Python_course_4_MatPlotLib

May 7, 2021

1 Python course 4: Introduction to MatPlotLib

by Markus Reinert, 2021-05-07

1.1 Import the necessary module(s)

There are two ways to load MatPlotLib. They are equivalent, just choose whatever looks consistent in your case.

[1]:
```
from matplotlib import pyplot as plt
# or shorter: import matplotlib.pyplot as plt
```

We will also use NumPy extensively in this course. The module cmocean provides very nice colourmaps for oceanography. An overview can be found here: https://matplotlib.org/cmocean/

[2]:
```
import numpy as np
import cmocean
```

1.2 Configure the visualization

This is a way to change how *all* the plots in the Python script look like. It sets the default settings for a figure. You can still change individual aspects of any plot.

[3]:
```
plt.rcParams["font.size"] = 14
```

To see all parameters that can be modified and their default values, change the following cell from Raw to Code and run it. plt.rcParams

1.3 Plots of 2D data

1.3.1 Line plots

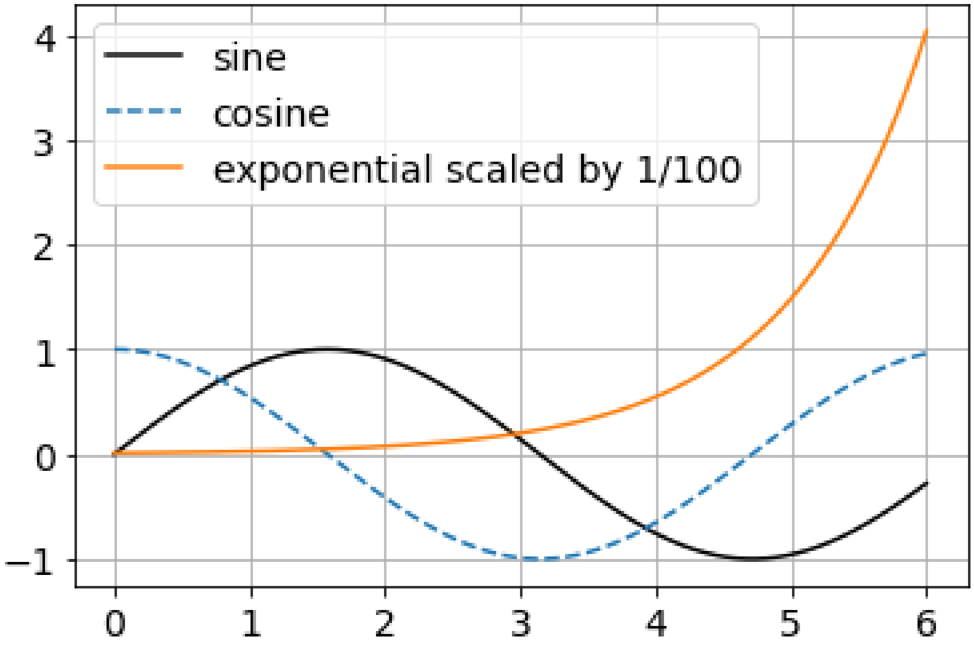
[4]:
```
# Create an x-axis or x-coordinates
x = np.linspace(0, 6, 10000)
# Create some function data
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.exp(x)
```

[5]:

```
plt.plot(x, y1, label="sine", color="k")  # k = key stands for black color, see https://gearside.com/color-black-represented-k-cmyk/
plt.plot(x, y2, "--", label="cosine")  # make it dashed
plt.plot(x, y3  / 100, label="exponential scaled by 1/100")

plt.legend()

plt.grid()
```



1.3.2 Scatter plots

There are two ways to make scatter plots, depending on what kind of information you want to show.

[6]:

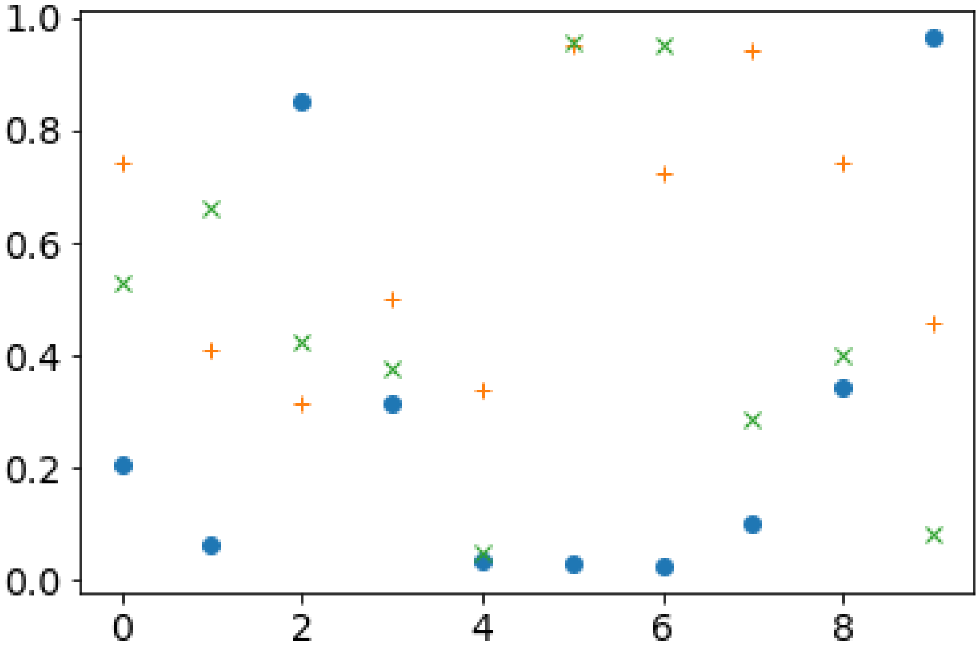
```
# Create some random data for scatter plots
x_r = np.arange(10)
y_r1 = np.random.random(x_r.size)
y_r2 = np.random.random(x_r.size)
y_r3 = np.random.random(x_r.size)
```

[7]:

```
# Show the random data with different symbols
plt.plot(x_r, y_r1, "o")
plt.plot(x_r, y_r2, "+")
plt.plot(x_r, y_r3, "x")
```

[7]:

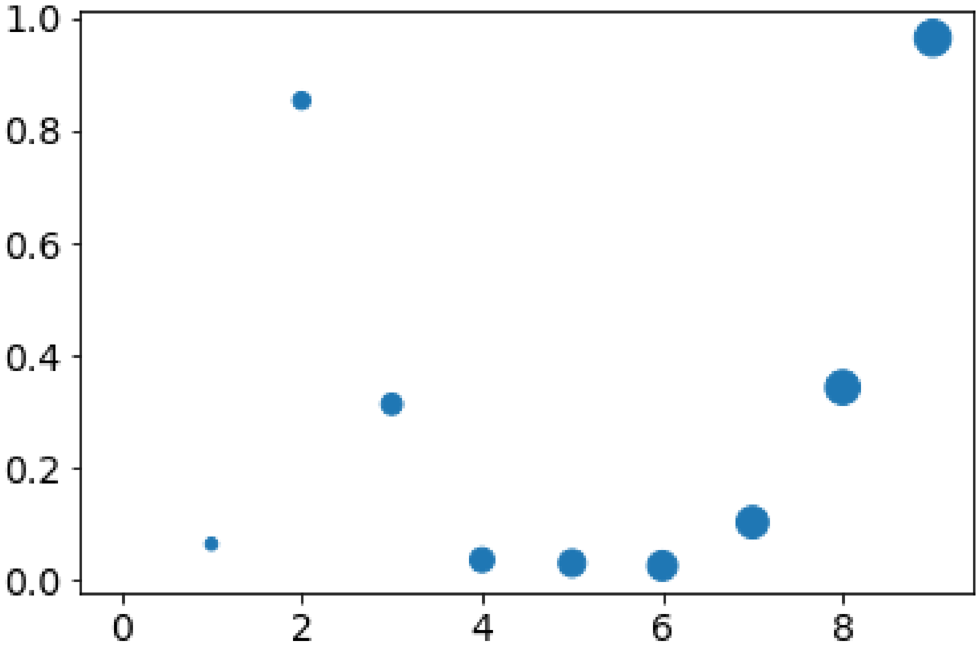
```
[<matplotlib.lines.Line2D at 0x7f287f6bd750>]
```



If there is some third kind of data associated with the x- and y-coordinates, use the scatter function.

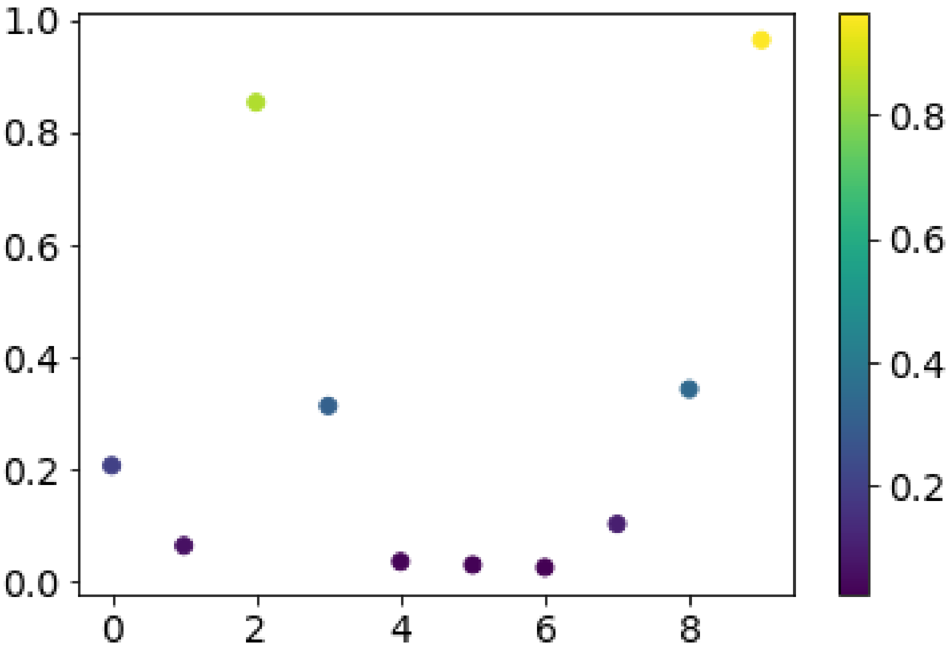
[8]:
```
# Represent x-value by size
plt.scatter(x_r, y_r1, s=20*x_r)
```

[8]:
```
<matplotlib.collections.PathCollection at 0x7f287f637310>
```



[9]:
```
# Represent y-value by color
plt.scatter(x_r, y_r1, c=y_r1)
plt.colorbar()
```

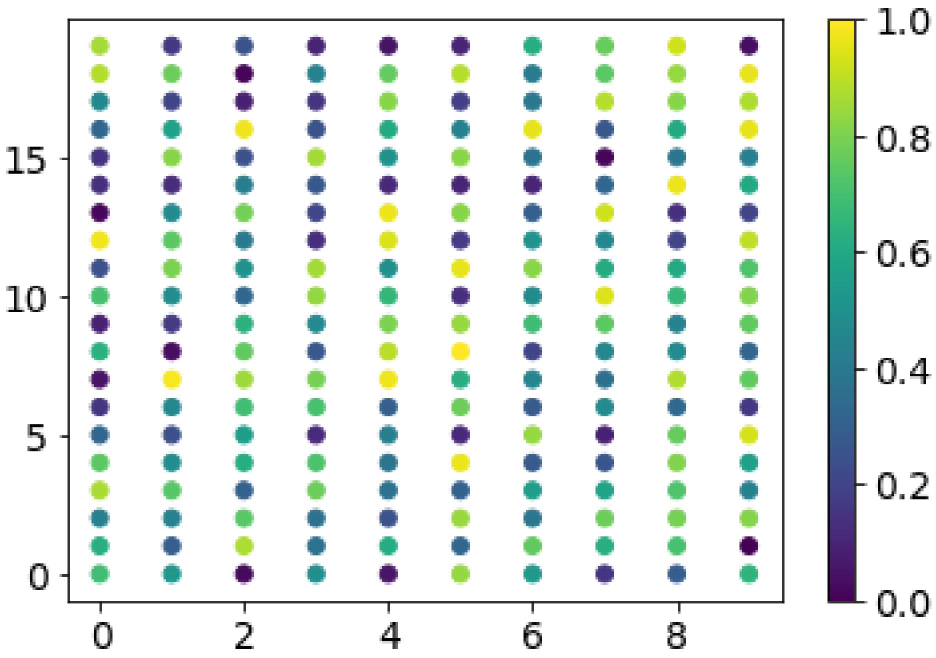
[9]:
```
<matplotlib.colorbar.Colorbar at 0x7f287f5d77d0>
```



Here is an example with 3 unrelated quantities:

[10]:
```
# Create 2D equally spaced coordinates in x and y
X, Y = np.meshgrid(np.arange(10), np.arange(20))
# Create random data for every point
Z = np.random.random(X.shape)
# Show it in color
plt.scatter(X, Y, c=Z, vmin=0, vmax=1)
plt.colorbar()
```

[10]:
```
<matplotlib.colorbar.Colorbar at 0x7f287f4ac290>
```



In the above example, we use “vmin” and “vmax” to force the colorbar to start at 0 and end at 1, irrespective of the actual data range.

To see more information on any function in a Jupyter notebook, use the following notation: (you need to convert the cell from Raw to Code to run it) plt.scatter?

1.4 Figure formats and structures

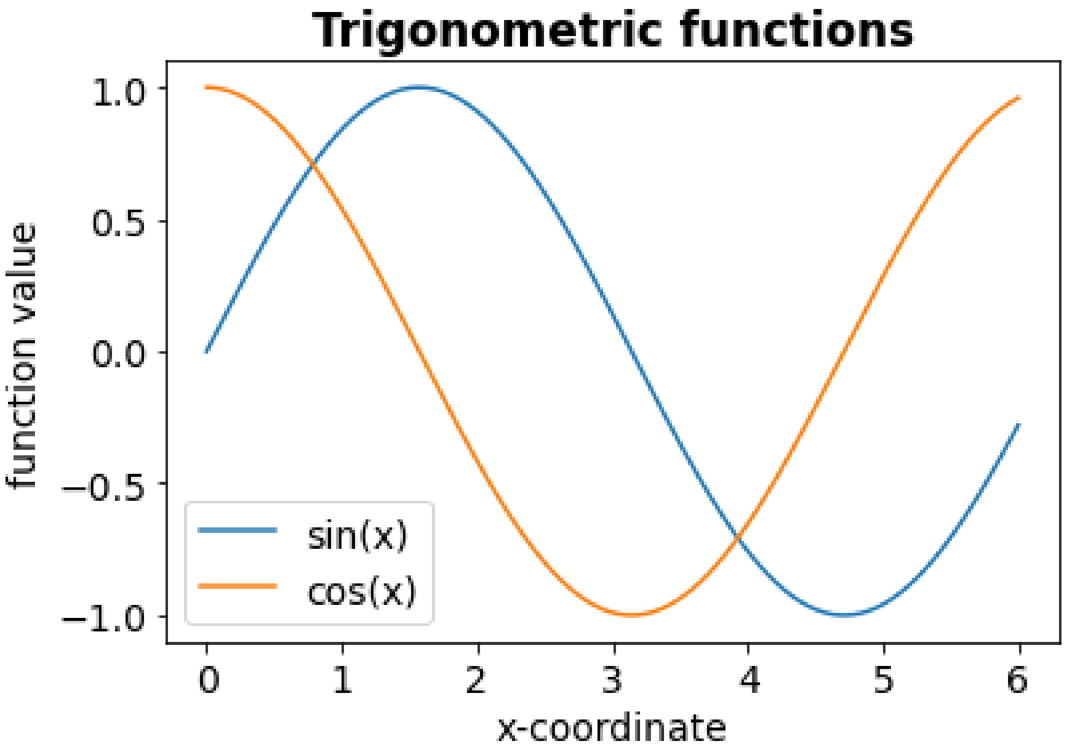
1.4.1 Labels

You should always label the data and your axes. I usually start my plots by giving them a title and labeling the axis. This way, I always know what I want to display and I don’t forget to label them.

[11]:
```
plt.title("Trigonometric functions", fontweight="bold")
plt.xlabel("x-coordinate")
plt.ylabel("function value")

plt.plot(x, np.sin(x), label="sin(x)")
plt.plot(x, np.cos(x), label="cos(x)")
plt.legend()  # you can give the location with loc="upper right"
```

[11]:
```
<matplotlib.legend.Legend at 0x7f288042f2d0>
```



1.4.2 Subplots

This is how you can put several plots into one figure:

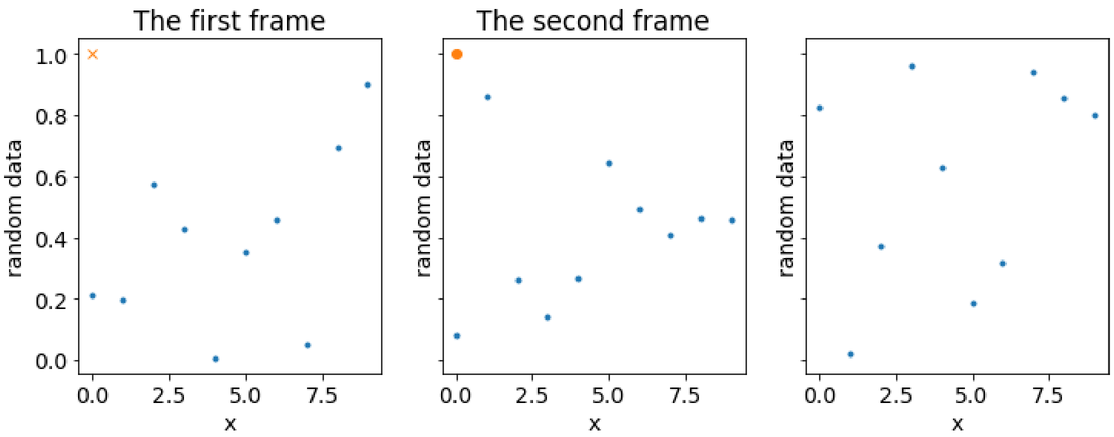
[12]:
```python
fig, axs = plt.subplots(ncols=3, figsize=(12, 4), sharey=True)

# We fill every plot with random data
for ax in axs:
    ax.plot(x_r, np.random.random(x_r.size), ".")
    ax.set_xlabel("x")
    ax.set_ylabel("random data")

# We make some specific adjustments to some of the plots
ax = axs[0]
ax.set_title("The first frame")
ax.plot([0], [1], "x")

ax = axs[1]
ax.set_title("The second frame")
ax.plot([0], [1], "o")
```

[12]:
```
[<matplotlib.lines.Line2D at 0x7f287f3fff50>]
```



Have a look at the following link if you need to have a single colorbar for several subplots: https://stackoverflow.com/questions/13784201/matplotlib-2-subplots-1-colorbar

1.4.3 Displaying figures

The following two lines are not needed in notebooks, but if you use Python in a script.

Use the following line to create a new figure, that means, a new window for the plots. Otherwise, all plot-commands will end up in the same figure. plt.figure() Use the following line to show the figure, that means, to open the window or to update it. plt.show()

1.4.4 Saving and exporting figures

To save your figure as an image file, use the following line directly after you have completed your plot (with labels, colorbars etc.). I usually put it directly before plt.show().

Saving figures as a PDF is usually good, because text is saved as a copyable text and graphics are saved as vector graphics, so you can zoom in without getting a blury image. Only for pcolormesh-plots, it is often better to save them as a PNG. plt.savefig(”filename.pdf”)

1.4.5 Backends

If you use Python in a script and want to add more functionality to your plots, have a look at different backends. They can simplify your life (at least the part of it that deals with the creation of nice plots).

https://matplotlib.org/stable/api/matplotlib_configuration_api.html

https://matplotlib.org/stable/tutorials/introductory/usage.html#backends import matplotlib matplotlib.use(”Qt5Agg”) from matplotlib import pyplot as plt Make sure to put “matplotlib.use” before the import of pyplot.

1.5 Interpolation

You can use NumPy or SciPy for interpolation, depending on the functionality you need. As usualy, the basic functionality is in NumPy, the more advanced methods can be found in SciPy.

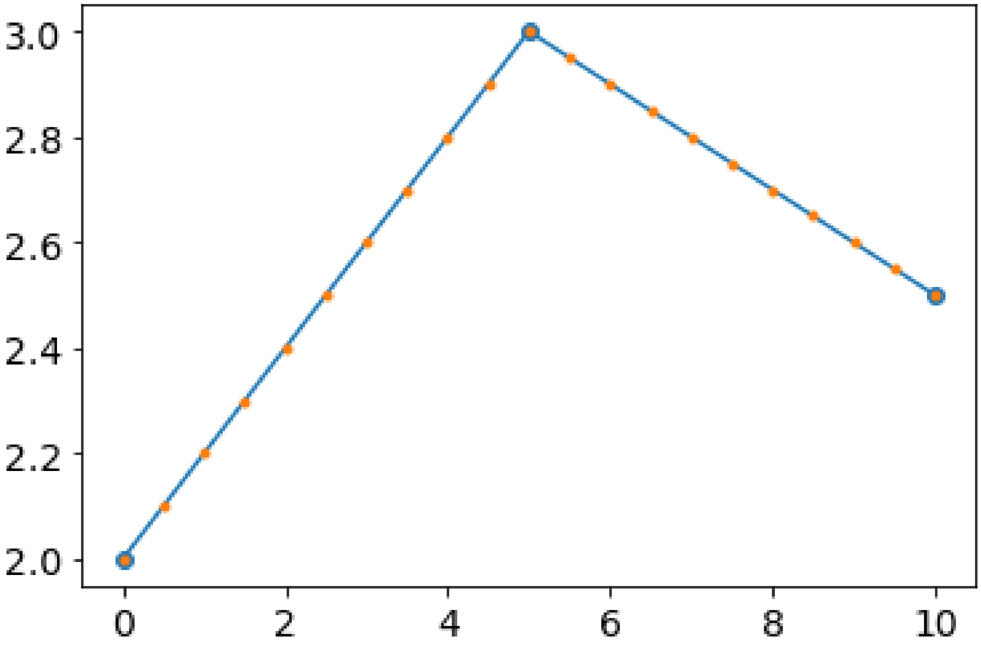
[13]:
```
# Create some simple data (these are lists, not arrays)
x_3points = [0, 5, 10]
y_3points = [2, 3, 2.5]

# Show it in a line plot with dots at the data points
plt.plot(x_3points, y_3points, "o-")

# Interpolate to 21 points between and including the end-points
x_interp = np.linspace(min(x_3points), max(x_3points), 21)
y_interp = np.interp(x_interp, x_3points, y_3points)

# Compare with the original data
plt.plot(x_interp, y_interp, ".")
```

[13]:
```
[<matplotlib.lines.Line2D at 0x7f287f329c10>]
```



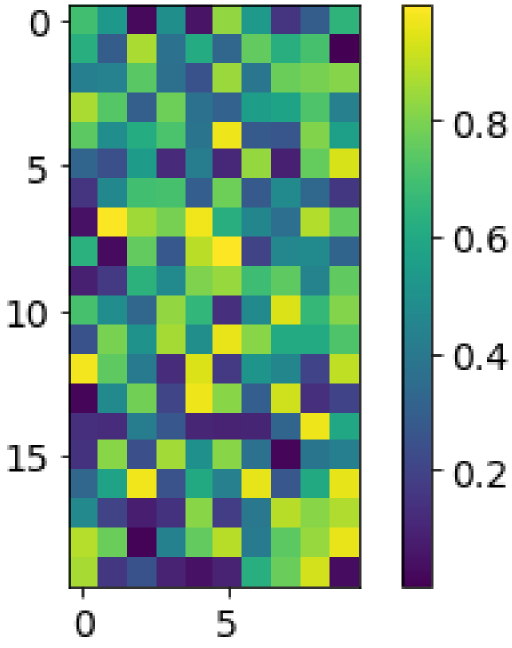
1.6 Plots of 3D data

1.6.1 The imshow function

This is the simplest way to look at a 2D array.

[14]:
```
plt.imshow(Z)
plt.colorbar()
```

[14]:
```
<matplotlib.colorbar.Colorbar at 0x7f287f2d1410>
```



1.6.2 Pseudo-colour plots

The function `pcolormesh` is very useful to show model results. However, its usage is a bit tricky, because we need to provide it with x,y,z data of different sizes. The x- and y-coordinates must be the corner points of the cells that are filled with the z-data, so x and y must be 1 point larger than z.

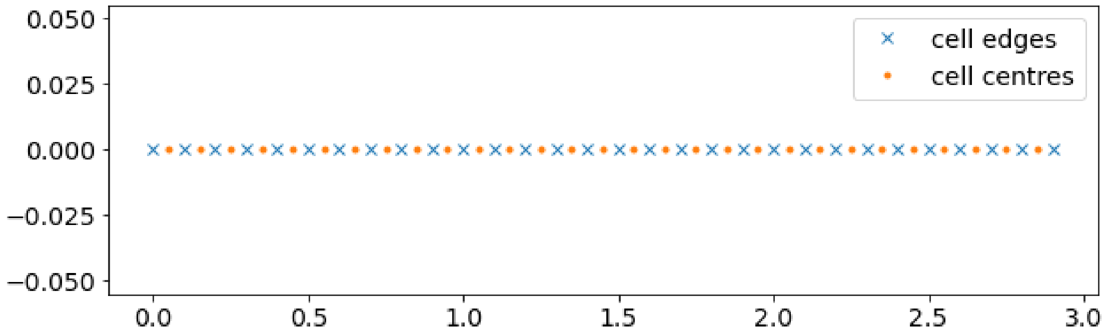
[15]:

```
# Define cell edges
x_e = np.arange(0, 3, 0.1)
y_e = np.arange(0, 6, 0.1)
# Calculate the cell centres
x_c = x_e[:-1] + np.diff(x_e) / 2
y_c = y_e[:-1] + np.diff(y_e) / 2
# Make a 2D grid of the cell centres
X_C, Y_C = np.meshgrid(x_c, y_c)
# Calculate some nice function on the grid
Z_data = np.sin(X_C) * np.cos(Y_C)

# Make a visualization of the x-coordinates
plt.figure(figsize=(10, 3))
plt.plot(x_e, 0*x_e, "x", label="cell edges")
plt.plot(x_c, 0*x_c, ".", label="cell centres")
plt.legend()
```

[15]:

```
<matplotlib.legend.Legend at 0x7f287f245910>
```

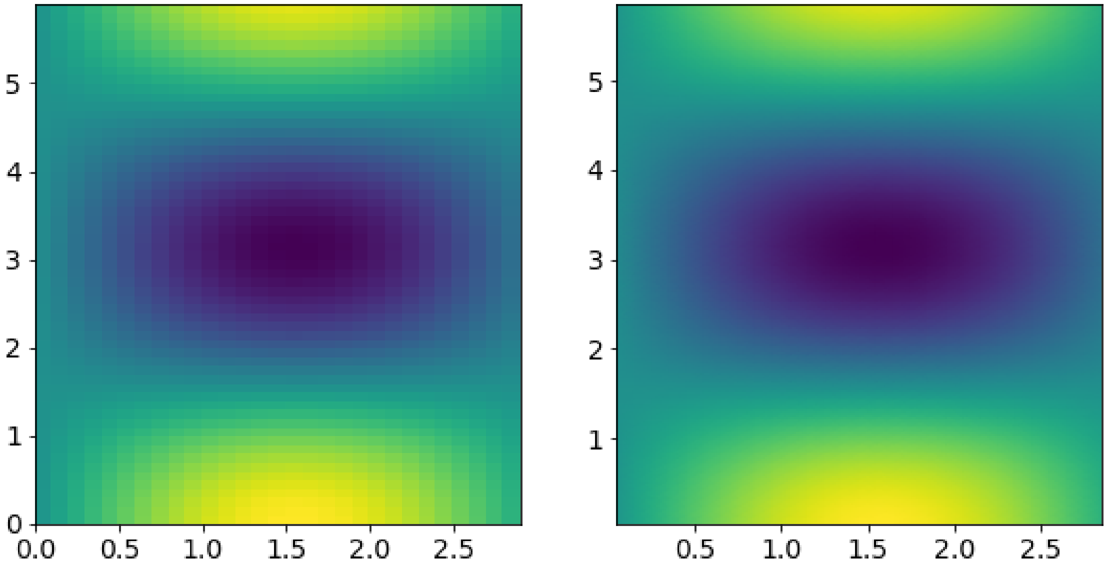


```
[16]: fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

      ax = axs[0]
      ax.pcolormesh(x_e, y_e, Z_data)

      ax = axs[1]
      ax.pcolormesh(x_c, y_c, Z_data, shading="gouraud")
```

```
[16]: <matplotlib.collections.QuadMesh at 0x7f287f18f150>
```



The second subplot provides only the cell centres and uses an interpolation in between. Here we don't see the "model cells" anymore, but this might not make a big difference if you have a high resolution. The second way can be easier to implement if you have only the coordinates of the cell centres, but for model outputs I would generally advice to make the plot with visible cells (first subplot).

1.6.3 Contour plots

[17]:

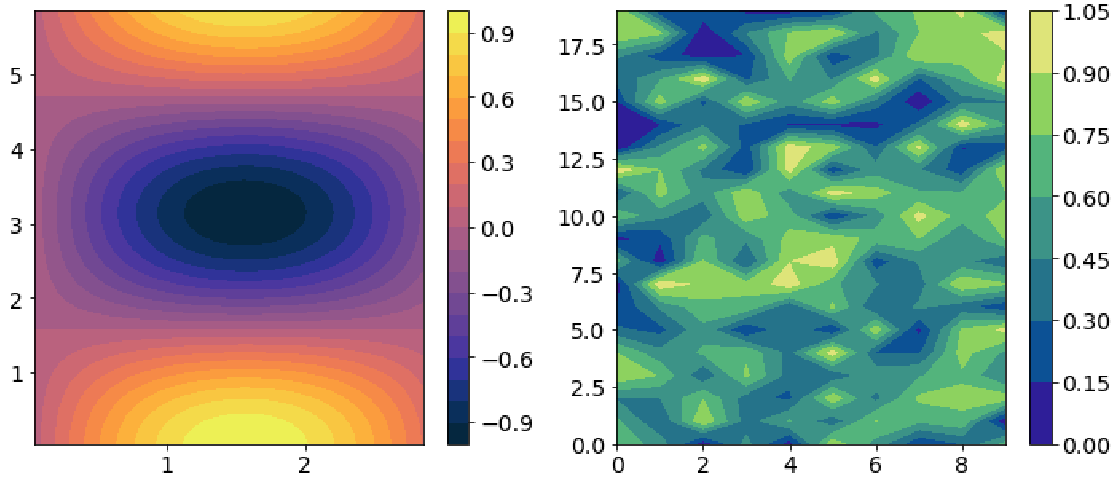
```python
fig, axs = plt.subplots(ncols=2, figsize=(12, 5))

ax = axs[0]
im = ax.contourf(
    X_C, Y_C, Z_data,
    levels=20,
    cmap=cmocean.cm.thermal,
)
fig.colorbar(im, ax=ax)

ax = axs[1]
im = ax.contourf(X, Y, Z, cmap=cmocean.cm.haline)
fig.colorbar(im, ax=ax)
```

[17]:

```
<matplotlib.colorbar.Colorbar at 0x7f287d7c2d10>
```



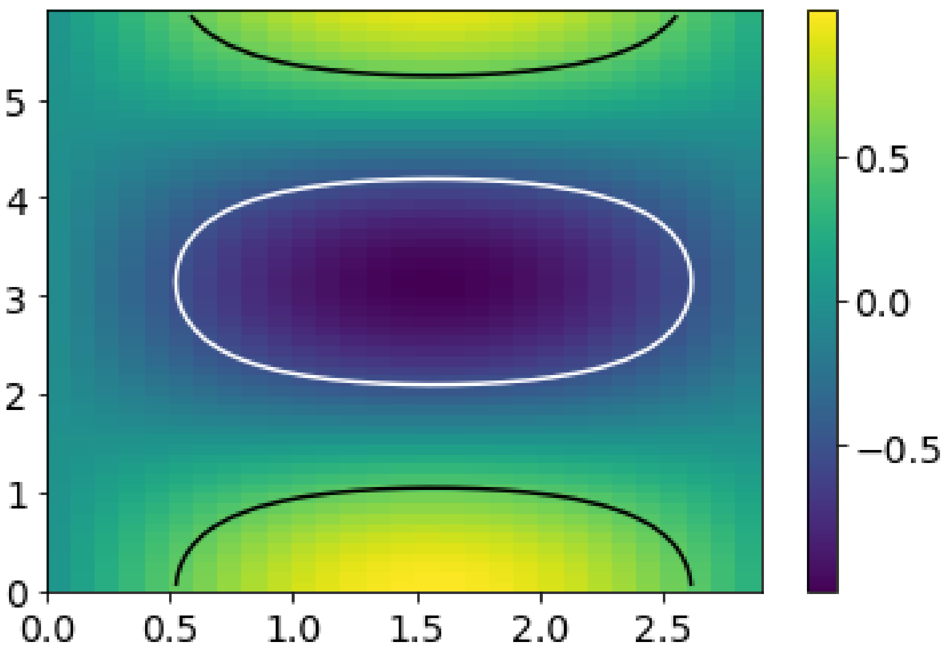
1.6.4 Combinations

[18]:

```python
plt.pcolormesh(x_e, y_e, Z_data)
plt.colorbar()
plt.contour(X_C, Y_C, Z_data, levels=[-0.5, 0.5], cmap="Greys")
```

[18]:

```
<matplotlib.contour.QuadContourSet at 0x7f287d6c15d0>
```


5
4
3
2
1
0
0.0
0.5
1.0
1.5
2.0
2.5
0.5
0.0
−0.5