2023-05-15_ChartTypes_012_VectorFields_DeformationImages

May 25, 2023

[1]:
```
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')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 Vector fields

1.1 Deformation plots

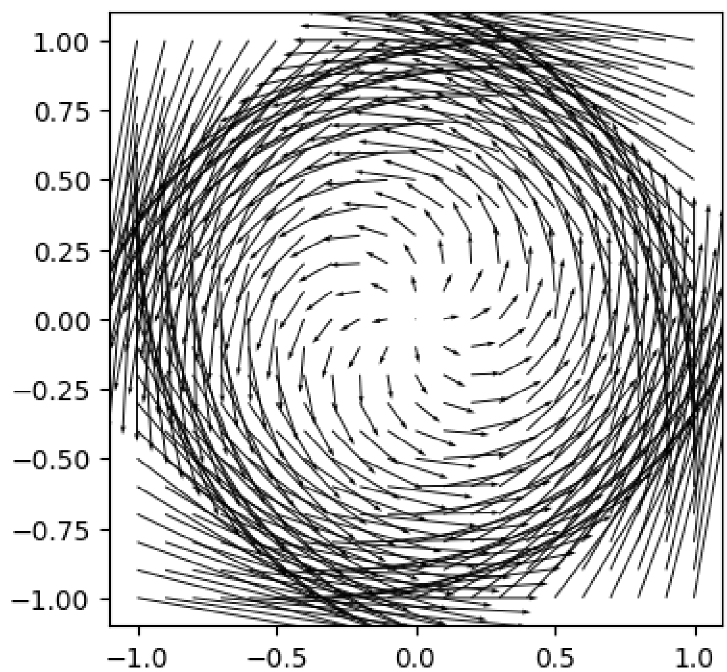
[2]:
```
# build some non-trivial examples of deformations for us to visualize
def map1(x,y):
    """merely rescale radius a bit"""
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**0.8
    phiOut=phiIn
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v

def map1inv(x,y):
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**(1/0.8)
    phiOut=phiIn
    u=np.cos(phiOut)*rOut
```

```
    v=np.sin(phiOut)*rOut
    return u,v


def map2(x,y):
    """rescale radius + fixed rotation"""
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**0.8
    phiOut=phiIn+0.25*np.pi
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v

def map2inv(x,y):
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**(1/0.8)
    phiOut=phiIn-0.25*np.pi
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v


def map3(x,y):
    """rescale radius + adaptive rotation"""
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**0.8
    phiOut=phiIn+0.25*np.pi*rIn
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v

def map3inv(x,y):
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**(1/0.8)
    phiOut=phiIn-0.25*np.pi*rOut
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v


usemap=map3
usemapinv=map3inv
```

```
def getUniformData(nPts1d,rng,usemap):
    x = np.linspace(*rng,num=nPts1d)
    X, Y = np.meshgrid(x, x)
    U, V = usemap(X,Y)
    return X,Y,U,V
```

1.1.1 Visualization as quiver plot

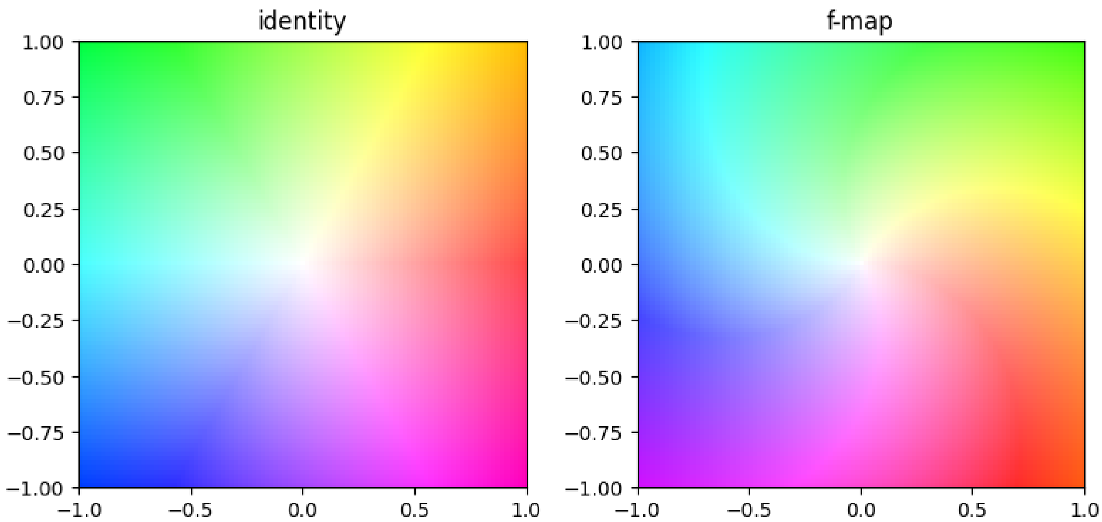
[3]:
```
X,Y,U,V=getUniformData(21,[-1,1],usemap)

fig=plt.figure(figsize=(4,4))
ax=fig.add_subplot(aspect=1.)
ax.quiver(X,Y,U-X,V-Y,pivot="tail",angles="xy",scale_units="xy",scale=1.)
plt.tight_layout()
plt.show()

# shown earlier: this tends not to work well on actual maps with long vectors
```



1.1.2 Visualization as HSV image

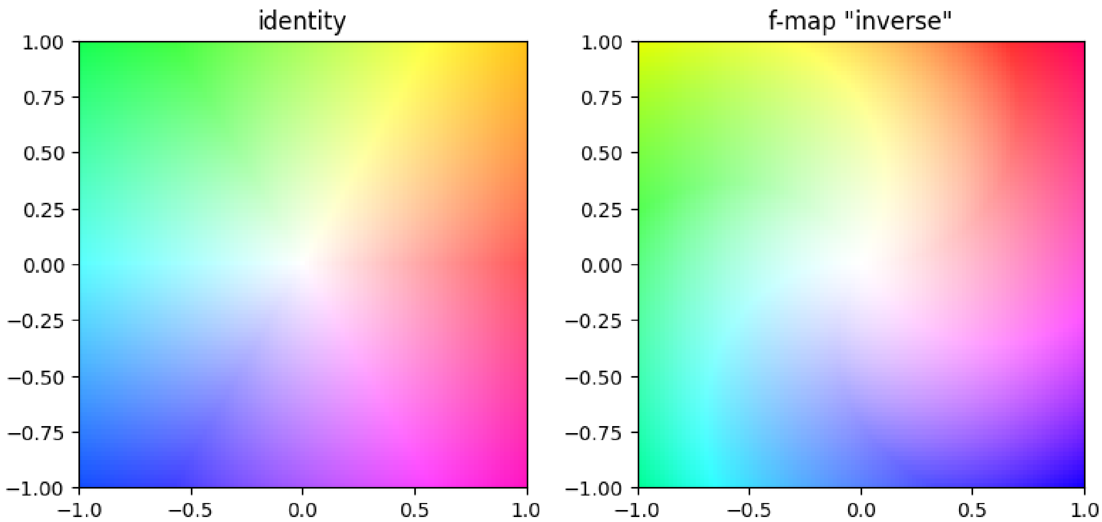
[4]:
```
def HSVfromUV(U,V,magmax=None):
    colorsHSV=np.zeros(U.shape+(3,))
    # set Saturation to 1
    colorsHSV[:,:,2]=1.
    # set Value to (rescaled) magnitude
    magnitude=(U**2+V**2)**0.5
    if magmax is None:
        vmax=np.max(magnitude)
    else:
        vmax=magmax
    colorsHSV[:,:,1]=np.clip(magnitude/vmax,0.,1.)
    # set Hue to orientation
    phi=np.mod(np.arctan2(V,U)/(2*np.pi),1.)
    colorsHSV[:,:,0]=phi
    # conver to RGB
    colorsRGB=matplotlib.colors.hsv_to_rgb(colorsHSV)
    return colorsRGB
```

[5]:
```
nPts1d=101
rng=[-1,1]
X,Y,U,V=getUniformData(nPts1d,rng,usemap)

magmax=max(np.max((U**2+V**2)**0.5),np.max((X**2+Y**2)**0.5))

img1=HSVfromUV(X,Y,magmax)
# try: color each pixel by color corresponding to target position
img2=HSVfromUV(U,V,magmax)

fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
plt.imshow(img1,extent=rng+rng,origin="lower")

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map")
plt.imshow(img2,extent=rng+rng,origin="lower")

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




```
[6]: # comments:
     # * see how perceptually non-uniform HSV is!
     # * in this way, displayed side-by-side like this, it seems like map is
     #   a rotation in the opposite direction!
     # -> we should color each pixel by the color corresponding to its preimage
```

```
[7]: nPts1d=101
     rng=[-1,1]
     X,Y,U,V=getUniformData(nPts1d,rng,usemapinv)

     magmax=max(np.max((U**2+V**2)**0.5),np.max((X**2+Y**2)**0.5))

     img1=HSVfromUV(X,Y,magmax)
     # try: color each pixel by color corresponding to target position
     img3=HSVfromUV(U,V,magmax)

     fig=plt.figure(figsize=(8,4))

     fig.add_subplot(1,2,1,aspect=1.)
     plt.title("identity")
     plt.imshow(img1,extent=rng+rng,origin="lower")

     fig.add_subplot(1,2,2,aspect=1.)
     plt.title("f-map \"inverse\"")
     plt.imshow(img3,extent=rng+rng,origin="lower")

     plt.tight_layout()

     plt.show()
```

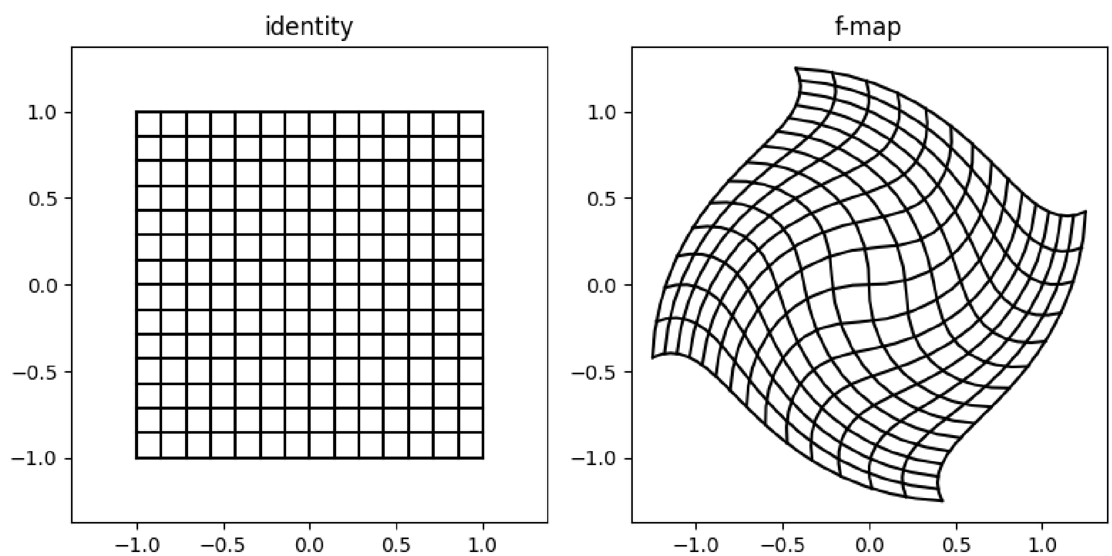


1.1.3 Visualization as warped grid

[8]:
```
# generate data for evaluating the map on a nice regular grid
# make spacings in X and Y direction very different, so that we can get
# good visualization as warped grid (will become clear in a moment)
def getLines(rng,nLines,nPtsPerLine):
    x=np.linspace(*rng,num=nLines)
    y=np.linspace(*rng,num=nPtsPerLine)
    X, Y = np.meshgrid(x, y)
    X=X.transpose()
    Y=Y.transpose()
    return np.concatenate((X,Y)),np.concatenate((Y,X))
```

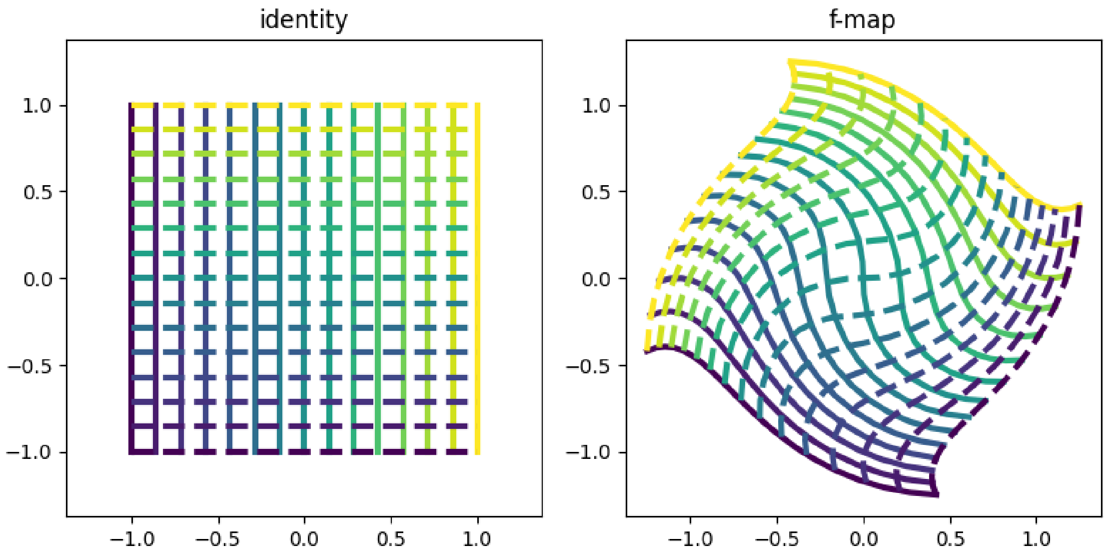
[9]:
```
rng=[-1,1]
nLines=15
nPtsPerLine=20
X,Y=getLines(rng,nLines,nPtsPerLine)
U,V=usemap(X,Y)

# get consistent plot range in both images
magmax=max([np.max(np.abs(Z)) for Z in [X,Y,U,V]])*1.1


fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
for x,y in zip(X,Y):
```

```
    plt.plot(x,y,c="k")
plt.xlim([-magmax,magmax])
plt.ylim([-magmax,magmax])

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map")
for x,y in zip(U,V):
    plt.plot(x,y,c="k")
plt.xlim([-magmax,magmax])
plt.ylim([-magmax,magmax])

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



[10]:
```
# experiment: add color to lines to make identification a bit more obvious in
#   ambiguous cases
# try also: line style (solid vs dashed)
# make lines a bit thicker, so that these features (color, dash pattern)
#   can be seen better

colors=np.linspace(0,1,num=nLines)
colors=np.concatenate((colors,colors))
linestyle=["solid" for _ in range(nLines)]+["dashed" for _ in range(nLines)]

fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
```

```
for x,y,c,ls in zip(X,Y,colors,linestyle):
    plt.plot(x,y,c=cm.viridis(c),ls=ls,lw=3)
plt.xlim([-magmax,magmax])
plt.ylim([-magmax,magmax])

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map")
for x,y,c,ls in zip(U,V,colors,linestyle):
    plt.plot(x,y,c=cm.viridis(c),ls=ls,lw=3)
plt.xlim([-magmax,magmax])
plt.ylim([-magmax,magmax])

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

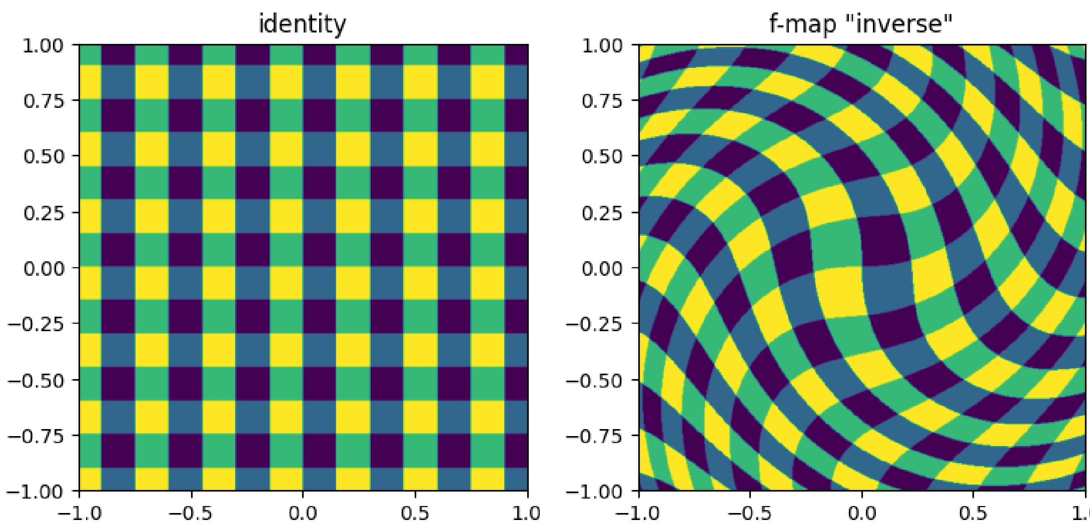


1.1.4 Visualization as warped grid, variant 2

This is a fusion of the two previous approaches. Visualize deformation as colored image, but instead of encoding full position as color, only encode position in a grid.

[11]:
```
def colorFromPos(X,Y,res):
    # divide coords into square grid of edge length res
    # determine color of "which square" pixel is in
    indexX=np.clip(np.floor(np.mod((X/res),2)),0,1)
    indexY=np.clip(np.floor(np.mod((Y/res),2)),0,1)
    return indexX*0.67+indexY*0.33
```

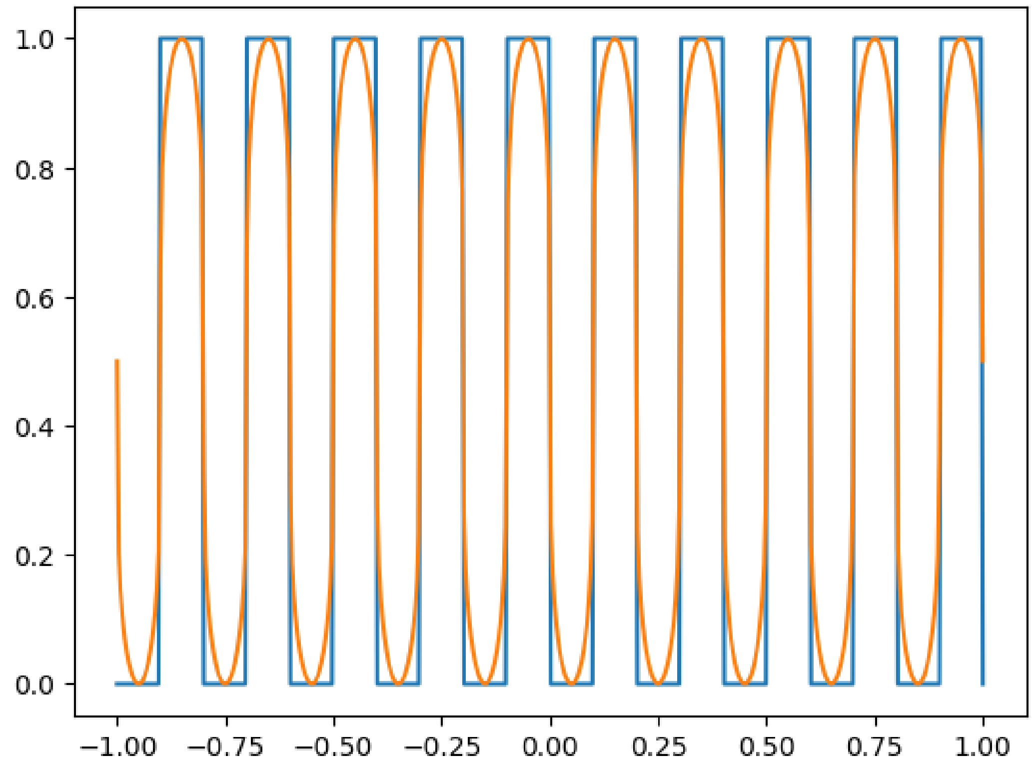
[12]:
```python
nPts1d=500
rng=[-1,1]
res=0.15

X,Y,U,V=getUniformData(nPts1d,rng,usemapinv)

img1=colorFromPos(X,Y,res)
img3=colorFromPos(U,V,res)

fig=plt.figure(figsize=(8,4))

# here an interpolation mode that is not "nearest" is more appropriate
# to dampen discretization artefacts
fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
plt.imshow(img1,extent=rng+rng,origin="lower",interpolation="bilinear")

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map \"inverse\"")
plt.imshow(img3,extent=rng+rng,origin="lower",interpolation="bilinear")

plt.tight_layout()

plt.show()
```



[13]:
```python
# another alternative: use a smoother pattern function
```

[14]:
```python
# visualize discrete grid vs smoth oscillation
def comb(x,res):
```

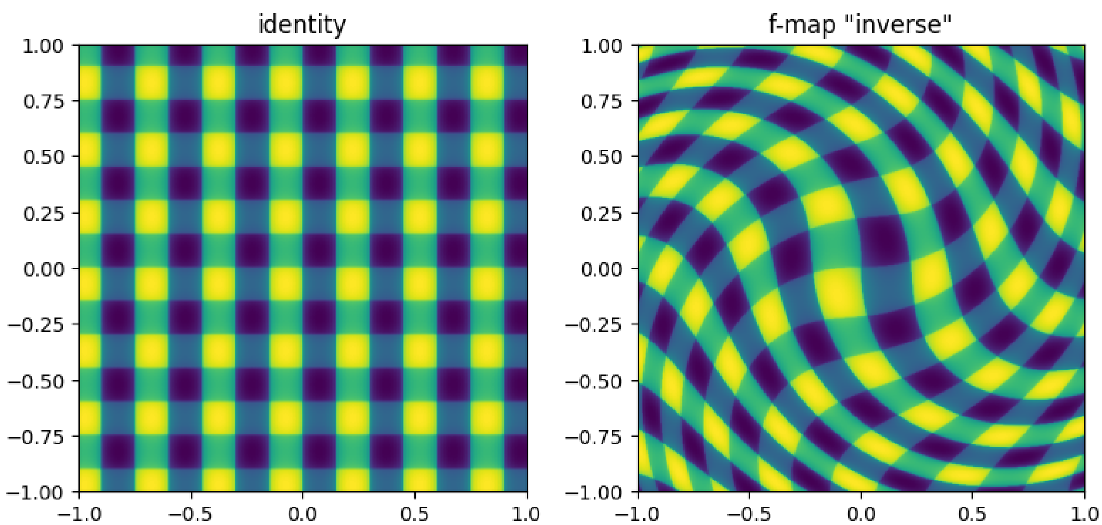
```
    return np.floor(np.mod(x/res,2))
def smooth_comb(x,res):
    cosdat=-np.sin(x/res*np.pi)
    return 0.5*(1+np.sign(cosdat)*np.abs(cosdat)**0.25)
x=np.linspace(-1,1,num=500)
plt.plot(x,comb(x,0.1))
plt.plot(x,smooth_comb(x,0.1))
plt.show()
```



[15]:
```
def colorFromPos2(X,Y,res):
    # divide coords into square grid of edge length res
    # determine color of "which square" pixel is in
    indexX=smooth_comb(X,res)
    indexY=smooth_comb(Y,res)
    return indexX*0.67+indexY*0.33
```

[16]:
```
nPts1d=500
rng=[-1,1]
res=0.15

X,Y,U,V=getUniformData(nPts1d,rng,usemapinv)
```

```
img1=colorFromPos2(X,Y,res)
img3=colorFromPos2(U,V,res)

fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
plt.imshow(img1,extent=rng+rng,origin="lower",interpolation="bilinear")

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map \"inverse\"")
plt.imshow(img3,extent=rng+rng,origin="lower",interpolation="bilinear")

plt.tight_layout()

plt.show()
```



[ ]:

```
```