2023-05-08_Example-2-ColorSchemes

May 8, 2023

[30]:
```
import numpy as np
import scipy
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
```

[31]:
```
def scramble(dat):
    """scramble a linear array to test visual uniformness of color schemes,
    assume dat has shape (n,) with n even
    here just mix up adjacent values"""
    result=np.zeros_like(dat)
    n=len(dat)
    result[::2]=dat[1::2]
    result[1::2]=dat[::2]
    return result



def createTestImg(dat,k,colfun):
    """create a small (k,n)-image where colfun is applied to dat along the n␣
 ↪axis"""
    n=len(dat)
    result=np.zeros((n,k),dtype=np.double)
    result[...]=dat.reshape((n,1))
    result=result.transpose()
    result=colfun(result)
```

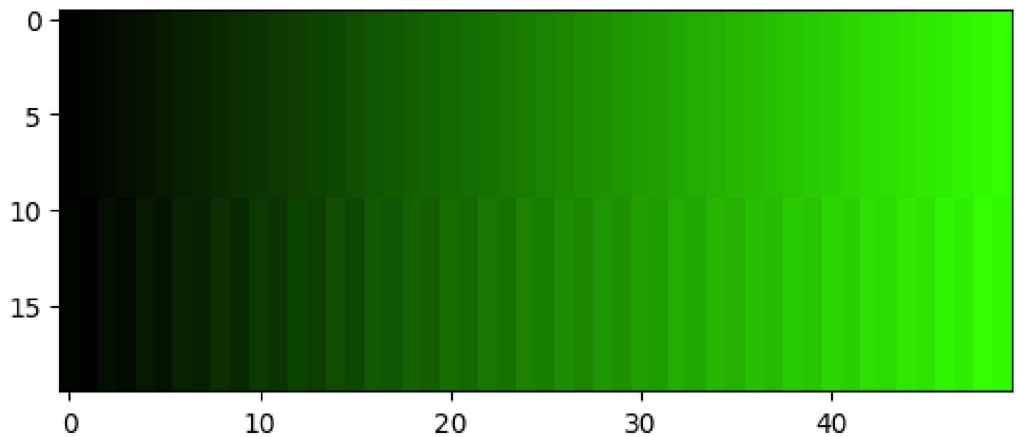
```
    return result

n=50
xrange1=np.arange(n)/(n-1)
xrange2=scramble(xrange1)

k=10

# create a pair of test images of a color scale, one with clean and one with␣
↪scambled sequence
# to shorten the rest of the code / signature of the function, use global␣
↪variables
def createTestImgPair(colfun):
    img1=createTestImg(xrange1,k,colfun)
    img2=createTestImg(xrange2,k,colfun)
    img=np.concatenate((img1,img2))
    return img
```

1 Create simple lightness variation maps

1.1 HSV: value variation for fixed hue

[32]:
```
def ex_colfun_1(dat,hue):
    """apply simple HSV-value-variation to a 2d array with values in [0,1]"""
    res=dat.shape
    imgHSV=np.zeros(res+(3,),dtype=np.double)
    imgHSV[:,:,0]=hue
    imgHSV[:,:,1]=1.
    imgHSV[:,:,2]=dat
    img=matplotlib.colors.hsv_to_rgb(imgHSV)
    return img
```

[33]:
```
fun=lambda dat : ex_colfun_1(dat,.3)
#img=createTestImg(xrange1,k,fun)
img=createTestImgPair(fun)
plt.imshow(img)
plt.show()
```

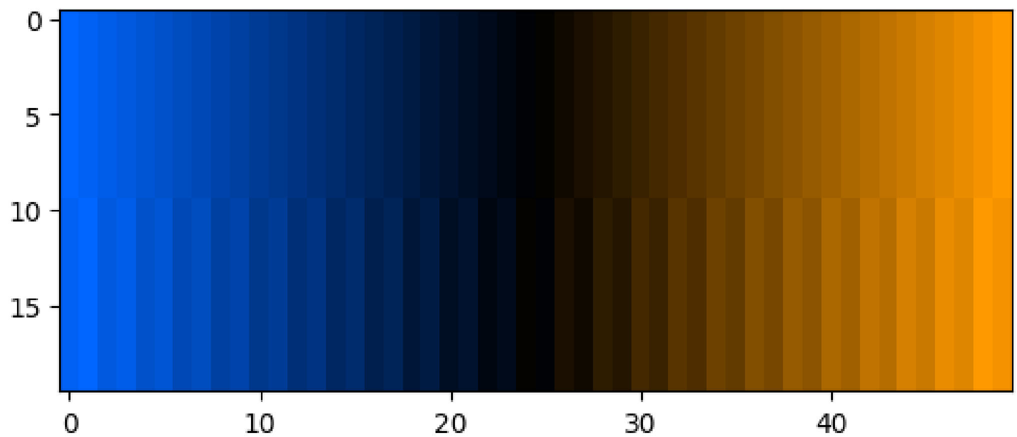


[34]:
```
# low end values are difficult to distinguish
```

1.1.1 Signed example

[35]:
```
def ex_colfun_2(dat,hue):
    """apply simple HSV-value-variation to a 2d array with values in [-1,1],␣
 ↪signed,
    negative values are mapped to opposite hue"""
    res=dat.shape
    imgHSV=np.zeros(res+(3,),dtype=np.double)
    imgHSV[:,:,0]=hue
    imgHSV[:,:,1]=1.
    imgHSV[:,:,2]=np.abs(dat)
    # now flip hue where dat is negative
    posNeg=np.where(dat<0)
    imgHSV[posNeg[0],posNeg[1],0]+=0.5
    imgHSV[:,:,0]=np.mod(imgHSV[:,:,0],1.)
    img=matplotlib.colors.hsv_to_rgb(imgHSV)
    return img
```

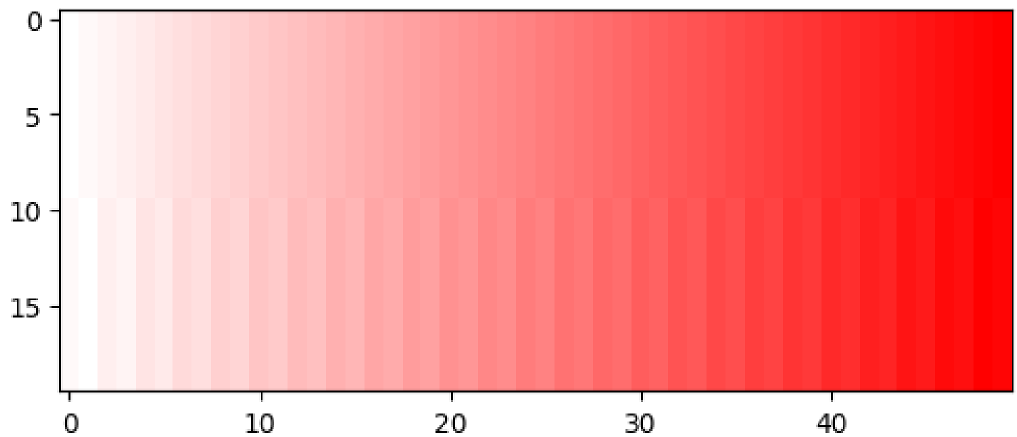
[36]:
```
fun=lambda dat : ex_colfun_2(2*dat-1,.1)
#img=createTestImg(xrange1,k,fun)
img=createTestImgPair(fun)
plt.imshow(img)
plt.show()
```



1.2 HSV: saturation variation for fixed hue

[37]:
```
def ex_colfun_3(dat,hue,s):
    """apply simple HSV-saturation-variation to a 2d array with values in␣
↪[0,1]"""
    res=dat.shape
    imgHSV=np.zeros(res+(3,),dtype=np.double)
    imgHSV[:,:,0]=hue
    imgHSV[:,:,1]=dat
    imgHSV[:,:,2]=s+(1-s)*dat
    img=matplotlib.colors.hsv_to_rgb(imgHSV)
    return img
```

[38]:
```
fun=lambda dat : ex_colfun_3(dat,.0,1.)
#img=createTestImg(xrange1,k,fun)
img=createTestImgPair(fun)
plt.imshow(img)
plt.show()
```

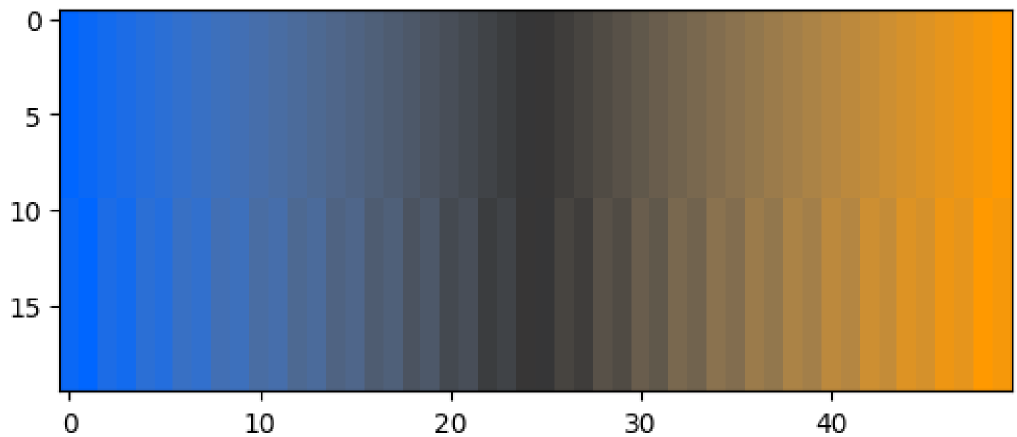


[39]:
```
# much weaker separation of steps at high end
```

1.2.1 Signed example

[40]:
```
def ex_colfun_4(dat,hue,s):
    """apply simple HSV-value/saturation-variation to a 2d array with values in␣
 ↪[-1,1], signed,
    negative values are mapped to opposite hue,
    dont go through black, but through some greyish value. regulate this with␣
 ↪param s."""
    res=dat.shape
    imgHSV=np.zeros(res+(3,),dtype=np.double)
    imgHSV[:,:,0]=hue
    imgHSV[:,:,1]=np.abs(dat)
    imgHSV[:,:,2]=s+(1-s)*np.abs(dat)
    # now flip hue where dat is negative
    posNeg=np.where(dat<0)
    imgHSV[posNeg[0],posNeg[1],0]+=0.5
    imgHSV[:,:,0]=np.mod(imgHSV[:,:,0],1.)
    img=matplotlib.colors.hsv_to_rgb(imgHSV)
    return img
```

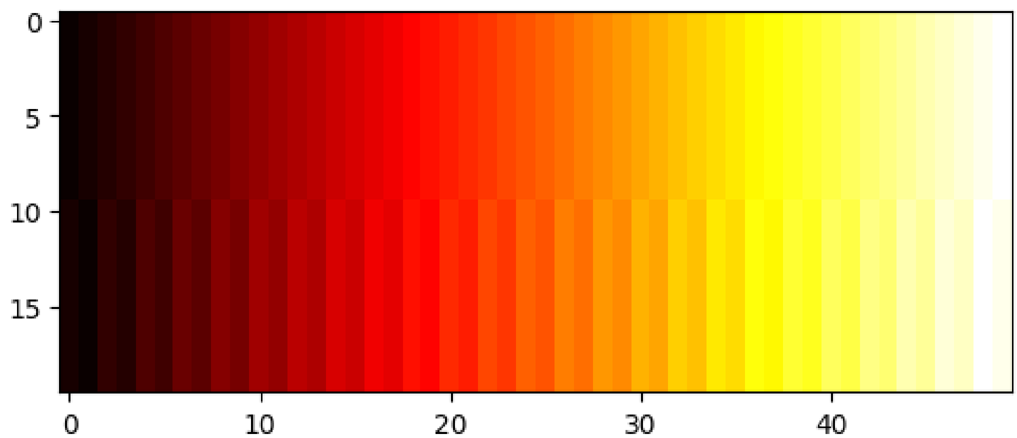
[41]:
```
fun=lambda dat : ex_colfun_4(2*dat-1,.1,0.2)
#img=createTestImg(xrange1,k,fun)
img=createTestImgPair(fun)
plt.imshow(img)
plt.show()
```



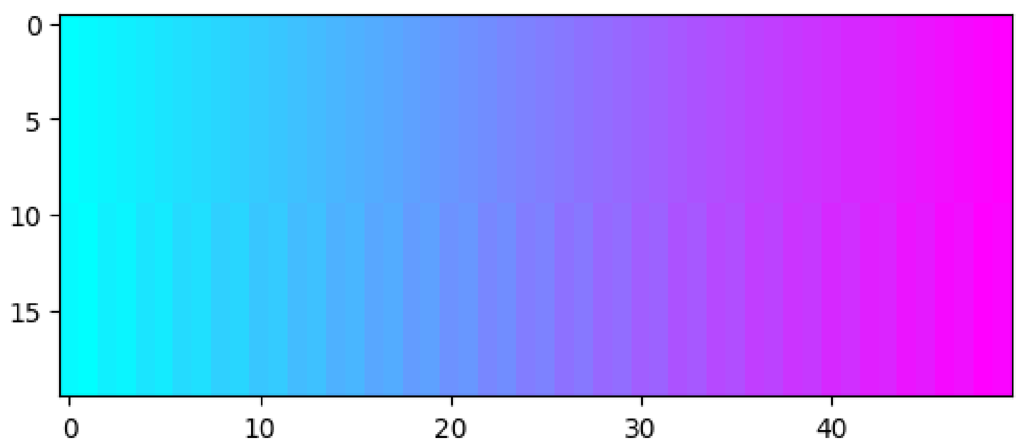
[42]:
```
# this one may look a bit less harsh at zero, but it has perceptually weaker
  transitions
```

2 Perceptual uniformity

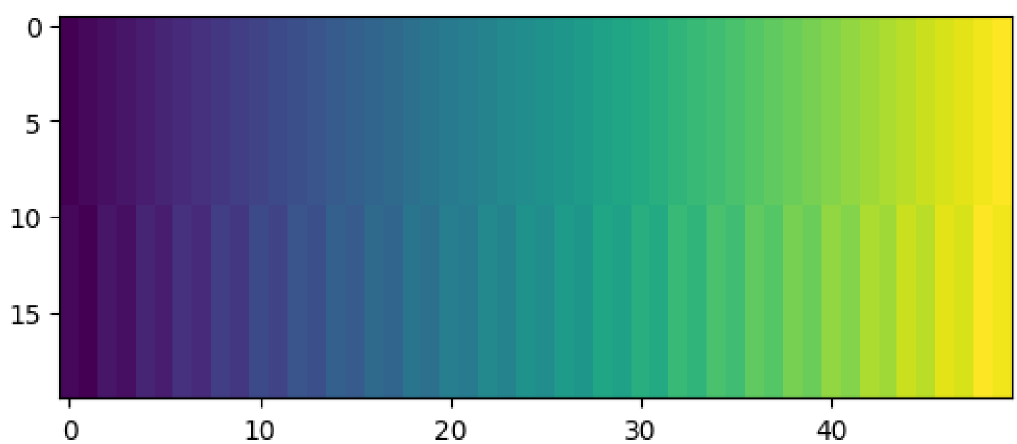
[43]:
```
img=createTestImgPair(cm.hot)
plt.imshow(img)
plt.show()
```



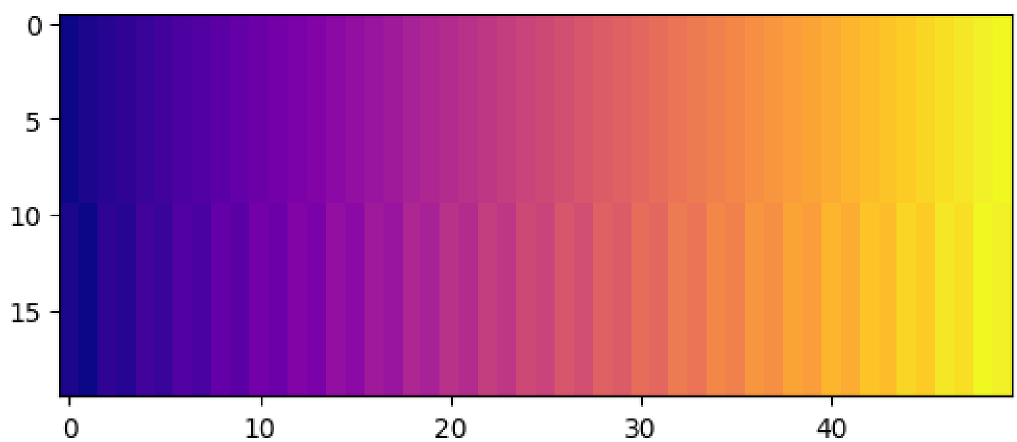
[44]:
```
img=createTestImgPair(cm.cool)
plt.imshow(img)
plt.show()
```



[45]:
```
img=createTestImgPair(cm.viridis)
plt.imshow(img)
plt.show()
```

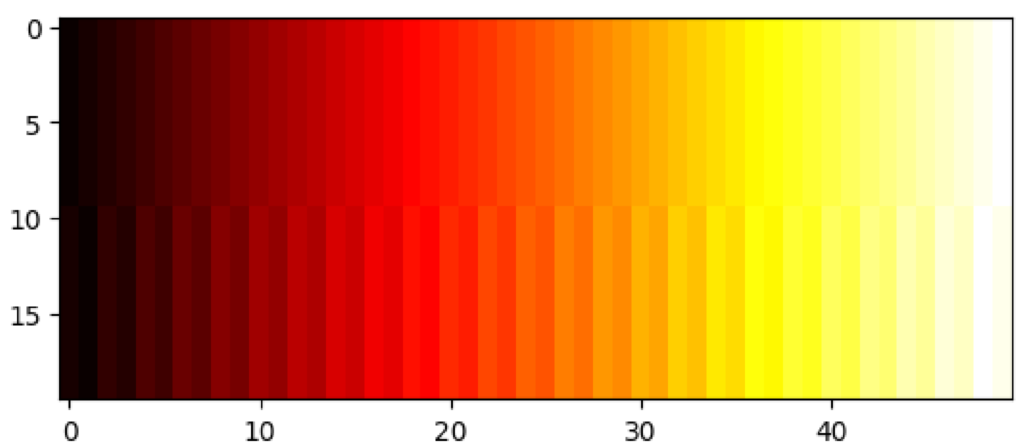


[46]:
```
img=createTestImgPair(cm.plasma)
plt.imshow(img)
plt.show()
```

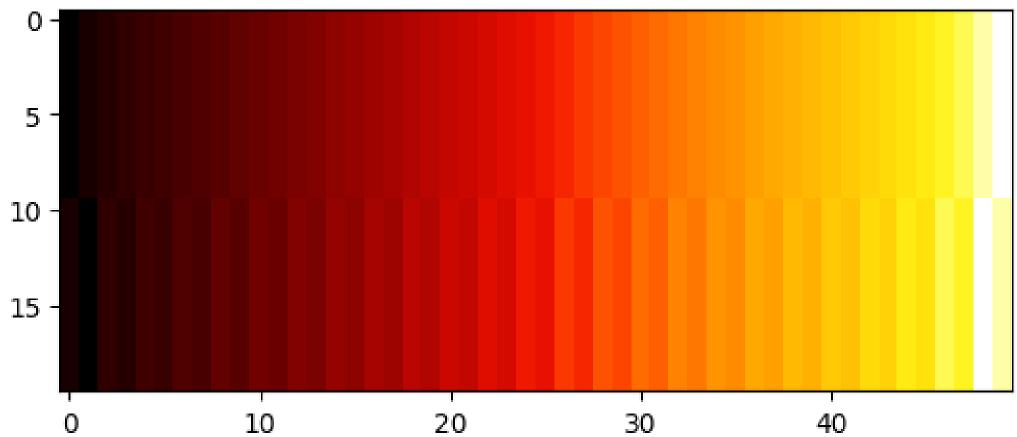


[47]:
```
import colorcet
```

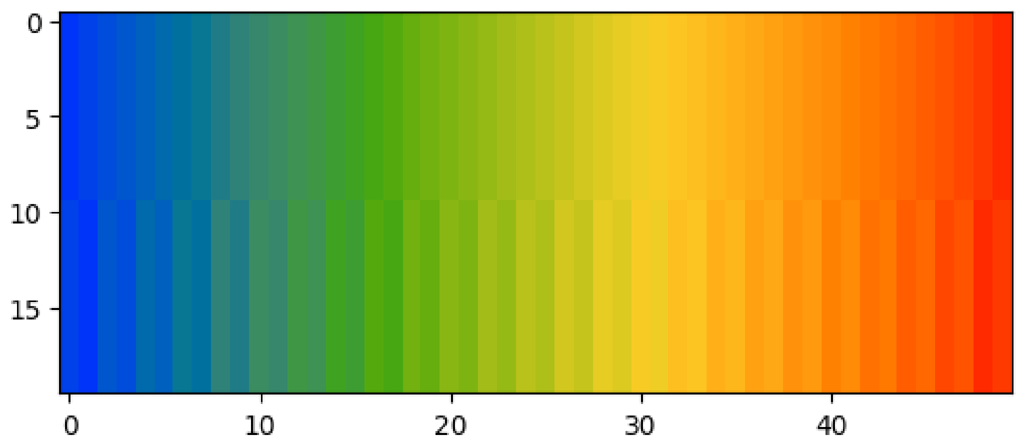
[48]:
```
img=createTestImgPair(cm.hot)
plt.imshow(img)
plt.show()
```



[49]:
```
img=createTestImgPair(colorcet.cm.fire)
plt.imshow(img)
plt.show()
```



[50]:
```
img=createTestImgPair(colorcet.cm.rainbow)
plt.imshow(img)
plt.show()
```

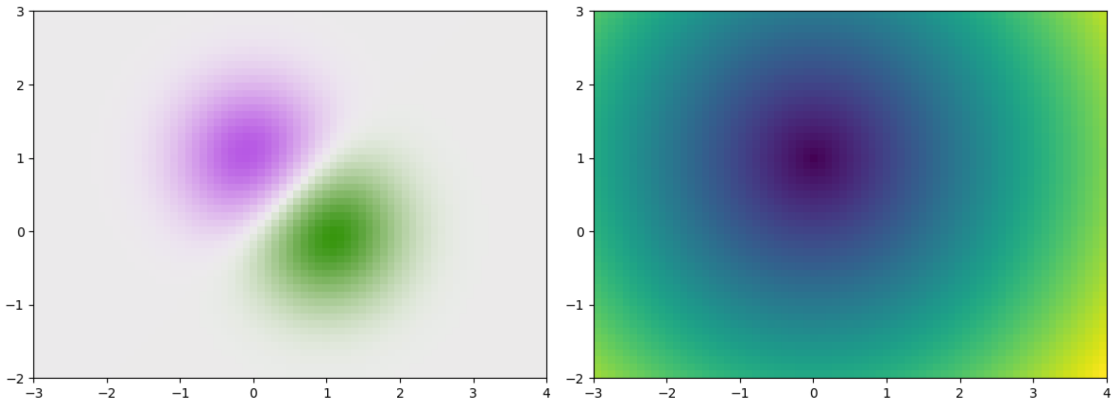


3 Improve perception

[51]:
```
# create test data
# taken from https://matplotlib.org/stable/gallery/images_contours_and_fields/
↪contour_image.html
extent = (-3, 4, -2, 3)
delta=0.1
x = np.arange(-3.0, 4.001, delta)
y = np.arange(-2.0, 3.001, delta)
```

```python
X, Y = np.meshgrid(x, y)
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2

ZB=(X**2+Y**2)**0.5

# crude normalization to [-1,1] (but still in [0,1]):
vmax=np.max(np.abs(Z))
img=(Z+vmax)/(2*vmax)

# crude normalization to [0,1]:
vmax=np.max(ZB)
vmin=np.min(ZB)
imgB=(ZB-vmin)/(vmax-vmin)
```

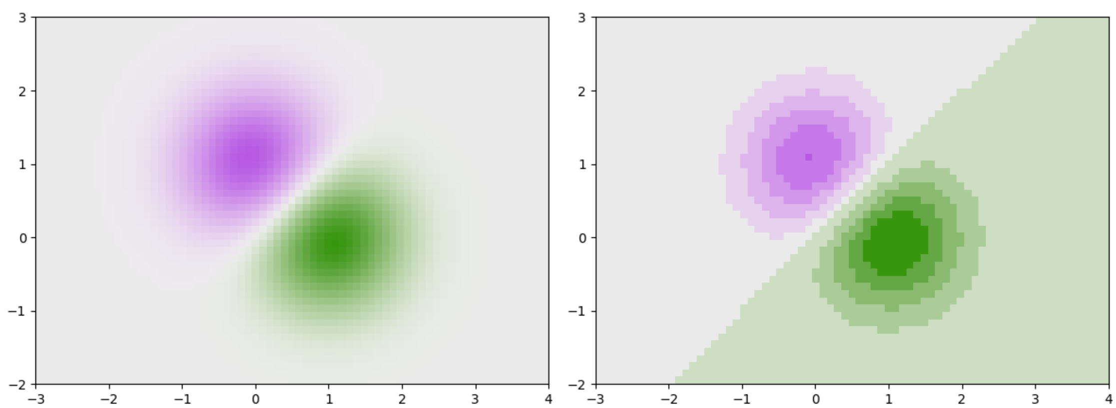
[52]:
```python
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(colorcet.cm.gwv(img),extent=extent)
#plt.imshow(cm.viridis(img),extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(imgB,extent=extent)
plt.tight_layout()
plt.show()
```

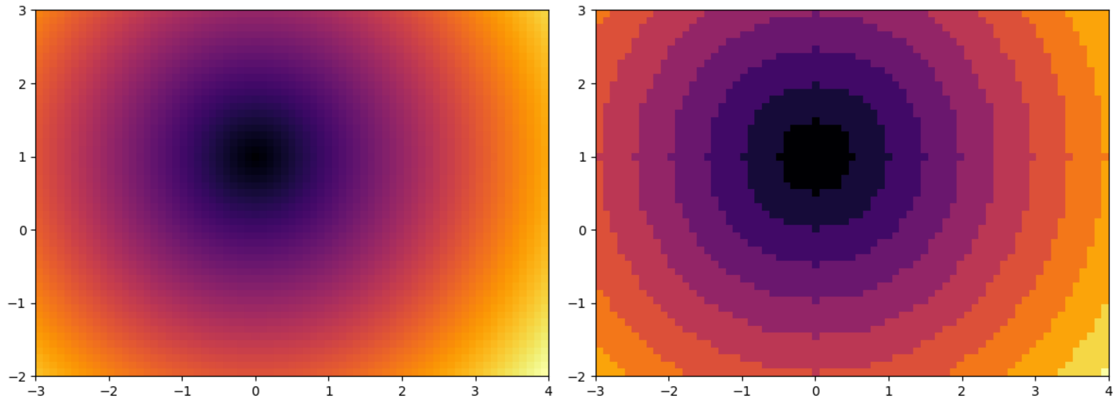


3.1 Discretize values into bins

[53]:
```python
colfun=colorcet.cm.gwv
colfunB=cm.inferno
scale=10
```

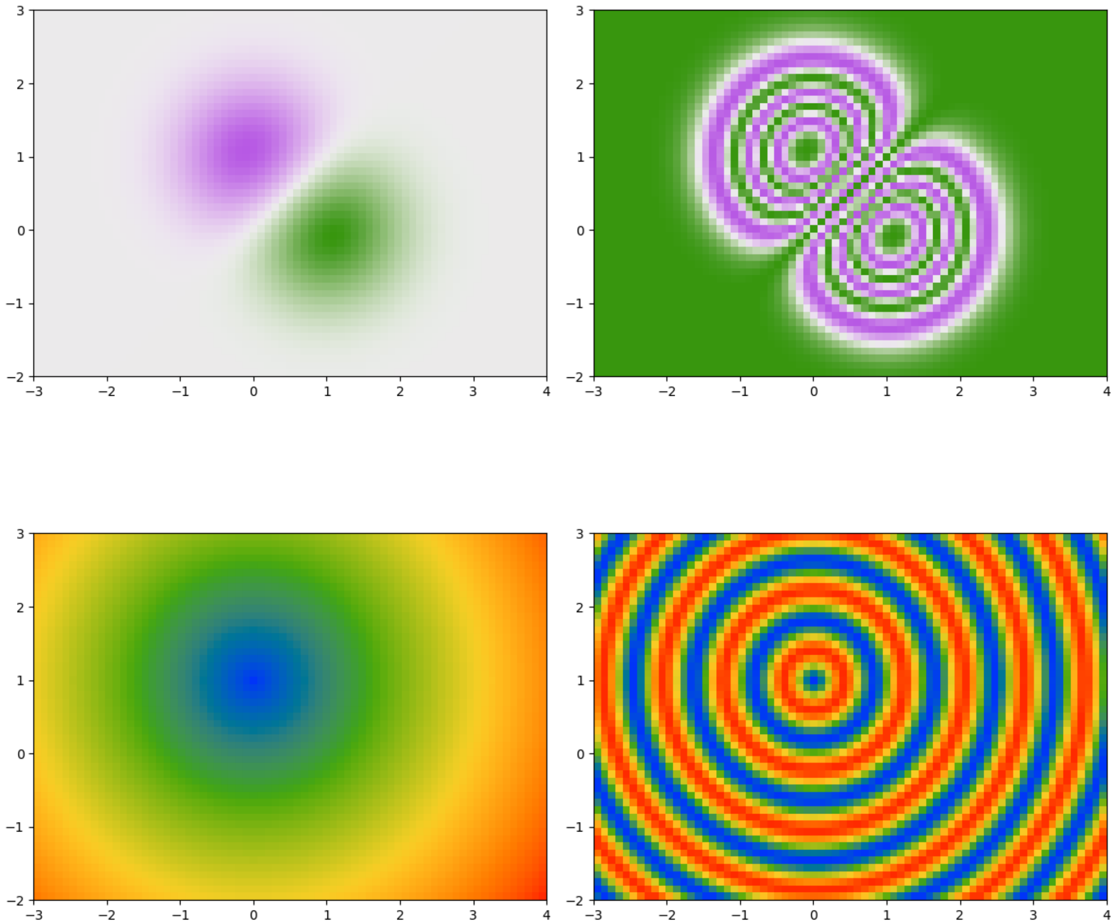
```
img1=colfun(img)
img2=colfun(np.trunc(img*scale)/scale)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img2,extent=extent)

plt.tight_layout()
plt.show()

img1=colfunB(imgB)
img2=colfunB(np.trunc(imgB*scale)/scale)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img2,extent=extent)

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





3.2 High-frequency periodic color scale

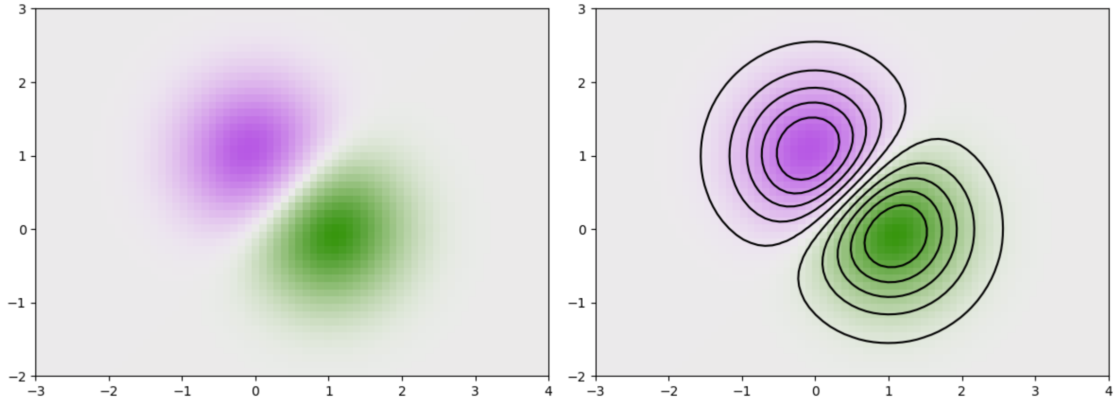
[55]:
```
colfun=colorcet.cm.gwv
colfunB=colorcet.cm.rainbow

img1=colfun(img)
img2=colfun(np.sin(img*2*np.pi*3)**2)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img2,extent=extent)

plt.tight_layout()
plt.show()

img1=colfunB(imgB)
img2=colfunB(np.sin(imgB*2*np.pi*3)**2)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img2,extent=extent)

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

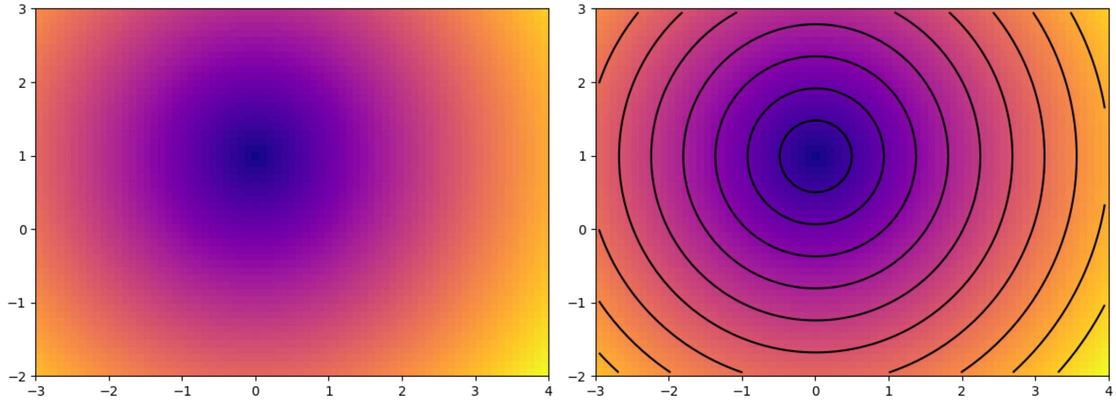


3.3 Add contours

[56]:
```python
colfun=colorcet.cm.gwv
colfunB=cm.plasma
levels=np.linspace(0.1,0.9,num=10)

img1=colfun(img)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img1,extent=extent)
ax=plt.gca()
ax.contour(img, levels, colors='k', origin='image', extent=extent)
plt.tight_layout()
plt.show()
```

```
img1=colfunB(imgB)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img1,extent=extent)
ax=plt.gca()
ax.contour(imgB, levels, colors='k', origin='image', extent=extent)

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



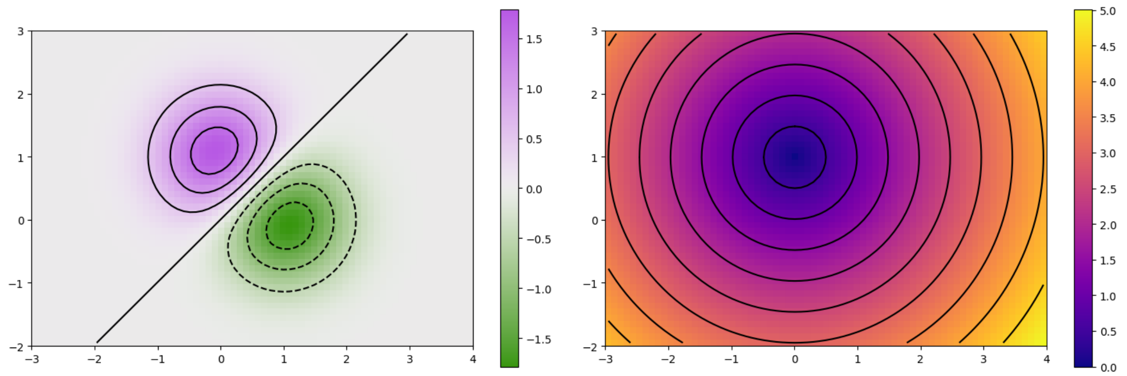


3.4 Plot with proper color bar legend

manual normalization: (what is done above) * unsigned data: scale into [0,1] from minimal to maximal value (or from zero to max) * signed data: scale first into [-1,1] from -max(abs(...)) to +max(abs(...)), then scale this into [0,1]

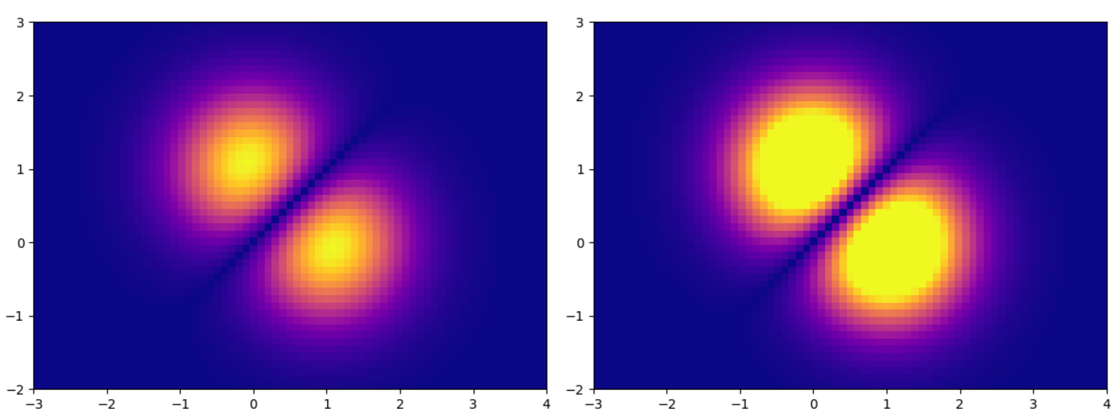
using the matplotlib normalization object:

[27]:
```
# normalization for signed data
norm = cm.colors.Normalize(vmax=abs(Z).max(), vmin=-abs(Z).max())
# normalization for unsigned data
normB = cm.colors.Normalize(vmax=abs(ZB).max(), vmin=0)

colfun=colorcet.cm.gwv
colfunB=cm.plasma
# now choose levels as absolute values, use them for contour plot and ticks in␣
↪colorbar
levelsA=np.linspace(-1.5,1.5,num=7)
levelsB=np.linspace(0,5,num=11)

fig=plt.figure(figsize=(15,5))
fig.add_subplot(1,2,1)
implt=plt.imshow(Z,extent=extent,cmap=colfun,norm=norm)
ax=plt.gca()
ax.contour(Z, levelsA, colors='k', origin='image', extent=extent)
fig.colorbar(implt, ax=ax,ticks=levelsA)

fig.add_subplot(1,2,2)
implt=plt.imshow(ZB,extent=extent,cmap=colfunB,norm=normB)
ax=plt.gca()
ax.contour(ZB, levelsB, colors='k', origin='image', extent=extent)
fig.colorbar(implt, ax=ax,ticks=levelsB)

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



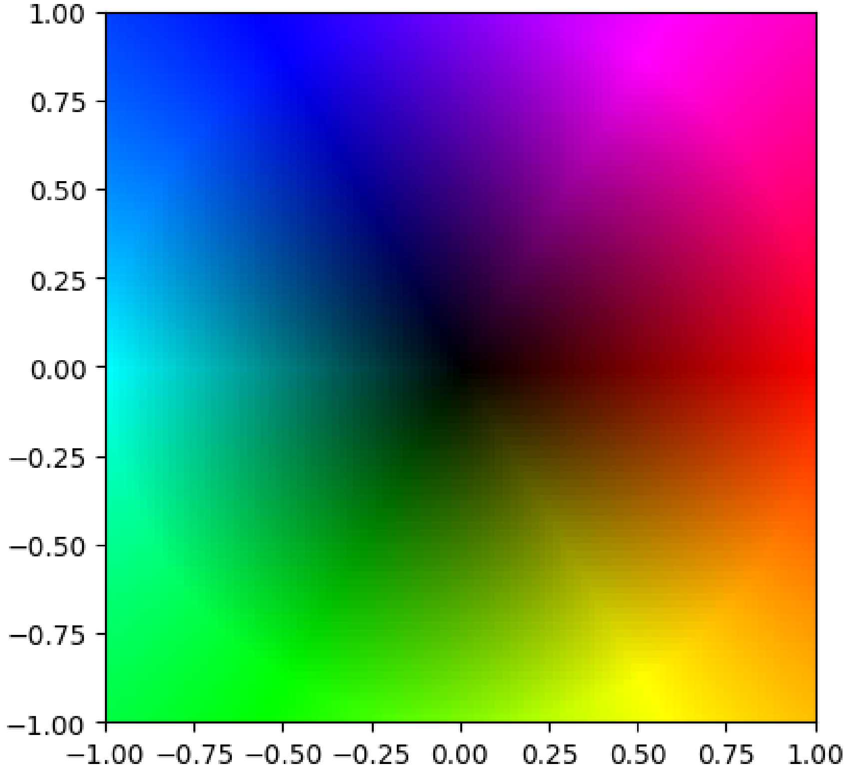
3.5 Choosing the value range

[28]:
```
colfun=cm.plasma

# for simplicity: work on unsigned data, use abs of first example
ZC=np.abs(Z)
vmax=np.max(ZC)

normC=cm.colors.Normalize(vmax=vmax, vmin=0)
if False:
    ## more focus on high end
    normC1=cm.colors.Normalize(vmax=vmax, vmin=0.8*vmax)
if True:
    ## more focus on small values
    normC1=cm.colors.Normalize(vmax=0.6*vmax,vmin=0)
if False:
    ## cut off small and/or large values
    s=0.2
    normC1=cm.colors.Normalize(vmax=(1-s)*vmax, vmin=s*vmax)



fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(ZC,extent=extent,cmap=colfun,norm=normC)

fig.add_subplot(1,2,2)
plt.imshow(ZC,extent=extent,cmap=colfun,norm=normC1)

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



4 Using HSV for encoding a simple map

[29]:
```
extent = (-1, 1, -1, 1)
delta=0.02
x = np.arange(extent[0], extent[1], delta)
y = np.arange(extent[2], extent[3], delta)
X, Y = np.meshgrid(x, y)

# translate X and Y to polar coordinates
# radius
Rad=(X**2+Y**2)**0.5
# angle (arctan2 returns values in [-pi,pi]. transform this to [0,2*pi] via mod)
Phi=np.mod(np.arctan2(Y,X),2*np.pi)

# allocate free space for an HSV image
imgHSV=np.zeros(X.shape+(3,),dtype=np.double)
# set hue to Phi (re-scale by 2*pi)
imgHSV[:,:,0]=Phi/(2*np.pi)
# saturation to full value
imgHSV[:,:,1]=1.
# value given by radius
# scale radius=1 to maximal value=1, replace all larger radii by 1
maxRad=1.
imgHSV[:,:,2]=np.minimum(Rad/maxRad,1.)

# transform to RGB
img=matplotlib.colors.hsv_to_rgb(imgHSV)

# show image
plt.imshow(img,extent=extent)
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

```
[ ]:
```