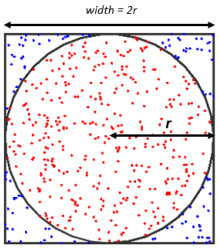
Estimating π using the Monte Carlo Method

Explanation

Assume we have a circle of radius *r*. This circle can be inscribed inside a square of width *2r*. Pi can then be estimated by randomly selecting an *x* and *y* point inside the square and looking at the number of points that land inside and outside the circle



$$Area_c = \pi r^2$$
$$Area_s = (2r)^2 = 4r^2$$
$$\frac{Area_c}{Area_s} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$
$$\pi \approx \frac{4 Area_c}{Area_s}$$

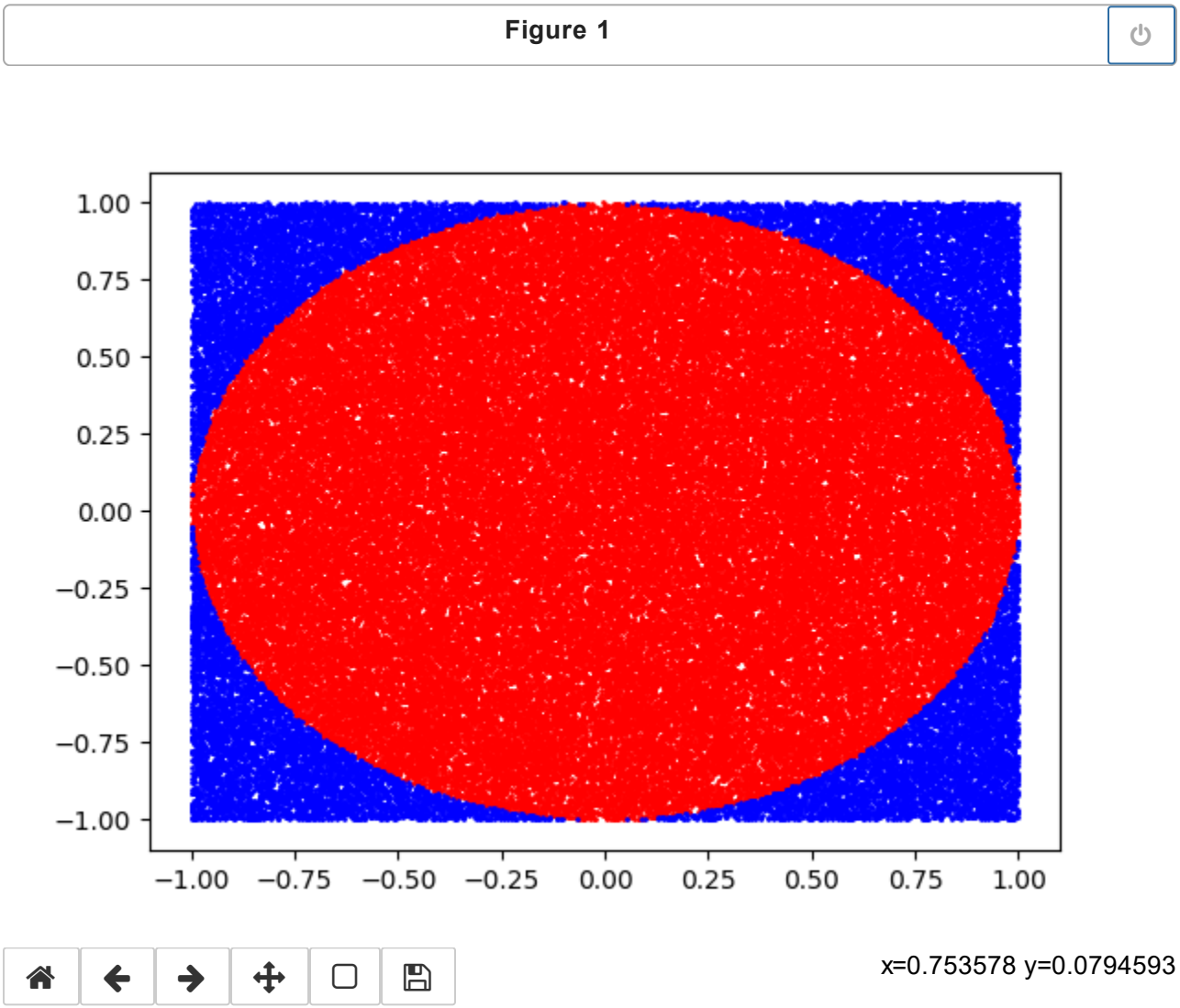
This means that computing the ratio of the areas and multiplying by 4 should approximate π

In [9]:

```
1 import random
2 import math
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from ipywidgets import IntProgress
6 from IPython.display import display
7 %matplotlib notebook
```

In [34]:

```
NUMBER_OF_SAMPLES = 10**5
REDRAW_EVERY_X = NUMBER_OF_SAMPLES/10

inside_circle, outside_circle = [], []
fig, ax = plt.subplots()

def plot_animate():
    ax.clear()
    _inside_points = np.array(inside_circle)
    _outside_points = np.array(outside_circle)

    _inside_x = 0 if len(_inside_points) == 0 else _inside_points[:,0]
    _inside_y = 0 if len(_inside_points) == 0 else _inside_points[:,1]
    _outside_x = 0 if len(_outside_points) == 0 else _outside_points[:,0]
    _outside_y = 0 if len(_outside_points) == 0 else _outside_points[:,1]

    ax.plot(_inside_x, _inside_y, marker='o', markersize=1, color='r',linestyle
    ax.plot(_outside_x, _outside_y, marker='o', markersize=1, color='b',linesty

    fig.canvas.draw()

# Will loop through randomly genarated x and y points centered at zero
# ranging from -1 to 1. Distance will be calculated and x and y points
# added to a list for plotting
for i in range(NUMBER_OF_SAMPLES):
    x = -1 + 2*random.random()
    y = -1 + 2*random.random()
    r = math.sqrt(x**2 + y**2)
    inside_circle.append([x,y]) if (r<1) else outside_circle.append([x,y])
    if not (i % REDRAW_EVERY_X): plot_animate()



_inside, _outside = len(inside_circle), len(outside_circle)
print("Points inside circle: {}".format(_inside))
print("Points outside circle: {}".format(_outside))
print("Estimate for Pi: {}".format(4*_inside/(_inside+_outside)))
```



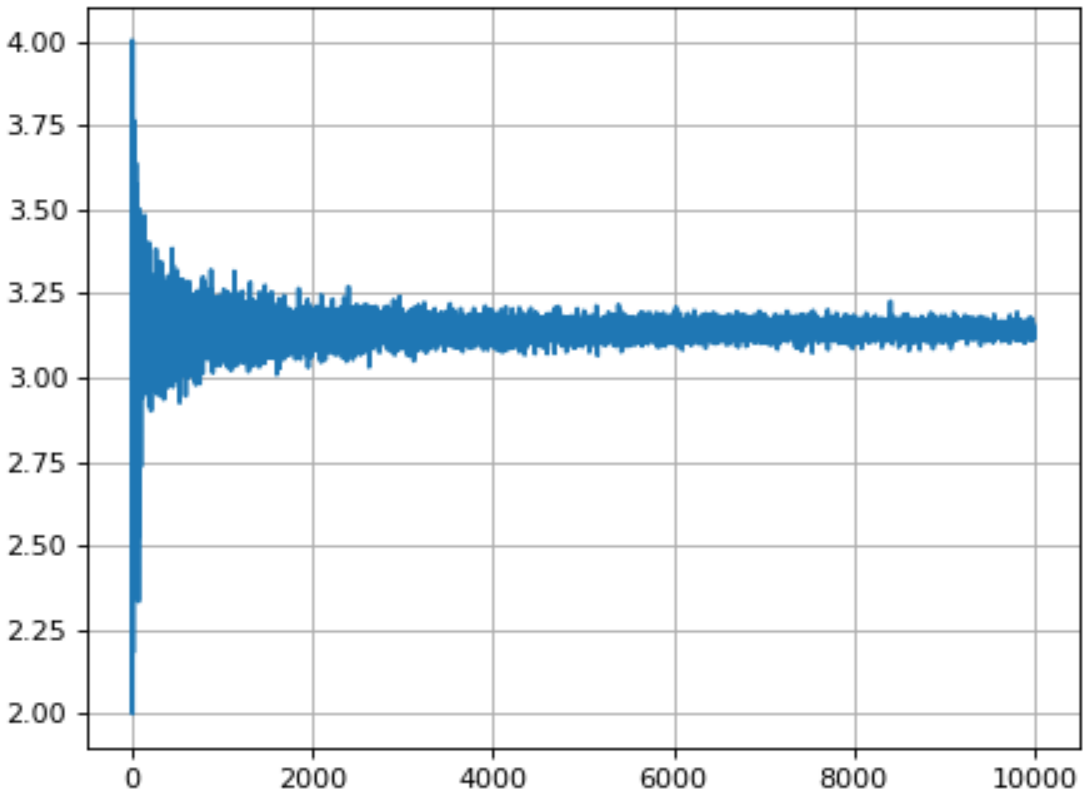

```
Points inside circle: 78688
Points outside circle: 21312
Estimate for Pi: 3.14752
```

Calculate how good the approximation gets as the number of samples increases

In [11]:

```
NUMBER_OF_SAMPLES = 10**4
pi_estimate = []

f = IntProgress(min=1, max=NUMBER_OF_SAMPLES)
display(f)
print("Checking values up to {}...".format(NUMBER_OF_SAMPLES))
for i in range(1, NUMBER_OF_SAMPLES):
    f.value += 1
    _estimate = 0
    inside_circle, outside_circle = 0, 0
    for j in range(i):
        x = -1 + 2*random.random()
        y = -1 + 2*random.random()
        r = math.sqrt(x**2 + y**2)
        if r<1:
            inside_circle += 1
        else:
            outside_circle+= 1
    _estimate = 4*inside_circle/(inside_circle+outside_circle)
    pi_estimate.append(_estimate)

print("Done simulating. Plotting {} points...".format(len(pi_estimate)))
fig, ax = plt.subplots()
ax.grid(True)
ax.plot(pi_estimate)
```

```
Checking values up to 10000...
Done simulating. Plotting 9999 points...
```



Out[11]: [<matplotlib.lines.Line2D at 0x19b32ed0f60>]

/