2023-07-09_PracticalRemarks_01_Images

July 11, 2023

[1]:
```
import numpy as np
import scipy
import scipy.io as sciio
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')

prop_cycle = plt.rcParams['axes.prop_cycle']
colors = prop_cycle.by_key()['color']

%matplotlib inline
```

1 Displaying images: some practical remarks

- we have already used images quite frequently in this lecture, in particular: color spaces and color maps
- now we discuss a few other technical details

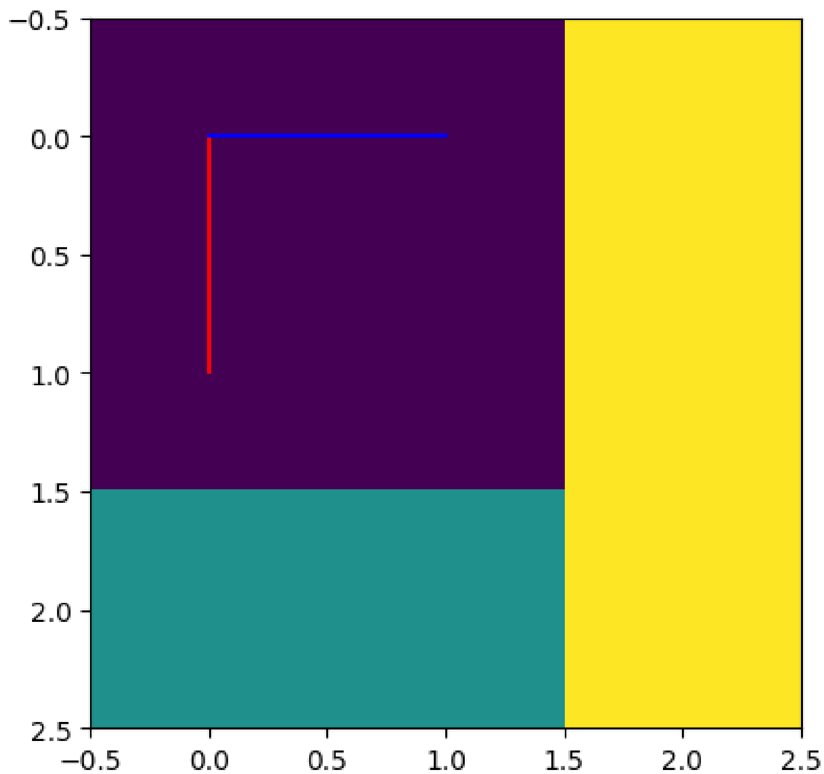
1.1 Axes and alignment

[2]:
```
# default order for pixels in an image:
# rows from top to bottom, columns from left to right
# like denoting entries in a matrix

# however: not compatible with standard convention for Cartesian coordinates
```

[3]:
```
img=np.array([\
        [0.0,0.0,1.0],\
        [0.0,0.0,1.0],\
```

```
        [0.5,0.5,1.0],\
        ],dtype=np.double)
```

[6]:
```
# default conventions in matplotlib: center of pixels at integer coordinates
# rows become y-coordinate, oriented form top to bottom
# columns become x-coordinate

plt.imshow(img)
# show lines on top of this
plt.plot([0,0],[0,1],c="r") # (0,0) to (0,1): vertical
plt.plot([0,1],[0,0],c="b") # (0,0) to (1,0): horizontal
plt.show()
```



now create a more complex example

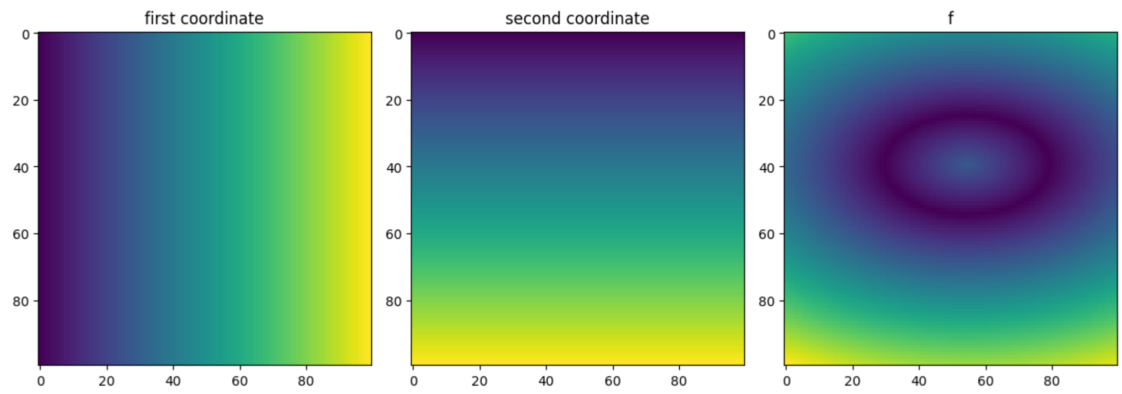
- goal: visualize image and other plot objects in a joint, consistent coordinate system
- example task:
- visualize ellipse, centered at (p,q)=(0.1,-0.2), with semi axes w=0.5, h=0.3
- and the function

$$f:(x,y)\mapsto \left|\left((x-p)/w\right)^2+\left((y-q)/h\right)^2-1\right|$$

  based on that ellipse in the background as image, over the domain $[-1,1]^2$

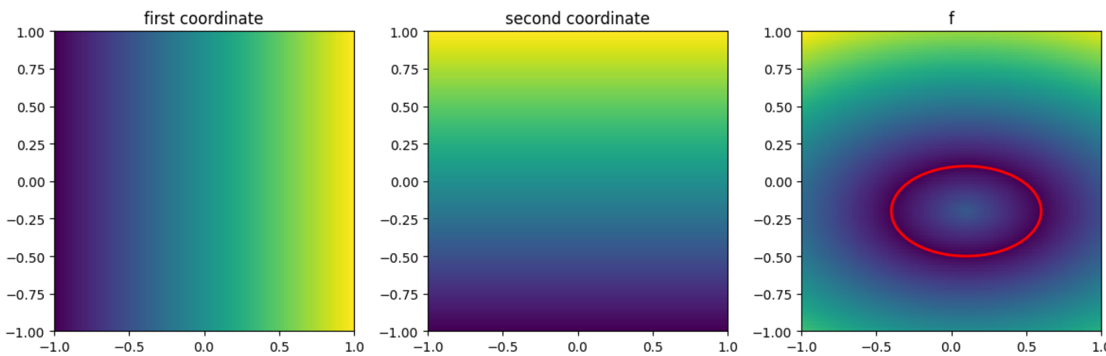
[7]:
```
# many ways to implement this, np.meshgrid is one choice
nPts1d=100
x = np.linspace(-1,1,num=nPts1d)
y = x
nPts=nPts1d**2
X, Y = np.meshgrid(x, y)

img=np.abs((((X-0.1)/0.5)**2+((Y+0.2)/0.3)**2)**0.5-1)
```

[8]:
```
# naive visualization
fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,3,1)
plt.title("first coordinate")
plt.imshow(X)
fig.add_subplot(1,3,2)
plt.title("second coordinate")
plt.imshow(Y)
fig.add_subplot(1,3,3)
plt.title("f")
plt.imshow(img)

plt.tight_layout()
plt.show()
# axis scaling is wrong
# y-axis is flipped
```



[9]:
```
# now squeeze image into proper region of coordinate system, flip y-axis
# extent: l,r,b,t
# origin="lower" flips orientation of Y-axis


fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,3,1)
```

```
plt.title("first coordinate")
plt.imshow(X,extent=[-1,1,-1,1],origin="lower")
fig.add_subplot(1,3,2)
plt.title("second coordinate")
plt.imshow(Y,extent=[-1,1,-1,1],origin="lower")
fig.add_subplot(1,3,3)
plt.title("f")
plt.imshow(img,extent=[-1,1,-1,1],origin="lower")
# add ellipse
ax=plt.gca()
e=matplotlib.patches.Ellipse([0.1,-0.2],width=1.,height=0.
 ↪6,fill=False,color="r",lw=2)
ax.add_artist(e)

plt.tight_layout()
plt.show()
```



- depending on conventions and implementation: sometimes also a transpose is necessary
- bottom line: whenever the orientation / alignment of an image is imporant, be sure to get it right
- verify with simple test images

1.2 Interpolation mode: upsampling of small images

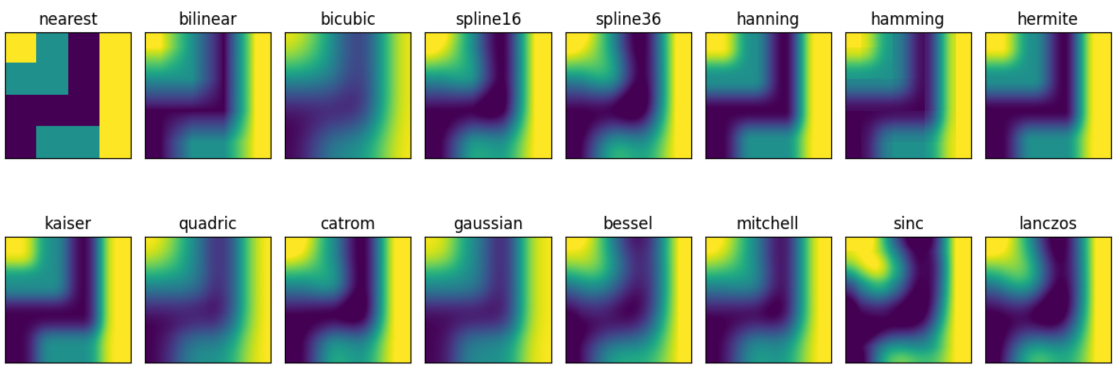
1.2.1 Small example

[10]:
```
# this is essentially taken from:
# https://matplotlib.org/stable/gallery/images_contours_and_fields/
 ↪interpolation_methods.html
```

[11]:
```
img=np.array([\
        [1.0,0.5,0.0,1.0],\
        [0.5,0.5,0.0,1.0],\
```

```
        [0.0,0.0,0.0,1.0],\
        [0.0,0.5,0.5,1.0],\
        ],dtype=np.double)
```

[12]:
```
methods = ['nearest', 'bilinear', 'bicubic', 'spline16',
           'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric',
           'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos']

fig, axs = plt.subplots(nrows=2, ncols=8, figsize=(12, 5),
                        subplot_kw={'xticks': [], 'yticks': []})

for ax, interp_method in zip(axs.flat, methods):
    ax.imshow(img, interpolation=interp_method, cmap='viridis')
    ax.set_title(str(interp_method))

plt.tight_layout()
plt.show()
```



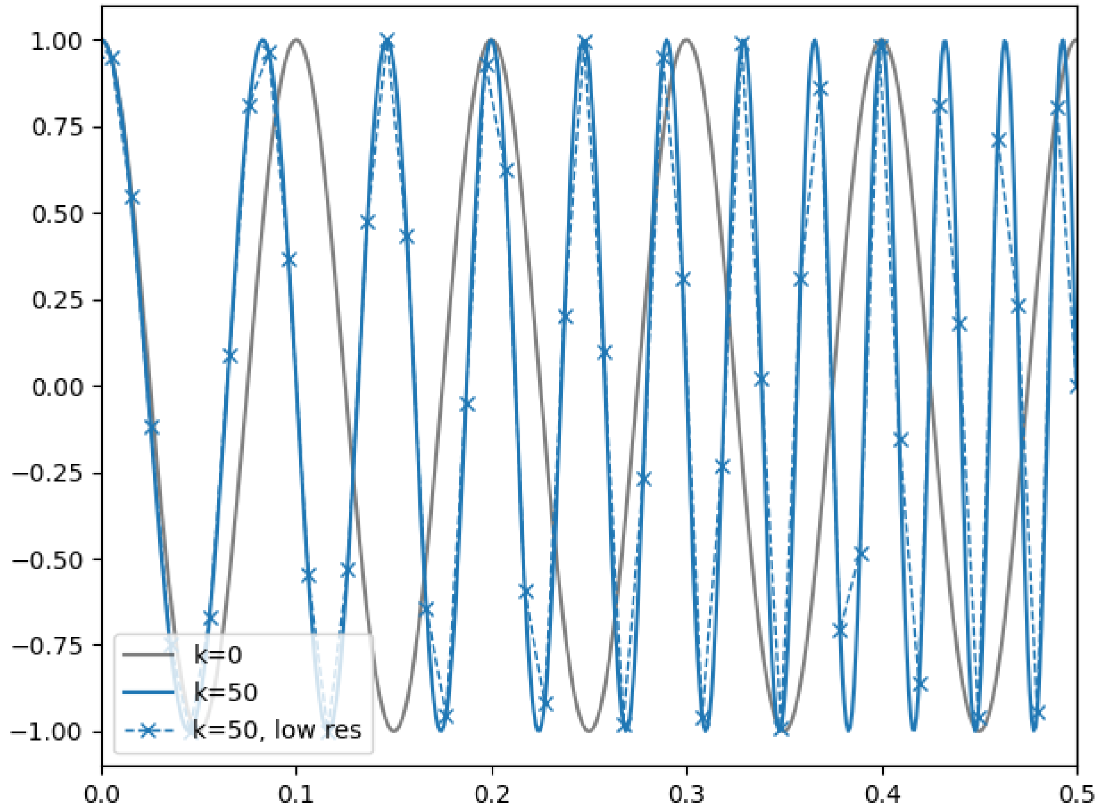
1.2.2 More complex example

[13]:
```
# based on
# https://matplotlib.org/stable/gallery/images_contours_and_fields/
 ↪image_antialiasing.html
# but without the downsampling
```

[14]:
```
nPts1d=100
x=np.linspace(-0.5,0.5,num=nPts1d)
y=x

X, Y = np.meshgrid(x, y)
R = np.sqrt(X**2 + Y**2)
f0 = 10
```

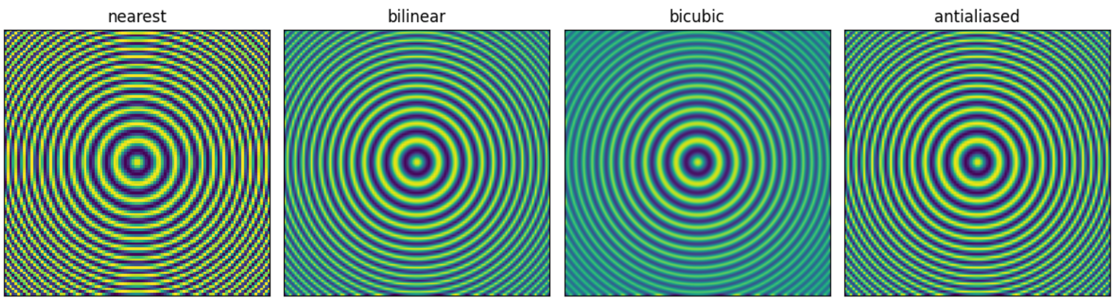
```
k = 50
def f(r,f0,k):
    return np.cos(2*np.pi * (f0 * r + 0.5*k * r**2))
img = f(R,f0,k)
```

[15]:
```
z=np.linspace(-0.5,0.5,num=1000)
plt.plot(z,f(np.abs(z),f0,0),label="k=0",c="k",alpha=0.5)
plt.plot(z,f(np.abs(z),f0,k),label="k={:d}".format(k),c=colors[0])
plt.plot(x,f(np.abs(x),f0,k),label="k={:d}, low res".
 ↪format(k),marker="x",lw=1,ls="dashed")
plt.xlim([0,0.5])
plt.legend(loc=3)
plt.tight_layout()
plt.show()
```



[16]:
```
methods = ['nearest', 'bilinear', 'bicubic', 'antialiased']

fig, axs = plt.subplots(nrows=1, ncols=4, figsize=(12,4),
                        subplot_kw={'xticks': [], 'yticks': []})

for ax, interp_method in zip(axs.flat, methods):
```

```python
    ax.imshow(img, interpolation=interp_method, cmap='viridis')
    ax.set_title(str(interp_method))

plt.tight_layout()
plt.show()
```



in practice: usually only need nearest, bilinear and bicubic * nearest is most transparent * bi-linear is simplest continuous choice * bi-cubic is simplest differentiable choice

[ ]:

```python

```