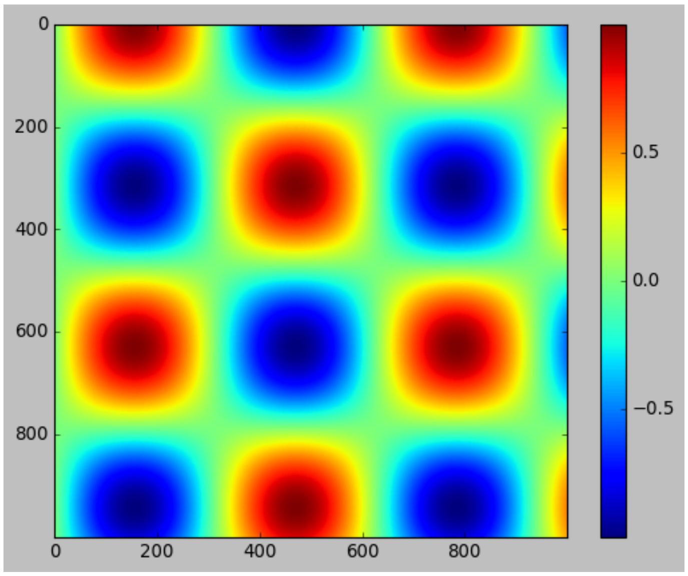
In [1]:
```
import matplotlib.pyplot as plt
plt.style.use("classic")
```

In [2]:
```
%matplotlib inline
import numpy as np
```

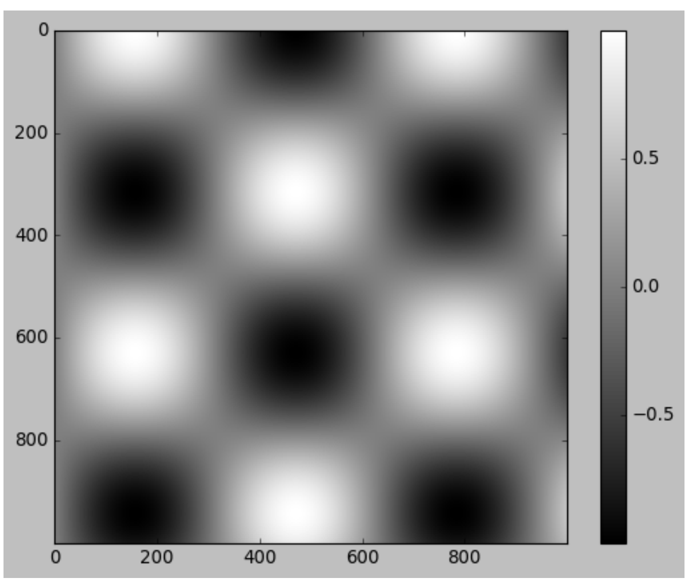
In [6]:
```
x=np.linspace(0,10,1000)
G=np.sin(x)*np.cos(x[:,np.newaxis])
plt.imshow(G)
plt.colorbar()
```

Out[6]: <matplotlib.colorbar.Colorbar at 0x1f98d4c0520>



In [9]:
```
#for black and white graph we give color map as gray in imshow
plt.imshow(G,cmap="gray")
plt.colorbar()
```

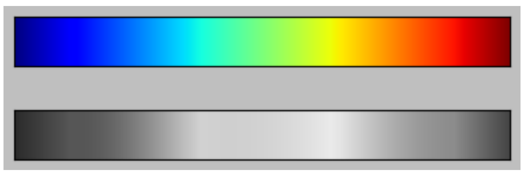
Out[9]: <matplotlib.colorbar.Colorbar at 0x1f98ee4f310>



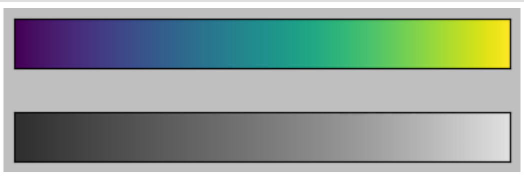
In [10]:
```
#choosing an appropriate colormap
#sequential colormaps-made of one contnious color sequence like viridis
#divergent colormaps-have two distinct colors to show positive and negative deviations from mean like RdBu
#qualitative colormaps-mix of colors with no sequence like rainbow and jet
```

In [12]:
```
from matplotlib.colors import LinearSegmentedColormap
def grayscale_cmap(cmap):
    """Return a grayscale version of the given colormap"""
    cmap = plt.cm.get_cmap(cmap)
    colors = cmap(np.arange(cmap.N))
    # convert RGBA to perceived grayscale luminance
    # cf. http://alienryderflex.com/hsp.html
    RGB_weight = [0.299, 0.587, 0.114]
    luminance = np.sqrt(np.dot(colors[:, :3] ** 2, RGB_weight))
    colors[:, :3] = luminance[:, np.newaxis]
    return LinearSegmentedColormap.from_list(cmap.name + "_gray", colors, cmap.N)
def view_colormap(cmap):
    """Plot a colormap with its grayscale equivalent"""
    cmap = plt.cm.get_cmap(cmap)
    colors = cmap(np.arange(cmap.N))
    cmap = grayscale_cmap(cmap)
    grayscale = cmap(np.arange(cmap.N))
    fig, ax = plt.subplots(2, figsize=(6, 2),subplot_kw=dict(xticks=[], yticks=[]))
    ax[0].imshow([colors], extent=[0, 10, 0, 1])
    ax[1].imshow([grayscale], extent=[0, 10, 0, 1])
```

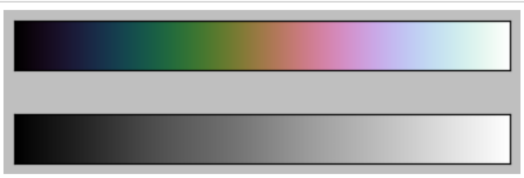
In [13]:
```
view_colormap("jet")#we avoid using greyscale due to differing brightness as our eyes may be drawn to unwanted graph areas
#default colormap is jet but jet is qualitative and hence does not give a good idea about intensities
```



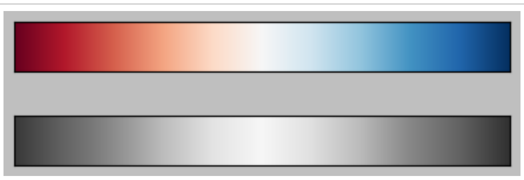
In [14]:
```
view_colormap("viridis")#viridis is the standard colormap for sequential data
```



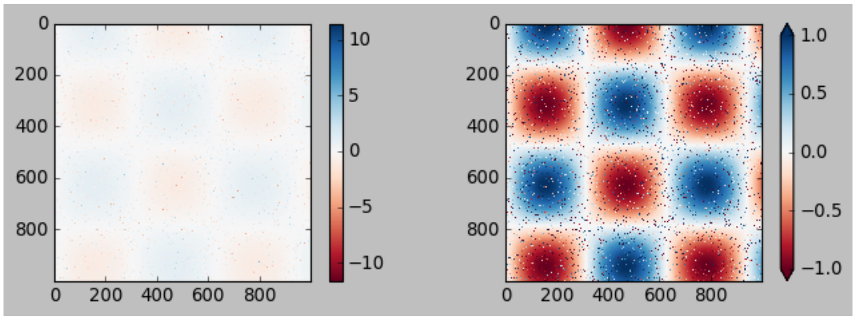
In [15]:
```
view_colormap("cubehelix")#cubehelix has rainbow colors but with a sequence for continous data
```



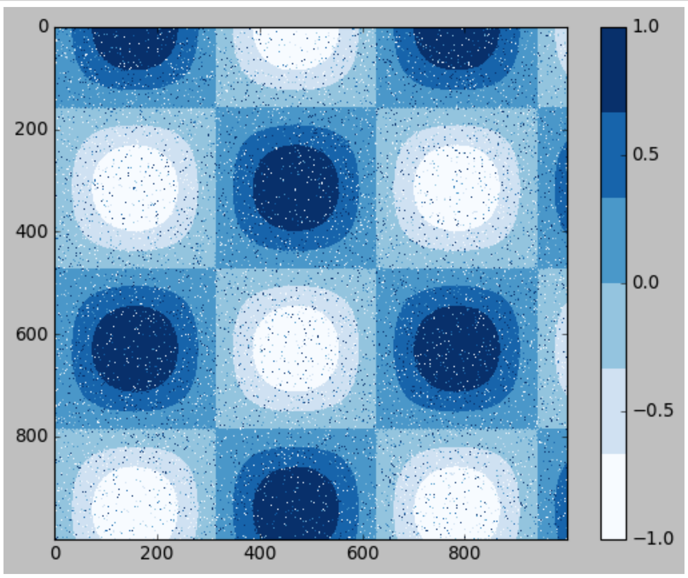
In [16]:
```
view_colormap("RdBu")#for showing positive and negative deviations from mean
```



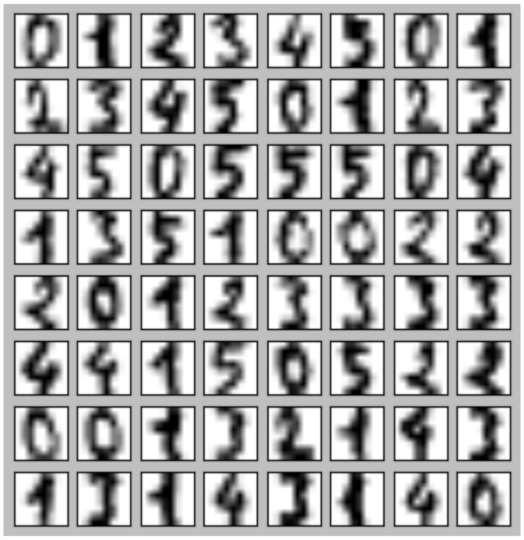
In [24]:
```
#extend property will help as indicate out of bounds values,it is used when image is subject to external signals
# make noise in 1% of the image pixels
speckles_array=np.random.random(G.shape)<0.01#boolean array of previous array G with condition less than 0.01
G[speckles_array]=np.random.normal(0,3,np.count_nonzero(speckles_array))#counting number of true elements and creating
#random array and then reassigning it to masked array of G
plt.figure(figsize=(10,3))
plt.subplot(1,2,1)
plt.imshow(G,cmap="RdBu")
plt.colorbar()
plt.subplot(1,2,2)#subplot to be discussed in future chapters
plt.imshow(G,cmap="RdBu")
plt.colorbar(extend="both")#extend will place triangular ends in colorbar to show out of bounds values
plt.clim(-1,1)#clim is used for setting limit in colorbar
```



In [25]:
```
#colormaps are continous,we can however change colormaps to discrete using plt.cm.get_cmap() and pass map and number of bins
plt.imshow(G,cmap=plt.cm.get_cmap("Blues",6))#colorbar will now correspond to 6 value ranges with Blues cmap
plt.colorbar()
plt.clim(-1,1)#setting limit for colorbar
```



In [27]:
```
# Load images of the digits 0 through 5 and visualize several of them
from sklearn.datasets import load_digits
digits = load_digits(n_class=6)
fig, ax = plt.subplots(8, 8, figsize=(6, 6))
for i, axi in enumerate(ax.flat):
    axi.imshow(digits.images[i], cmap="binary")
    axi.set(xticks=[], yticks=[])
```



In [28]:
```
# project the digits into 2 dimensions using IsoMap
from sklearn.manifold import Isomap
iso = Isomap(n_components=2)
projection = iso.fit_transform(digits.data)
```

```
C:\Users\kdmag\anaconda3\lib\site-packages\sklearn\manifold\_isomap.py:324: UserWarning: The number of connected components of
the neighbors graph is 2 > 1. Completing the graph to fit Isomap might be slow. Increase the number of neighbors to avoid this
issue.
  self._fit_transform(X)
C:\Users\kdmag\anaconda3\lib\site-packages\scipy\sparse\_index.py:103: SparseEfficiencyWarning: Changing the sparsity structure
of a csr_matrix is expensive. lil_matrix is more efficient.
  self._set_intXint(row, col, x.flat[0])
```

In [29]:
```
# plot the results
plt.scatter(projection[:, 0], projection[:, 1], lw=0.1,c=digits.target, cmap=plt.cm.get_cmap('cubehelix', 6))
plt.colorbar(ticks=range(6), label='digit value')
plt.clim(-0.5, 5.5)
```