Bug #1:

Link: https://github.com/matplotlib/matplotlib/issues/5004
Description:

The image produced in OffsetImage (image.BboxImage) does not properly perform alpha blending. Whereas if the same image is drawn by figure.figimage (image.FigureImage) there is no issue.

Replication:

```
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as image
from matplotlib.offsetbox import TextArea, DrawingArea, OffsetImage, AnnotationBbox

if __name__ == '__main__':
    matplotlib.rcParams['figure.facecolor'] = 'ffffff'

    fig, ax = plt.subplots()

    ax.set_title('Draw image')
    ax.set_xlim((-2,2))
    ax.set_ylim((-2,2))

    img = image.imread('test.png')
    fig.figimage(img, 0, 0)

    imagebox = OffsetImage(img, zoom=1)
    annotation = AnnotationBbox(imagebox, (-1.5, -1.5), frameon=False, xycoords='data',
boxcoords="offset points", pad=0)
    ax.add_artist(annotation)

    plt.show()
```

What classes/parts of matplotlib will need to be looked at/fixed:

Since the class that is giving this issue is BboxImage it will be the focus of the bug fix. The class is found in image.py and since FigureImage works (in same file) hopefully it will give us a better understanding of how to fix the bug.

Bug #2:

Link: https://github.com/matplotlib/matplotlib/issues/5472
Description:

Call legend.get_frame() will always return a frame, in other words it is incapable of throwing an error or some other reasonable “something went wrong!” flag.

Replication:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.random.rand(10,1)
# graph 1 - the green legend frame is generated properly
plt.plot(a, label='label')
legend = plt.legend()
frame = legend.get_frame()
frame.set_facecolor('green')
plt.show()

import seaborn as sns
# graph 2 - Seaborn overlaps the legend frame
plt.plot(a, label='label')
legend = plt.legend()
frame = legend.get_frame()
frame.set_facecolor('green')
plt.show()
```

What classes/parts of matplotlib will need to be looked at/fixed:

The bug is a result of bad coding as get_frame simply returns a frame class - it needs a way to check for potential conflicts. The _legendPatch variable in the Legend class has a visible attribute perfect for this situation but may require tinkering with the rcParams class and rcsetup.py.

Bug #3:

Link: https://github.com/matplotlib/matplotlib/issues/3292

Description:

When importing matplotlib __init__.py is run and it file searches for home directories or a temporary directory as defined by the environment. Some users would like the ability to control where matplotlib searches for these directories, particularly on systems where $HOME is defined but, may not be mounted.

Replication:

This feature would be used whenever a module from matplotlib is imported.

What classes/parts of matplotlib will need to be looked at/fixed:

The __init__.py file is run whenever a module from matplotlib is imported. The feature would have to be added to the _get_home() function in this file that returns where the home.

Bug #4:

Link: https://github.com/matplotlib/matplotlib/issues/6000

Description:

When attempting to set the visibility of a streamplot to be invisible, the resultant plot still has it's arrows showing.

Replication:

```
import numpy as np
import matplotlib.pyplot as plt

# Data
x = np.linspace(-10, 10, 10)
y = np.linspace(-10, 10, 10)
X, Y = np.meshgrid(x, y)
U = X
V = X**2

# basic streamline plot
plt.figure()
sp1 = plt.streamplot(x, y, U, V)

# Attempt to hide lines and arrows from streamline plot
plt.figure()
sp2 = plt.streamplot(x, y, U, V)
sp2.lines.set_visible(False)
sp2.arrows.set_visible(False)

plt.show()
```

What classes/parts of matplotlib will need to be looked at/fixed:
Introspection into how `streamplot.py` and figures (`figure.py`) interact with each other, definately look at the `set_visible()` method in artist.py, the states `self._visible` , `self.pchanged()`, and whether or not it has been implemented properly inside class `figure`.

Bug #5:

Link: https://github.com/matplotlib/matplotlib/issues/6002
Description:
Using pyplot.streamplot with a non-default start_points argument leads to a misplacement of the streamline.
Replication:

```
import numpy as np
import matplotlib.pyplot as plt

# Data
x = np.linspace(-10, 10, 10)
y = np.linspace(-10, 10, 10)
X, Y = np.meshgrid(x, y)
U = X*0 + 1
V = X*0
start_points = [[0, 0]]
```

```
# Base streamline plot
plt.figure()
sp1 = plt.streamplot(x, y, U, V, color=[.5]*3)

# Streamline plot with 'start_points' argument
sp2 = plt.streamplot(x, y, U, V, start_points=start_points, color='r')
plt.plot(*start_points[0], marker='o', label="Starting point")
plt.plot([], [], color='r', label="Associated streamline")

# Legend and limits
plt.xlim(-10, 10)
plt.ylim(-10, 10)
plt.legend(numpoints=1)

plt.show()
```

What classes/parts of matplotlib will need to be looked at/fixed:

Streamplot.py appears to be mishandling the starting_points input variable by possibly miscalculating the coordinate conversion for the grid. When starting_points is defaulted to None Streamplot generates coordinates at the origin thus avoiding the bug as seen in the our examples first streamplot.