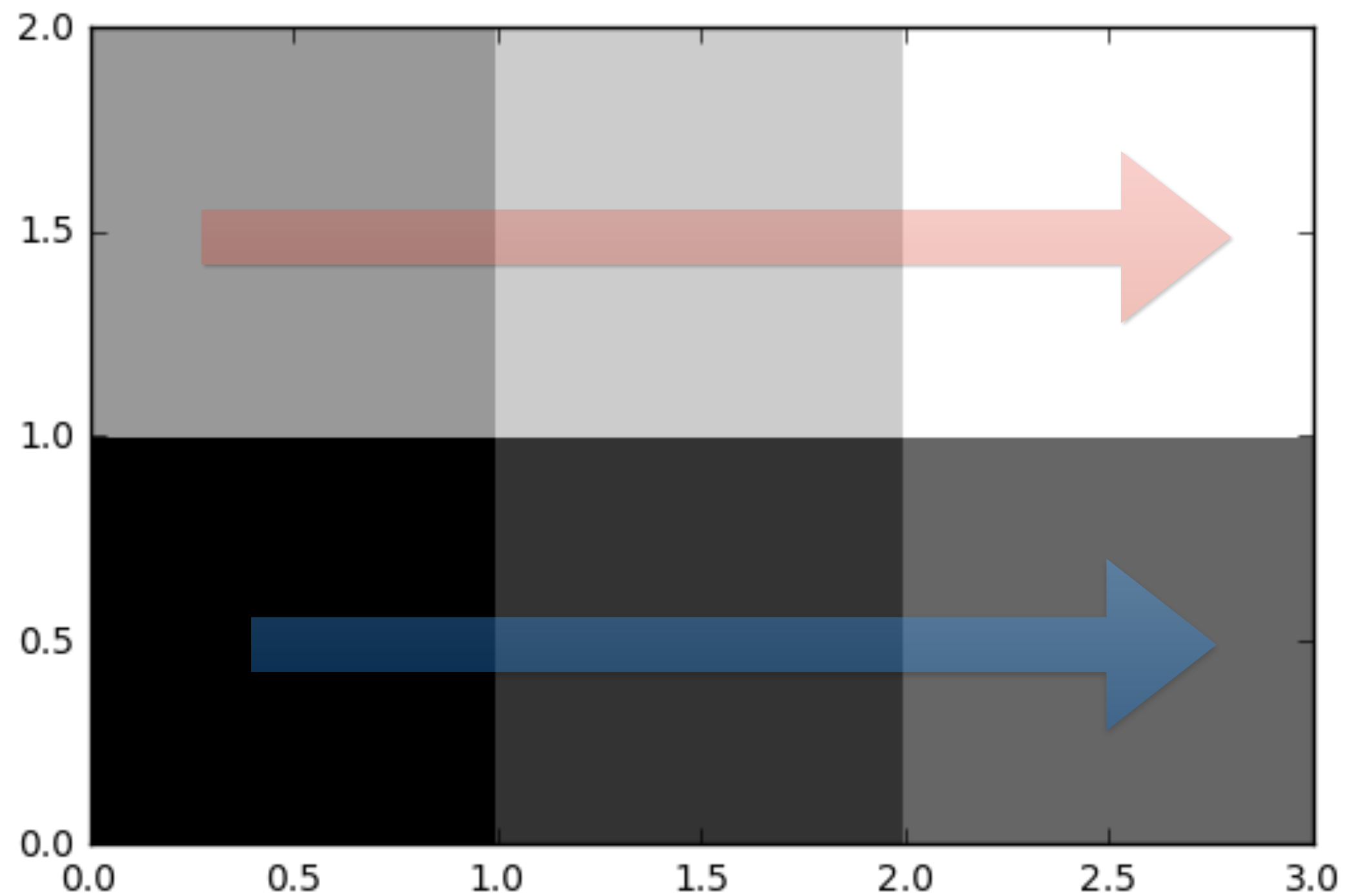
plt.pcolor(Z)
plt.show()
```

Orientations of 2D arrays & images

Output:

Z:

```
[[1 2 3] →  
 [4 5 6] →
```





INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Let's practice!

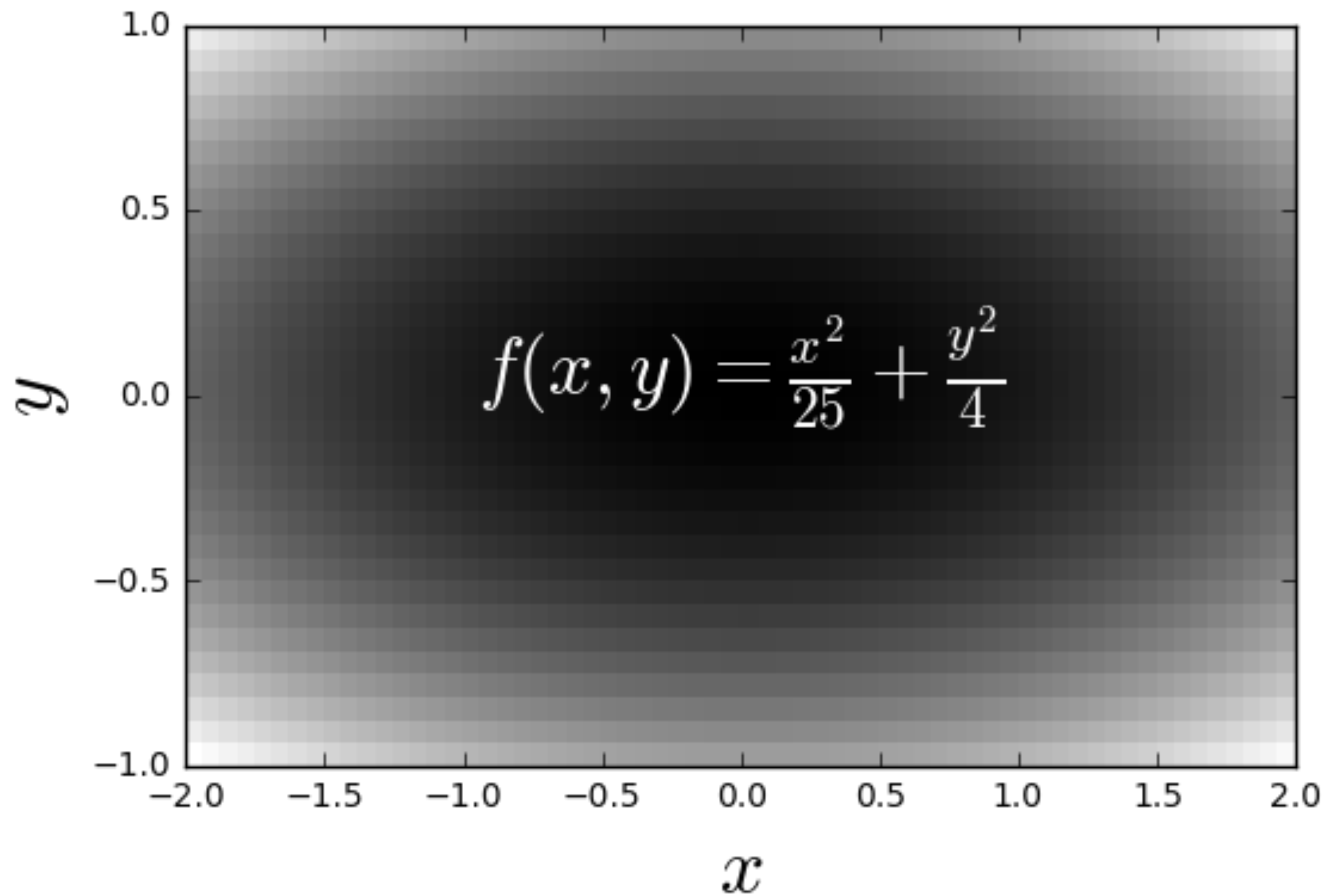


INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Visualizing bivariate functions



Bivariate functions





Pseudocolor plot

```
In [1]: import numpy as np
```

```
In [2]: import matplotlib.pyplot as plt
```

```
In [3]: u = np.linspace(-2, 2, 65)
```

```
In [4]: v = np.linspace(-1, 1, 33)
```

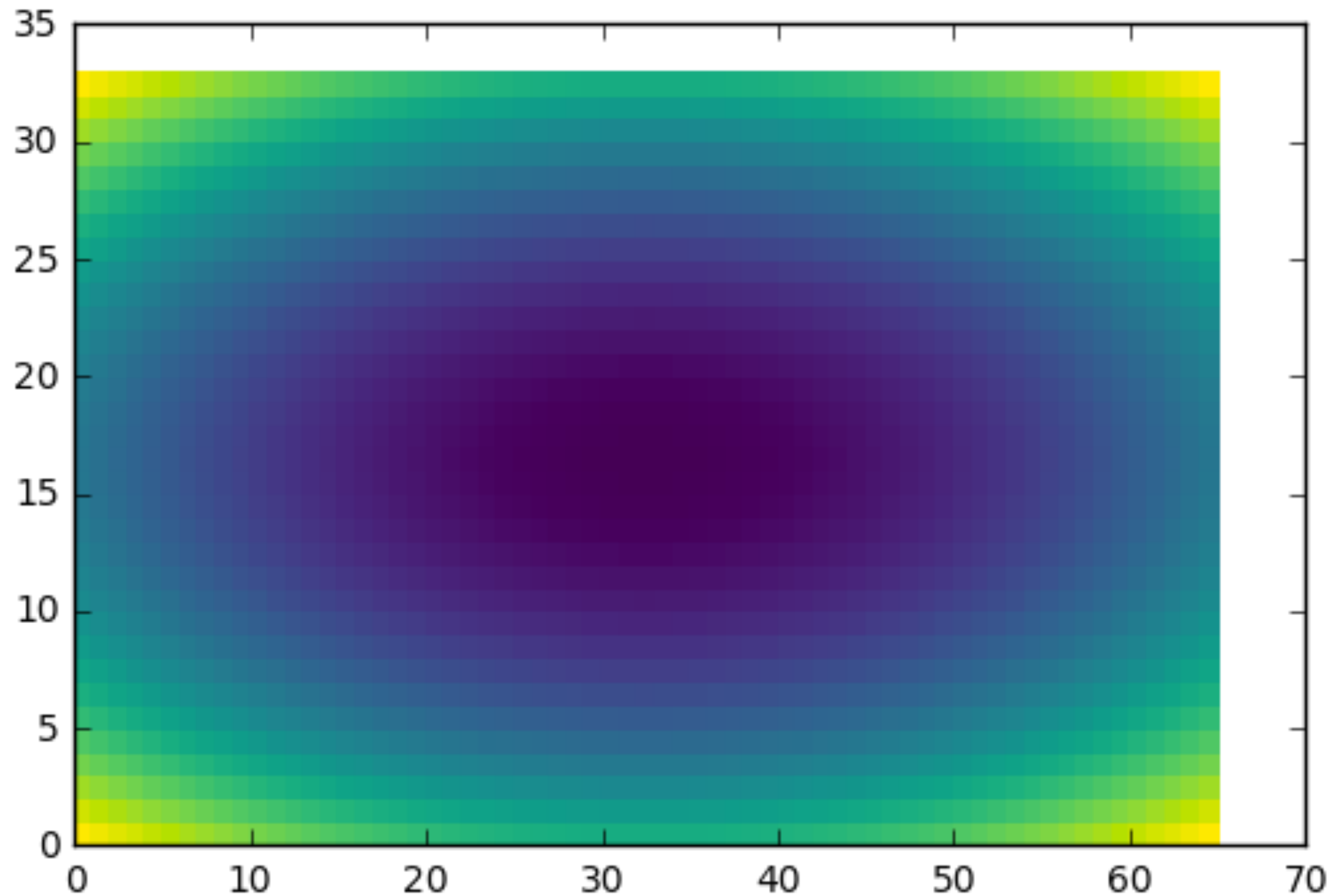
```
In [5]: X,Y = np.meshgrid(u, v)
```

```
In [6]: Z = X**2/25 + Y**2/4
```

```
In [7]: plt.pcolor(Z)
```

```
In [8]: plt.show()
```

Pseudocolor plot



Color bar

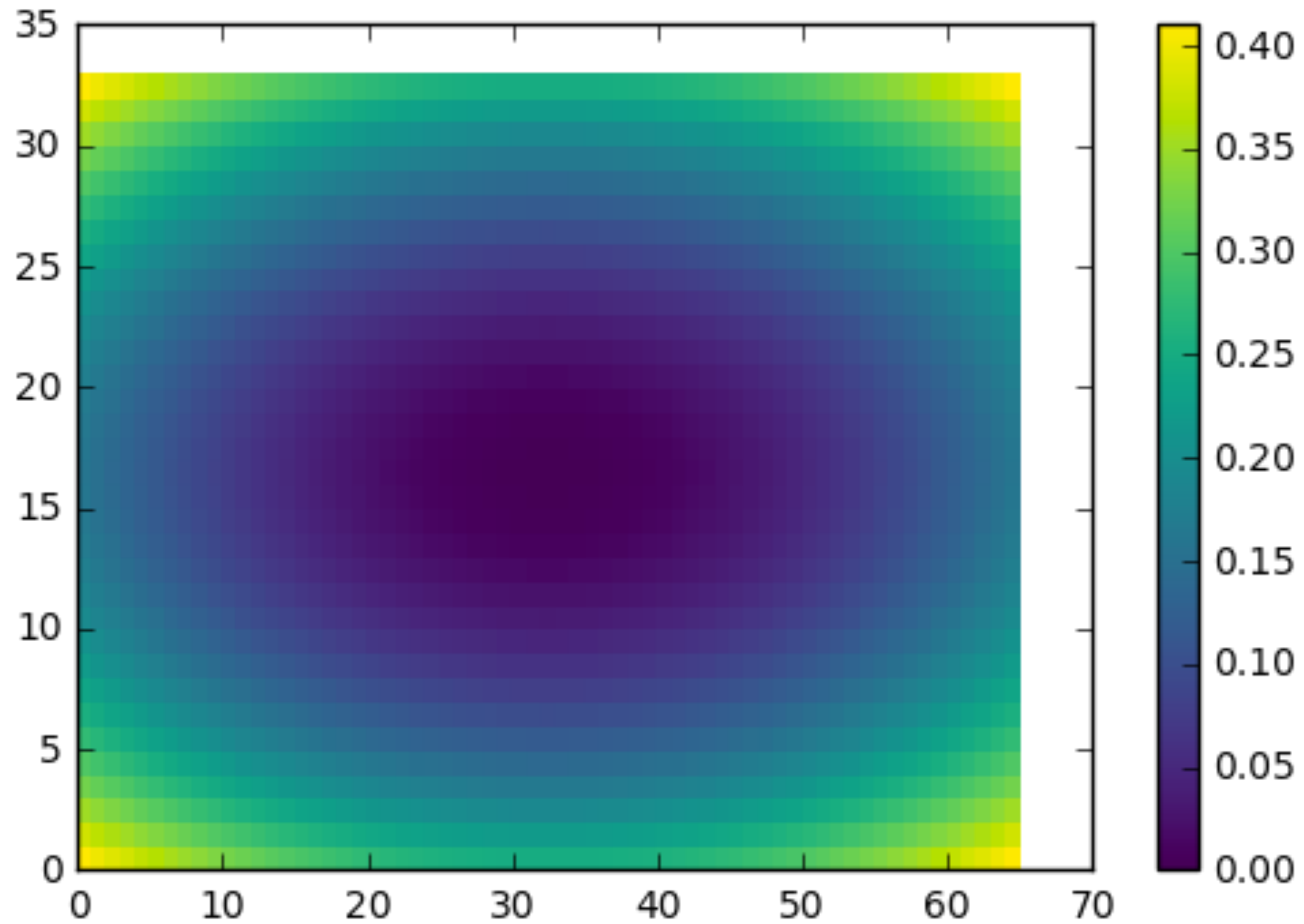
```
In [9]: plt.pcolor(Z)
```

```
In [10]: plt.colorbar()
```

```
In [11]: plt.show()
```



Color bar





Color map

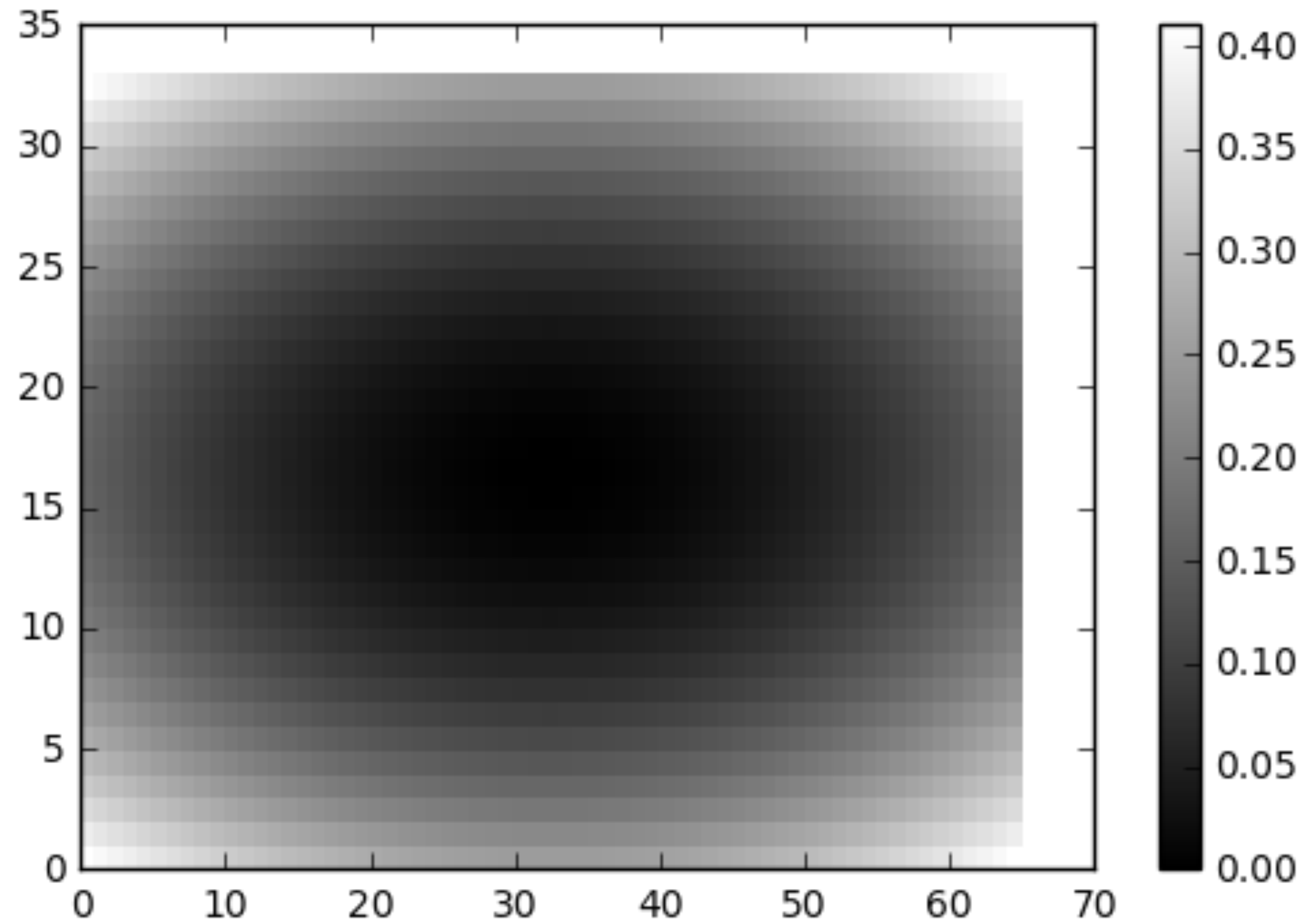
```
In [12]: plt.pcolor(Z, cmap= 'gray')
```

```
In [13]: plt.colorbar()
```

```
In [14]: plt.show()
```




Color map





Color map

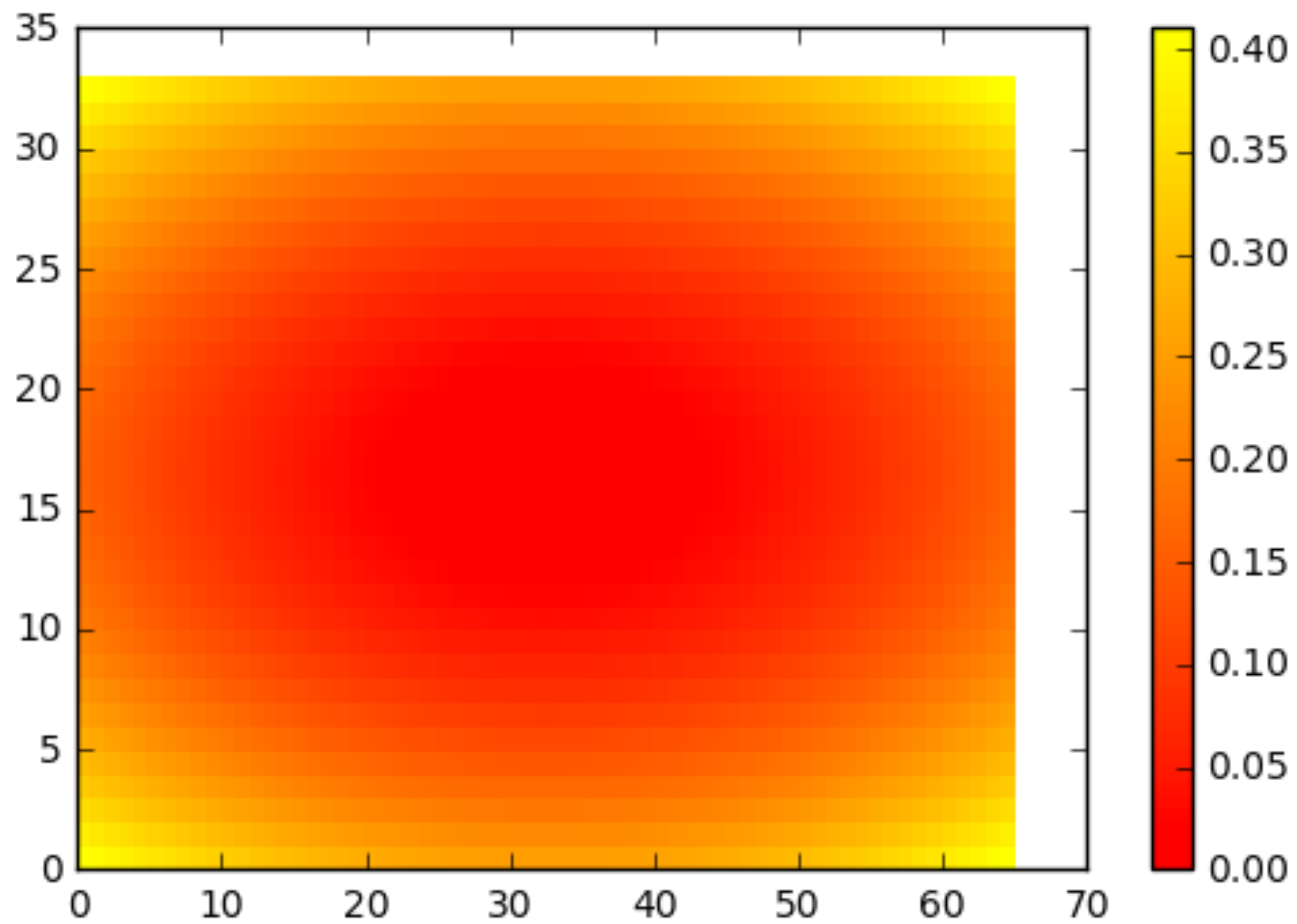
```
In [15]: plt.pcolor(Z, cmap= 'autumn')
```

```
In [16]: plt.colorbar()
```

```
In [17]: plt.show()
```



Color map





Axis tight

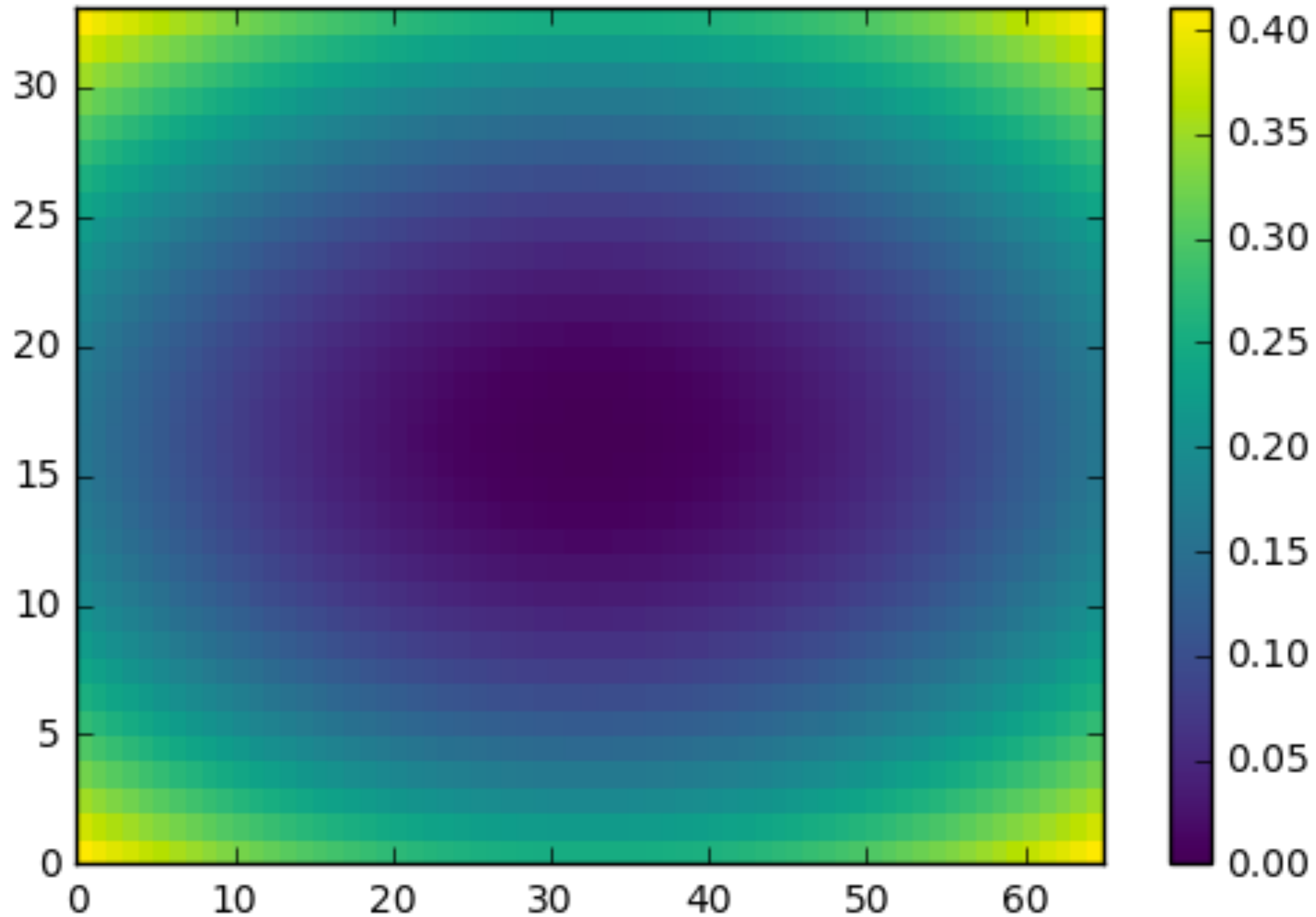
```
In [18]: plt.pcolor(Z)
```

```
In [19]: plt.colorbar()
```

```
In [20]: plt.axis('tight')
```

```
In [21]: plt.show()
```

Axis tight





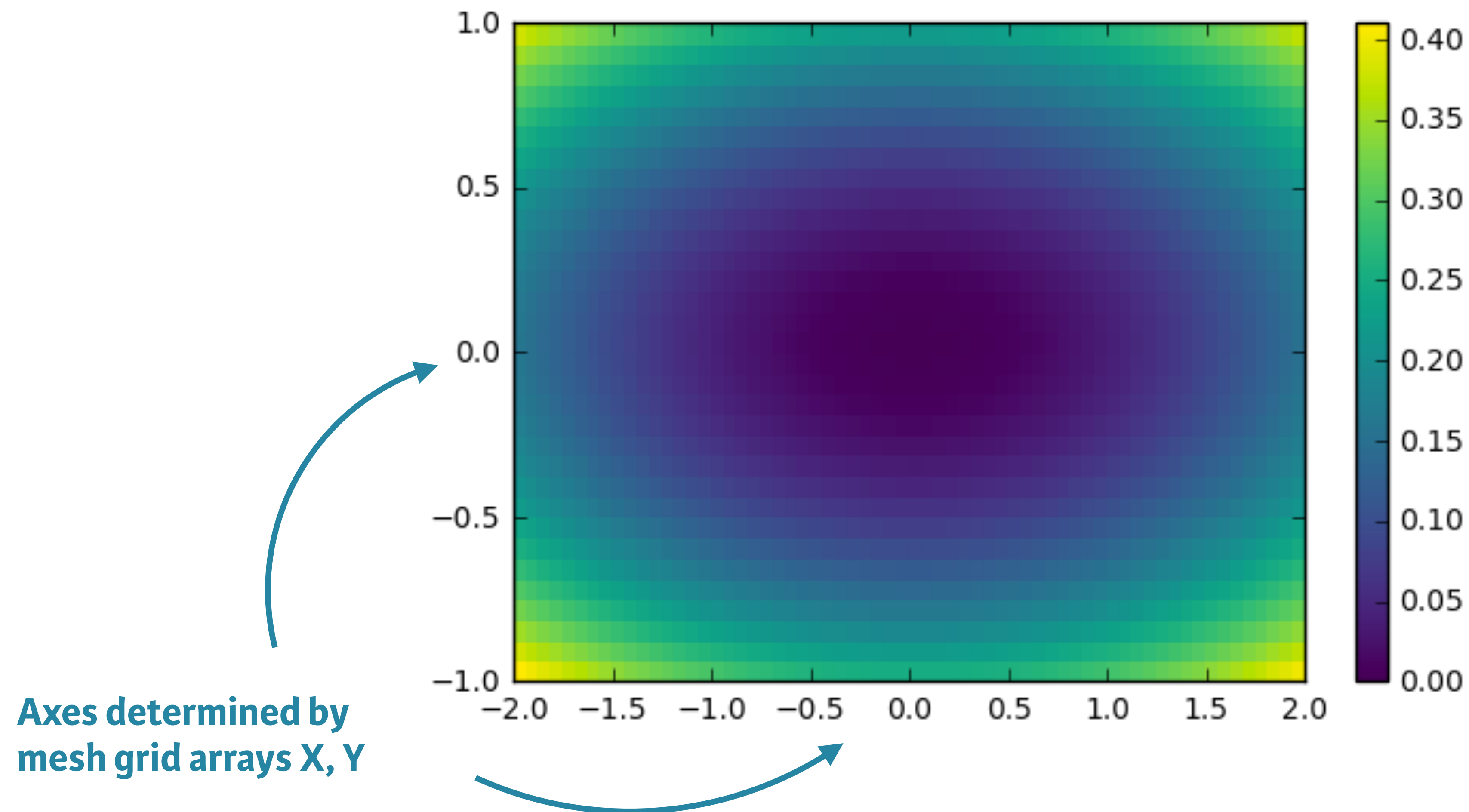
Plot using mesh grid

```
In [22]: plt.pcolor(X, Y, Z) # X, Y are 2D meshgrid
```

```
In [23]: plt.colorbar()
```

```
In [24]: plt.show()
```

Plot using mesh grid



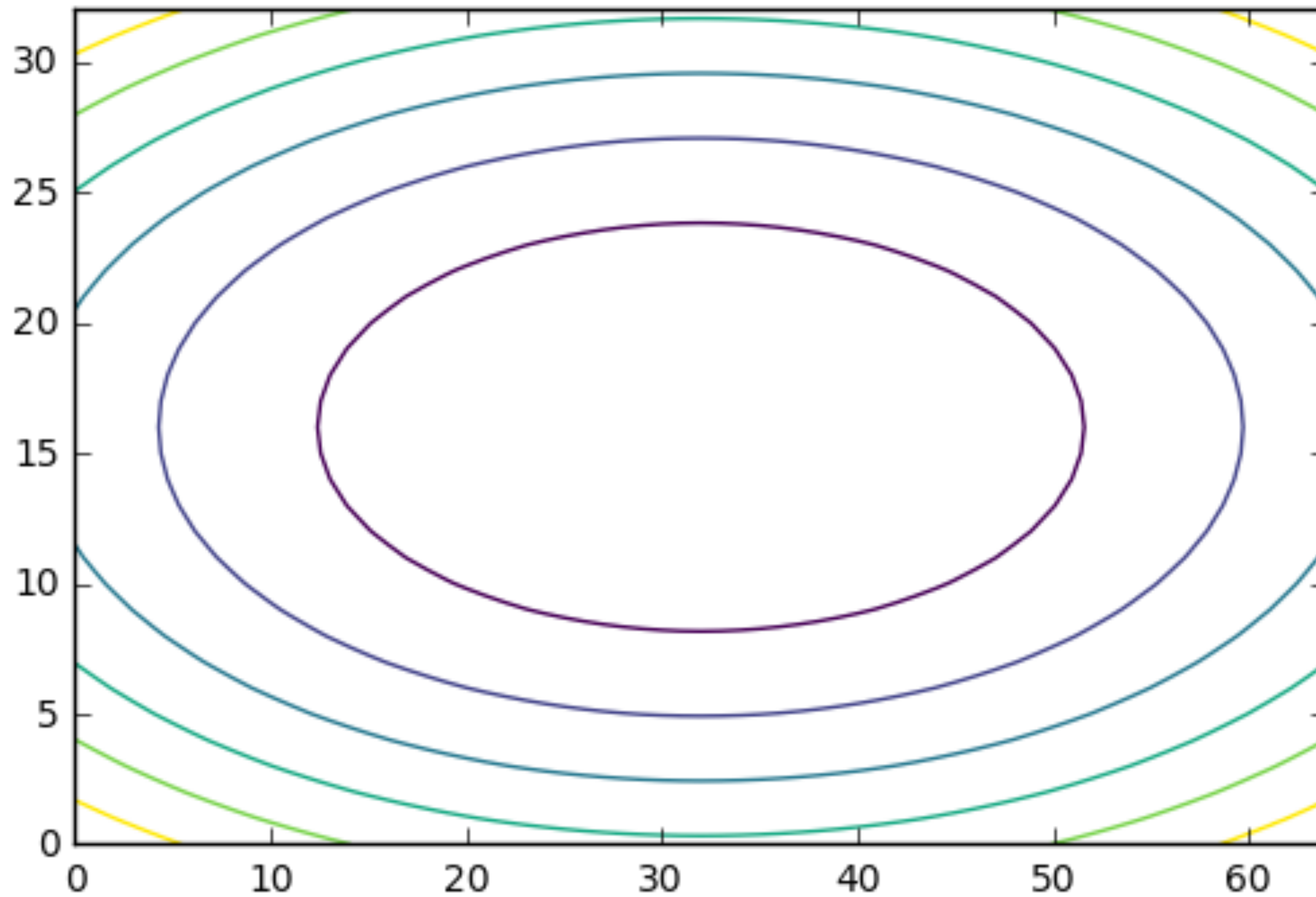
Contour plots

```
In [25]: plt.contour(Z)
```

```
In [26]: plt.show()
```



Contour plots





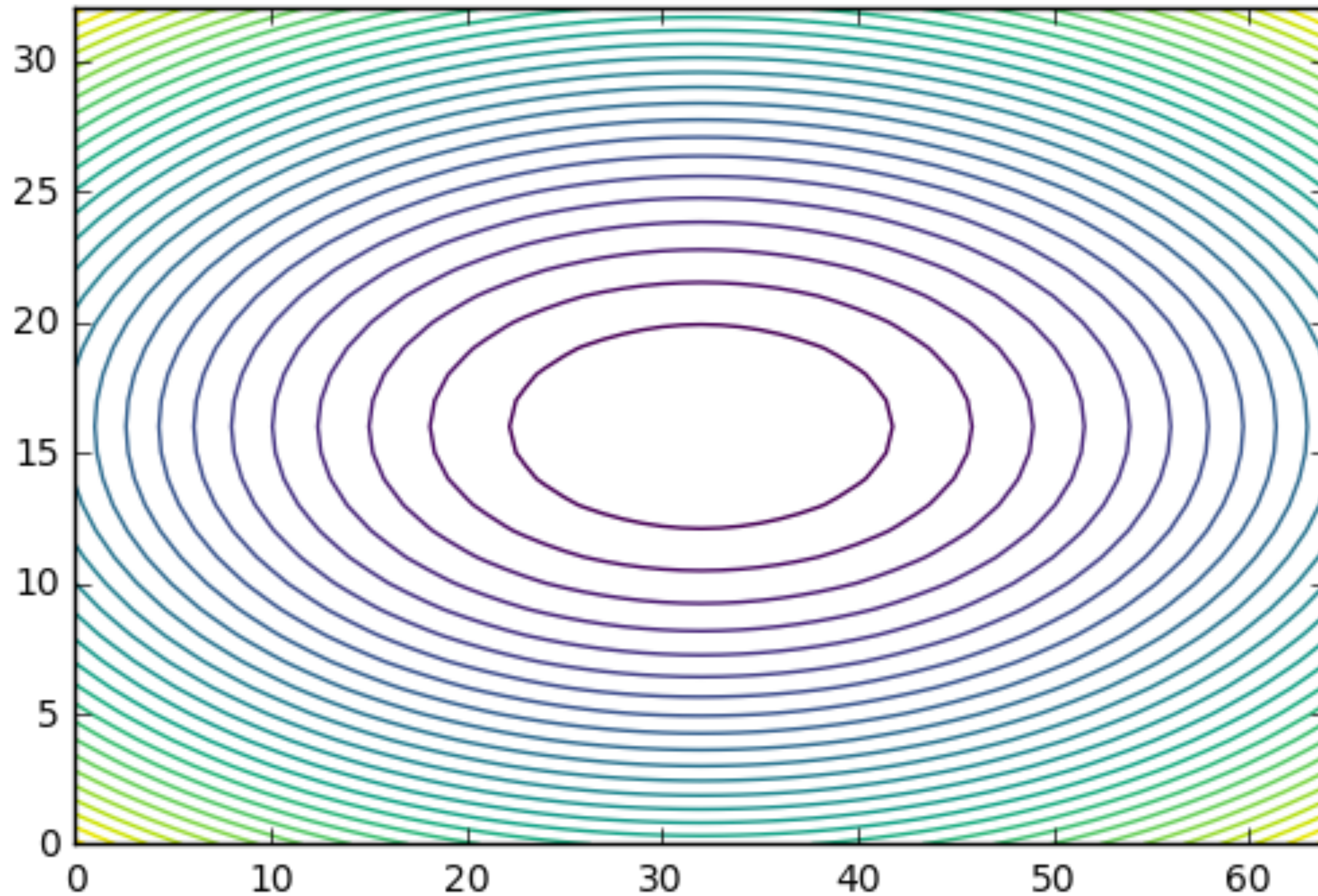
More contours

```
In [27]: plt.contour(Z, 30)
```

```
In [28]: plt.show()
```




More contours



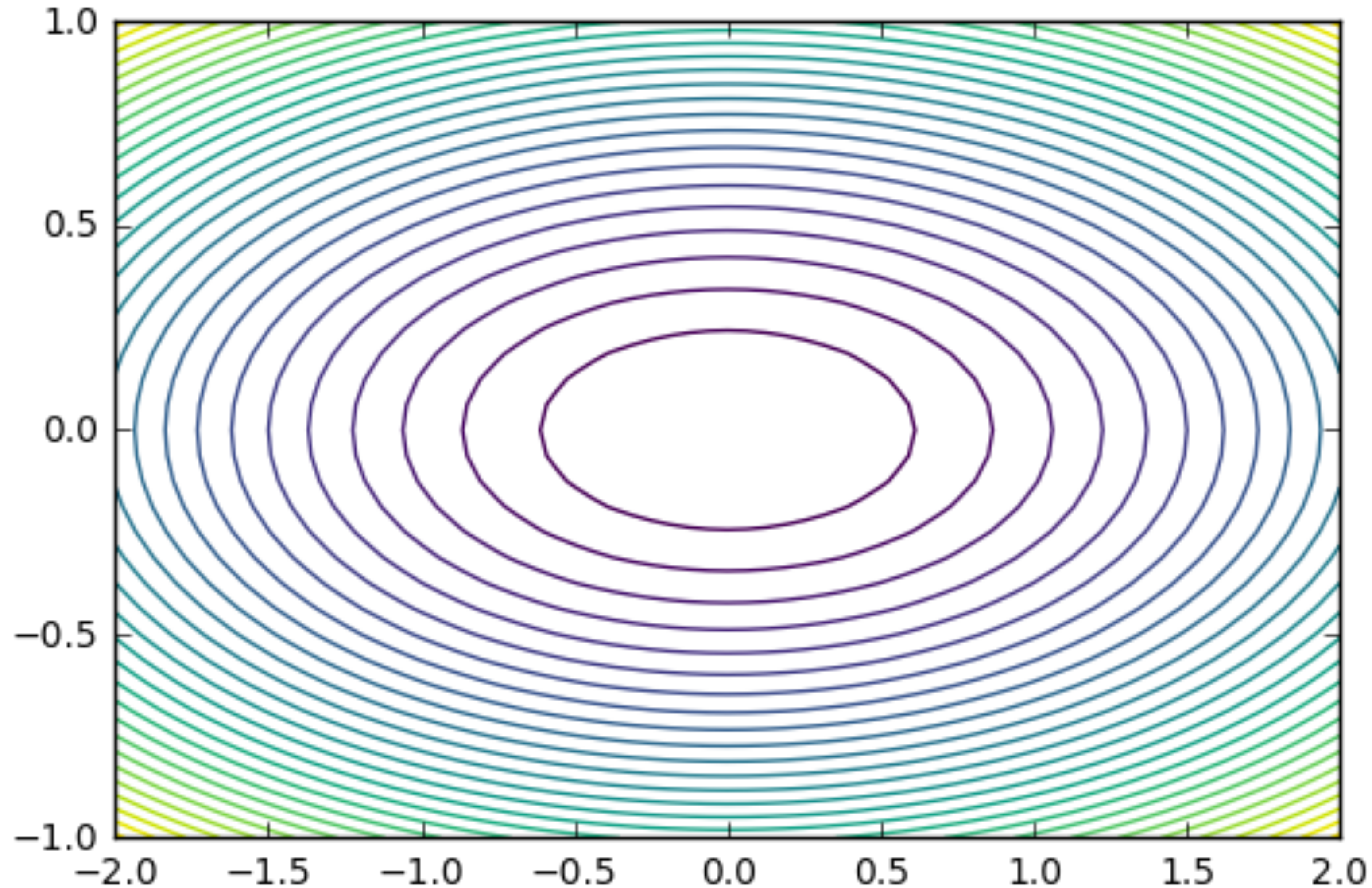


Contour plot using meshgrid

```
In [29]: plt.contour(X, Y, Z, 30)
```

```
In [30]: plt.show()
```


Contour plot using meshgrid





Filled contour plots

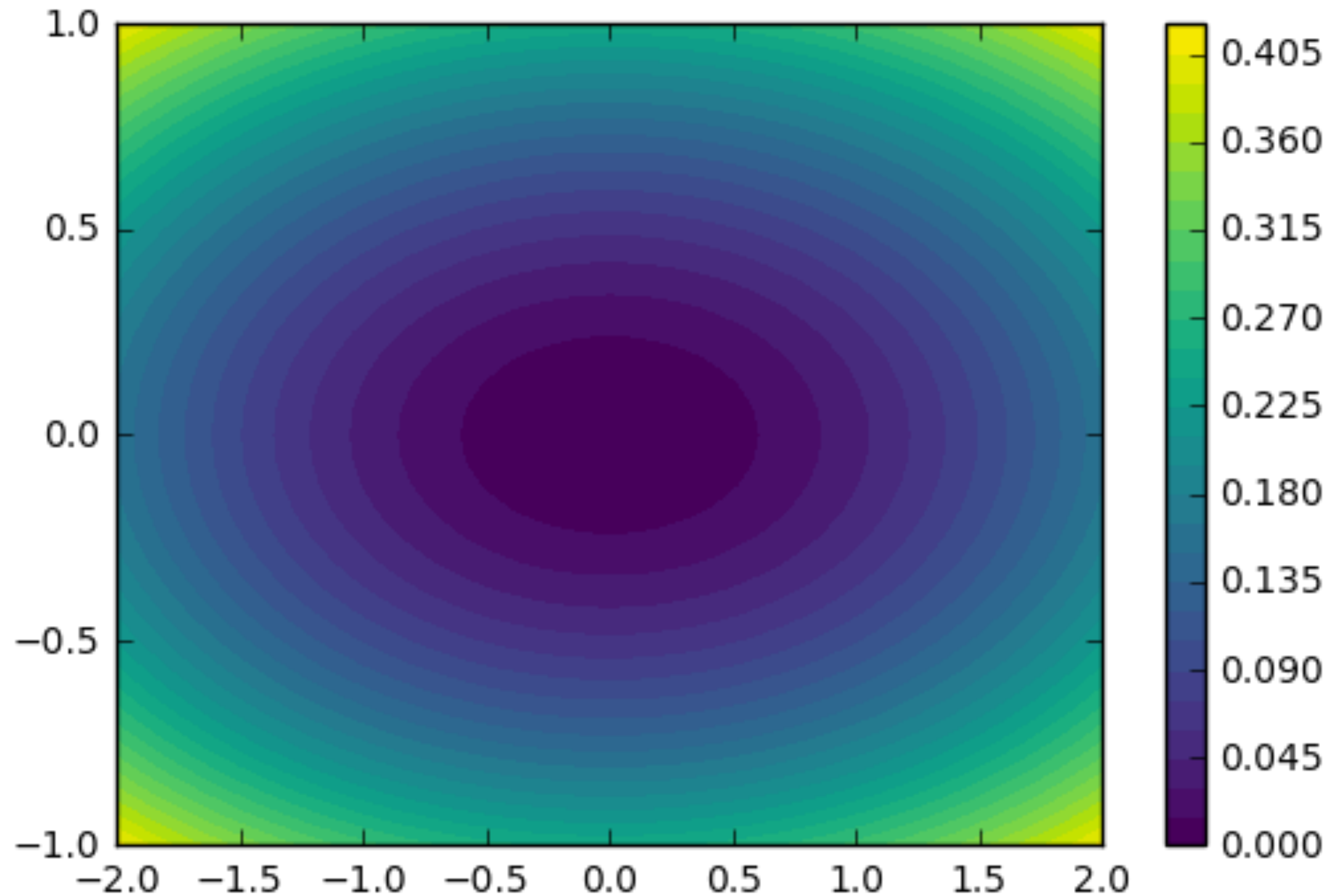
```
In [31]: plt.contourf(X, Y, Z, 30)
```

```
In [32]: plt.colorbar()
```

```
In [33]: plt.show()
```




Filled contour plots



More information

- API has many (optional) keyword arguments
- More in matplotlib.pyplot documentation
- More examples: <http://matplotlib.org/gallery.html>



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Let's practice!

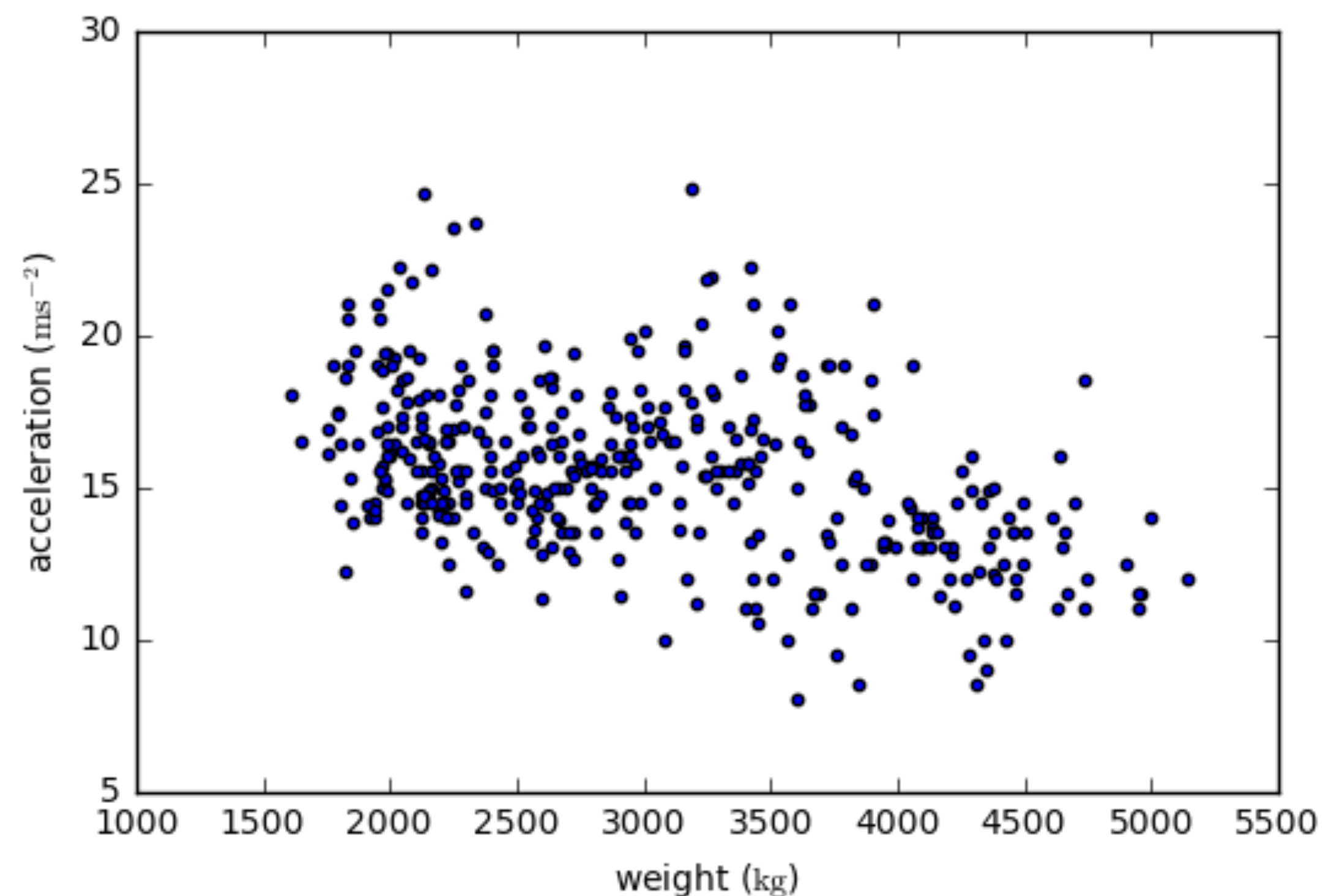


INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Visualizing bivariate distributions

Distributions of 2D points

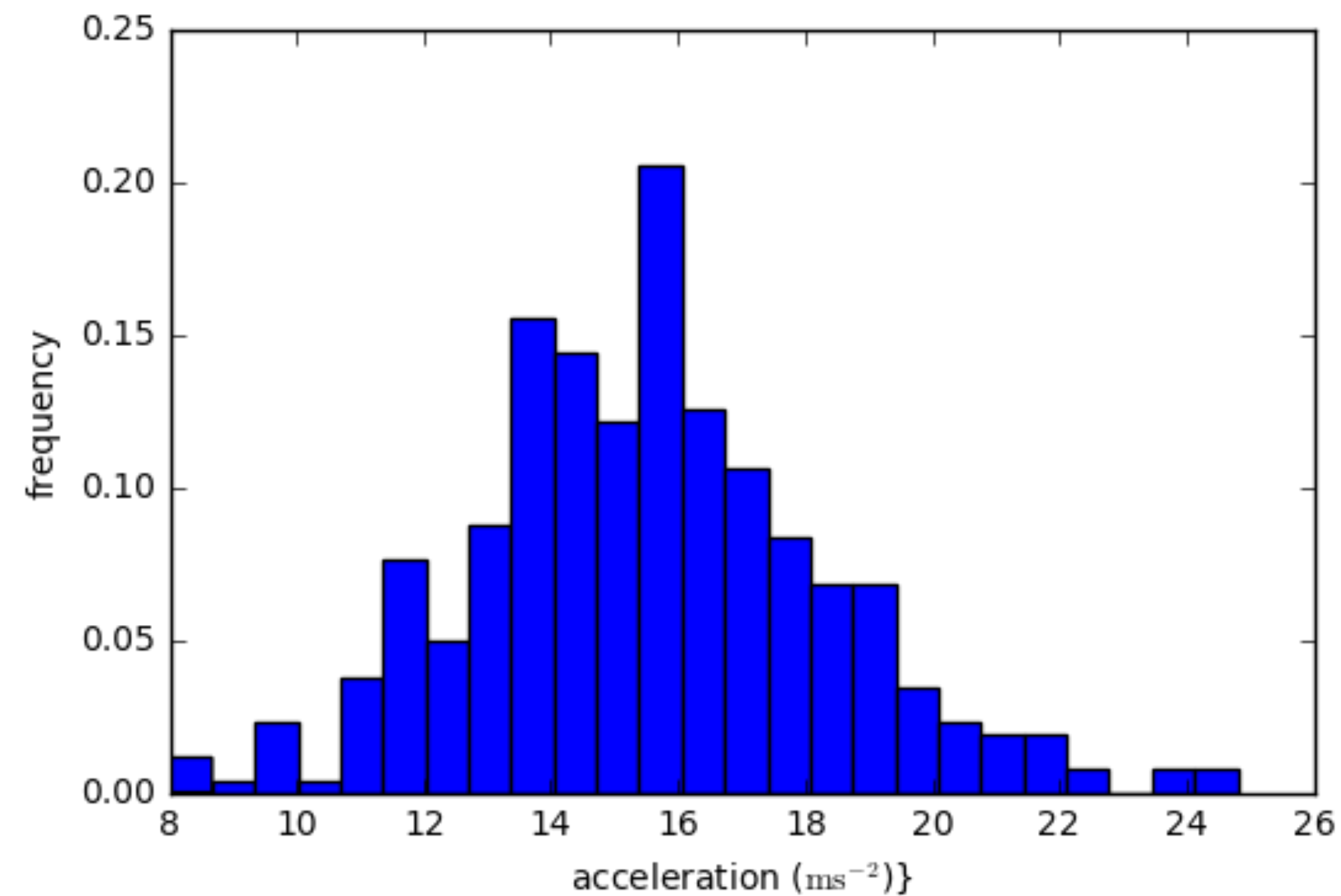
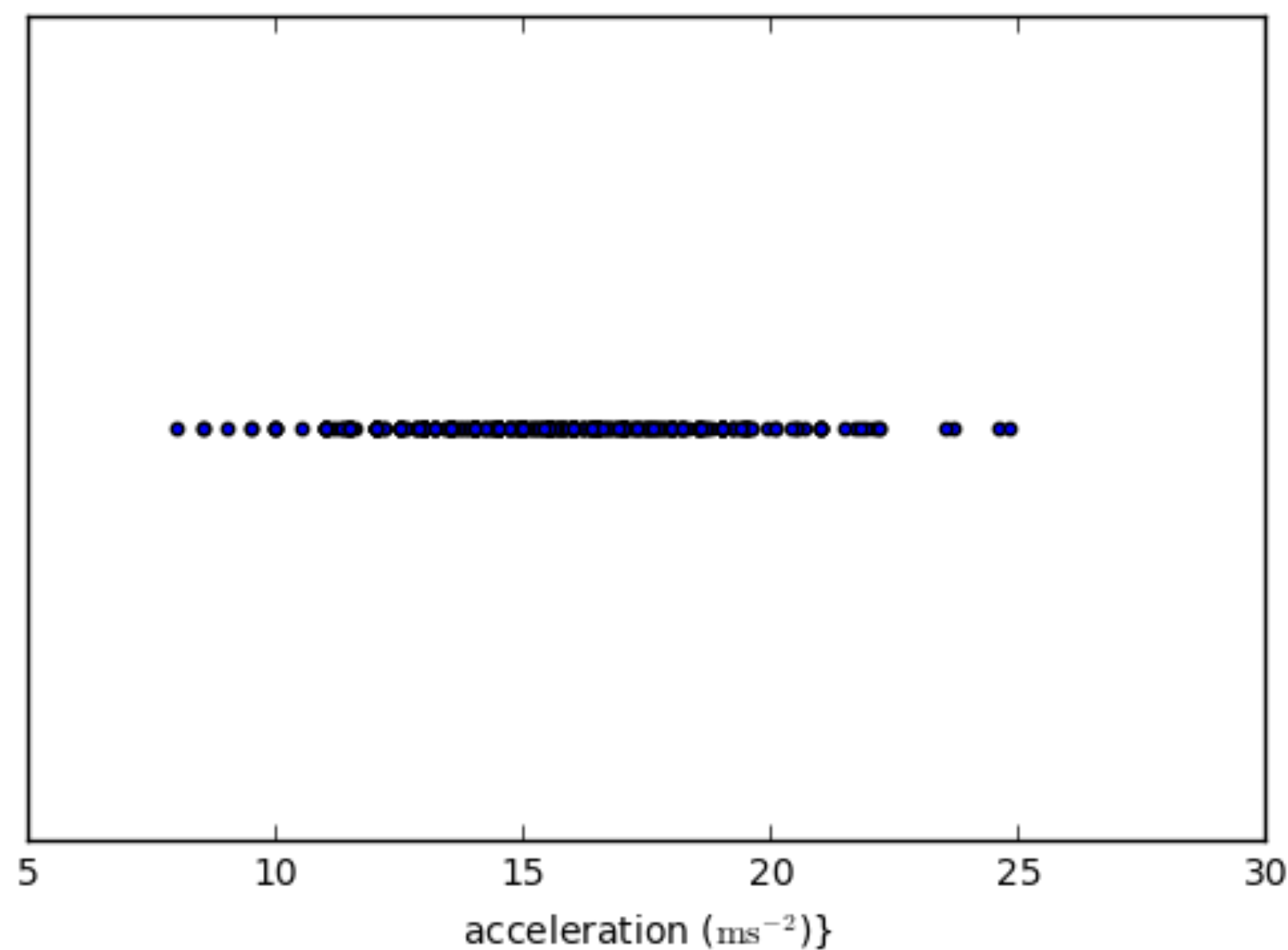
- 2D points given as two 1D arrays x & y
- Goal: generate a 2D histogram from x & y





Histograms in 1D

- Choose bins (intervals)
- Count realizations within bins & plot





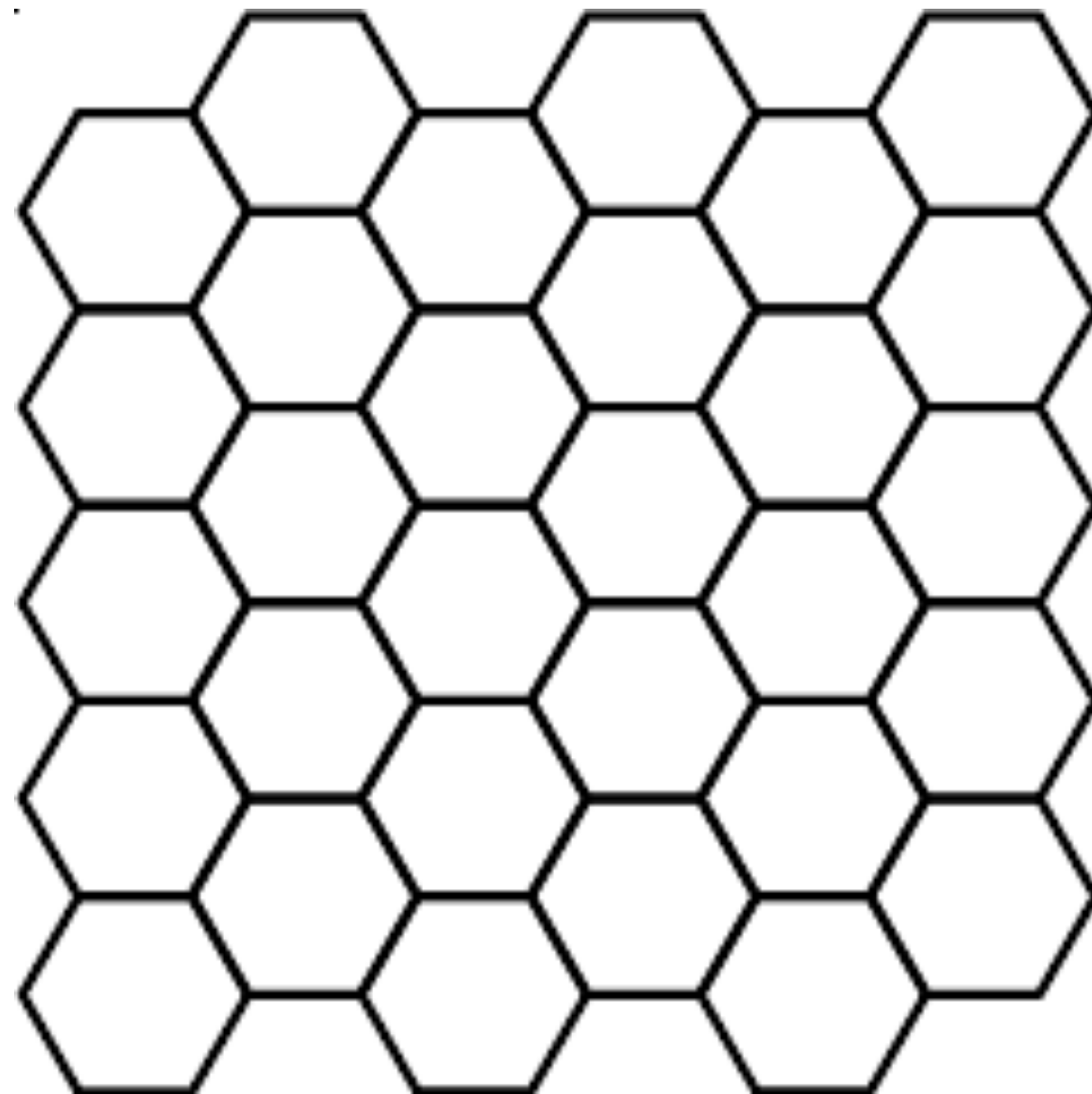
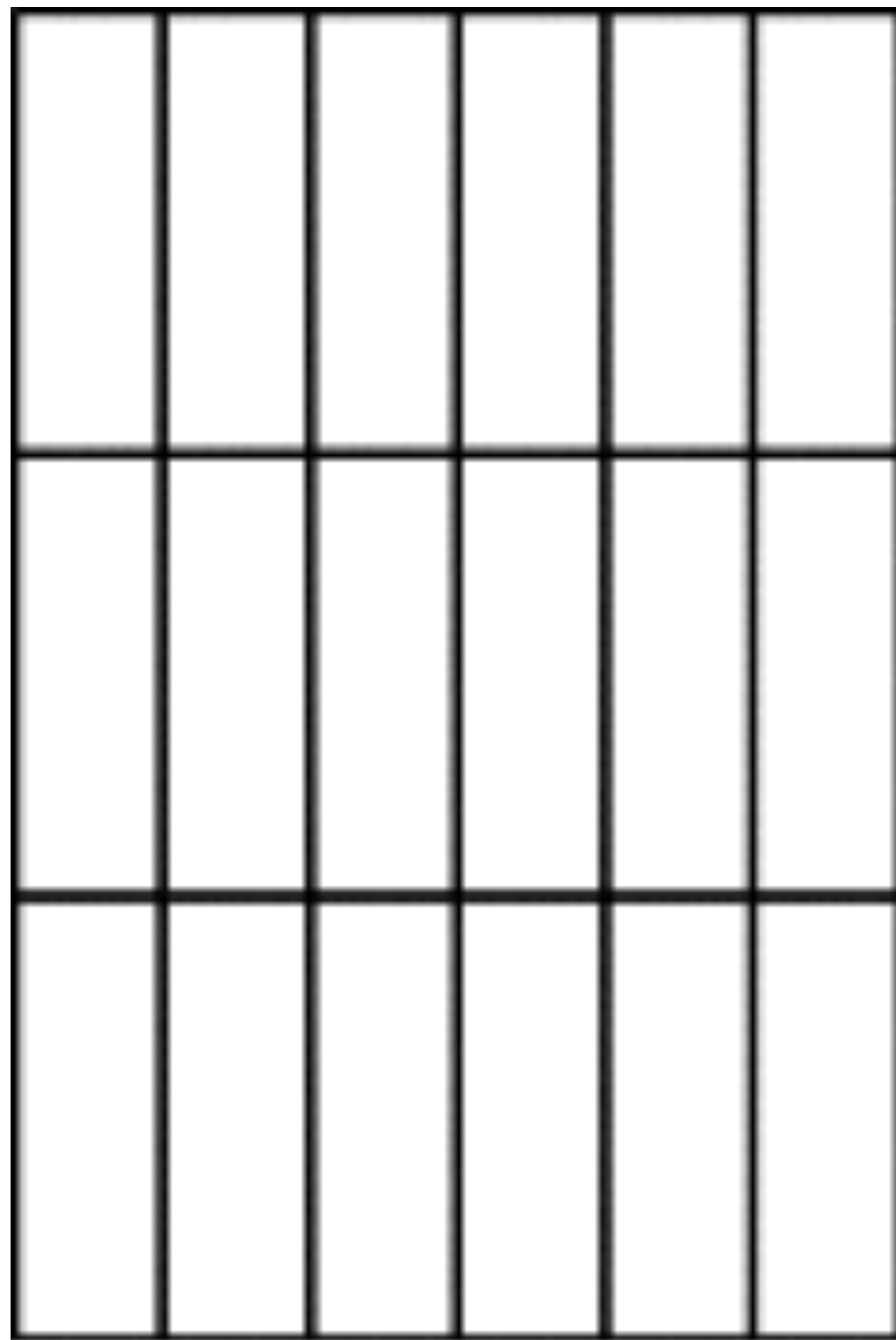
Histograms in 1D

```
In [1]: counts, bins, patches = plt.hist(x, bins=25)
```

```
In [2]: plt.show()
```


Bins in 2D

- Different shapes available for binning points
- Common choices: rectangles & hexagons





hist2d(): Rectangular binning

```
In [1]: plt.hist2d(x, y, bins=(10, 20)) # x & y are 1D arrays of  
....: same length
```

```
In [2]: plt.colorbar()
```

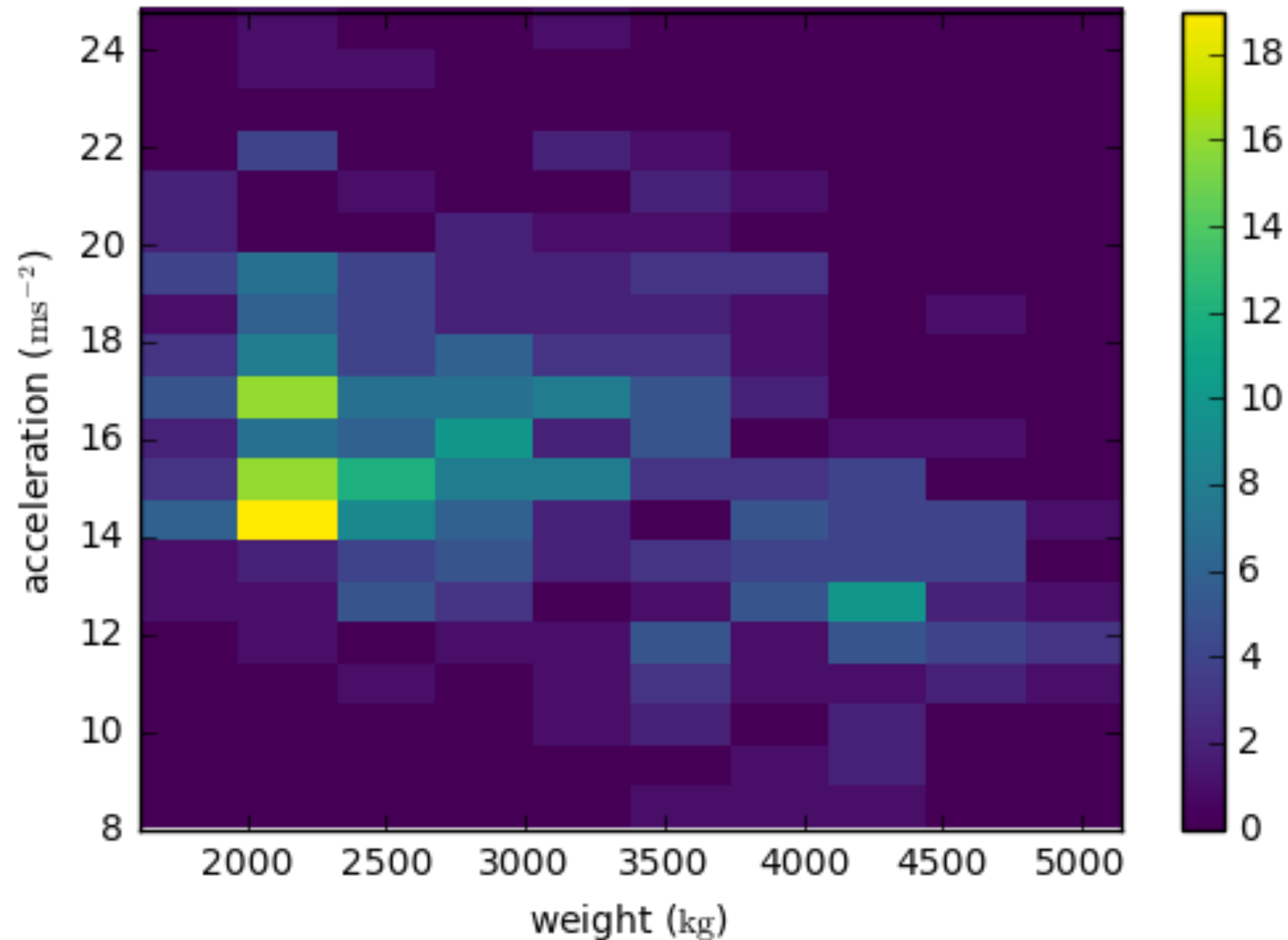
```
In [3]: plt.xlabel('weight ( $\mathrm{kg}$ )')
```

```
In [4]: plt.ylabel('acceleration ( $\mathrm{ms}^{-2}$ )')
```

```
In [5]: plt.show()
```



hist2d(): Rectangular binning





hexbin(): Hexagonal binning

```
In [1]: plt.hexbin(x, y, gridsize=(15,10))
```

```
In [2]: plt.colorbar()
```

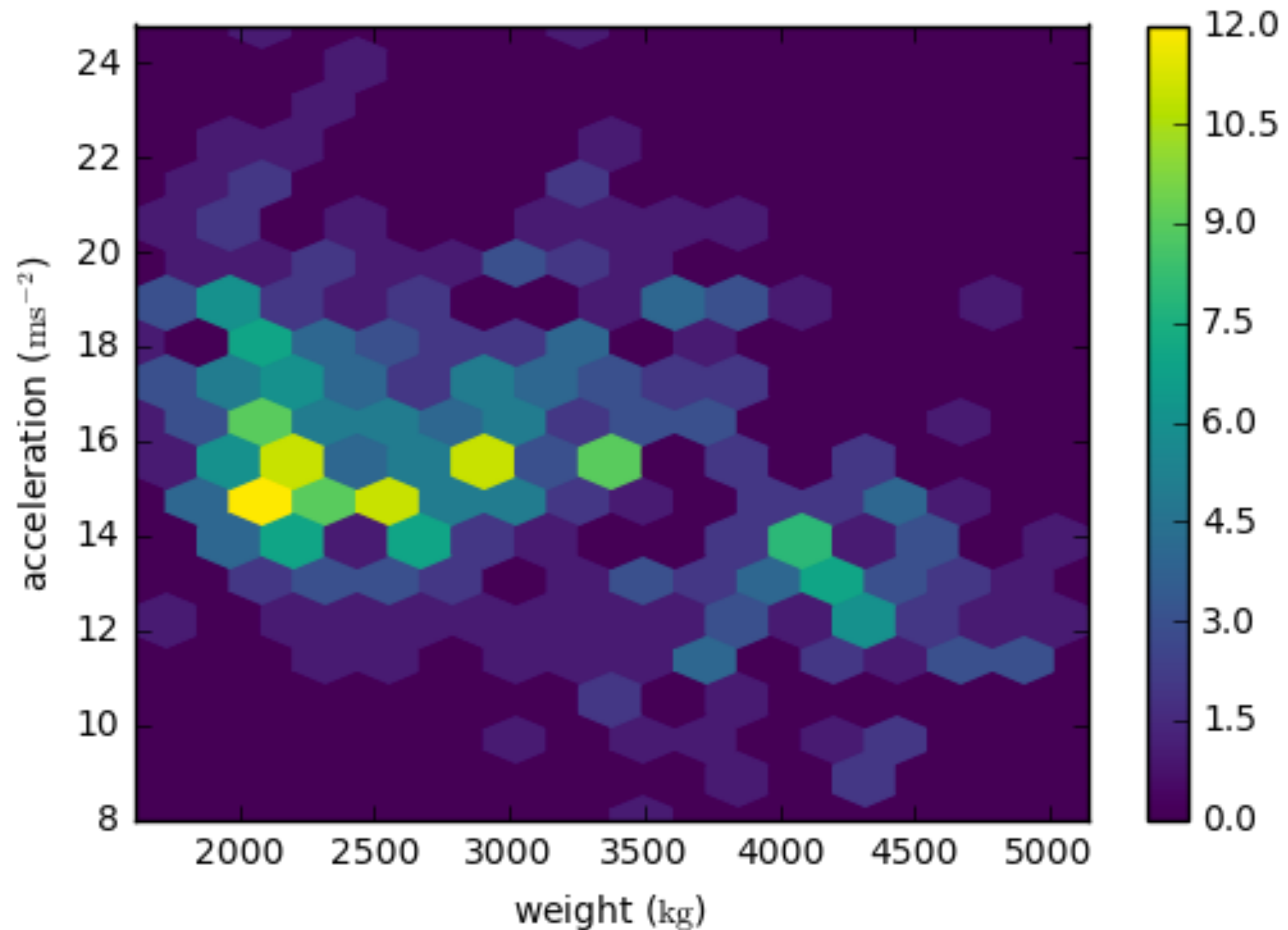
```
In [3]: plt.xlabel('weight ( $\mathrm{kg}$ )')
```

```
In [4]: plt.ylabel('acceleration ( $\mathrm{ms}^{-2}$ )')
```

```
In [5]: plt.show()
```



hexbin(): Hexagonal binning





INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Let's practice!



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Working with images



Images

- Grayscale images: rectangular 2D arrays
- Color images: typically three 2D arrays (channels)
 - RGB (Red-Green-Blue)
- Channel values:
 - 0 to 1 (floating-point numbers)
 - 0 to 255 (8 bit integers)



Loading images

```
In [1]: img = plt.imread('sunflower.jpg')
```

```
In [2]: print(img.shape)  
(480, 640, 3)
```

```
In [3]: plt.imshow(img)
```

```
In [4]: plt.axis('off')
```

```
In [5]: plt.show()
```



Loading images





Reduction to gray-scale image

```
In [6]: collapsed = img.mean(axis=2)
```

```
In [7]: print(collapsed.shape)
(480, 640)
```

```
In [8]: plt.set_cmap('gray')
```

```
In [9]: plt.imshow(collapsed, cmap='gray')
```

```
In [10]: plt.axis('off')
```

```
In [11]: plt.show()
```




Reduction to gray-scale image





Uneven samples

```
In [12]: uneven = collapsed[:,::4,::2] # nonuniform subsampling
```

```
In [13]: print(uneven.shape)
(120, 320)
```

```
In [14]: plt.imshow(uneven)
```

```
In [15]: plt.axis('off')
```

```
In [16]: plt.show()
```

Uneven samples





Adjusting aspect ratio

```
In [17]: plt.imshow(uneven, aspect=2.0)
```

```
In [18]: plt.axis('off')
```

```
In [19]: plt.show()
```



Adjusting aspect ratio





Adjusting extent

```
In [20]: plt.imshow(uneven, cmap='gray', extent=(0,640,0,480))
```

```
In [21]: plt.axis('off')
```

```
In [22]: plt.show()
```



Adjusting extent





INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

Let's practice!