ASTR 119: Session 17

Random Numbers

Outline

1) Visualization of the Day
2) Final projects
3) Generating random numbers
4) Monte Carlo integration
5) Final project organization time
6) Save your work to GitHub

Final Projects

1) You will be asked to perform one of four final projects, working in a **group of up to 4 people of your choosing from your SECTION**. If you would like to be assigned to a group, please let me know immediately.

2) Please notify me of your group by **8:00am Thursday Nov 19**. Anyone who does not respond by **8:00am Thursday** letting me know their group or their desire to be assigned to a group will work on their own final project.

3) You will choose from one of the following 4 project topics, described in the following slides:
    1) Damped, driven pendulum
    2) Logistic map and chaos
    3) Astronomical source detection
    4) Monte Carlo Integration

4) Groups must select a final project topic by **Tues Nov 24**. Detailed instructions for each project will be distributed Thursday Nov 19.

5) Groups should be organized through GitHub, invite your group members and us (TAs) as collaborators on your project. Tag us (TAs) when your final project is ready to grade.

6) Each python module should indicate which student authored which part of the code.

7) Final projects are due (tags must happen by) **Tuesday, December 15, 2020 at 3pm.**

Random Number Generation

In science, we often have to “fake it till we make it”, meaning that in order to understand our experiments and their systematic uncertainties we need to model our data. Given both natural randomness and measurement error, these models amount to random number generation.

Often, we would like to generate random numbers distributed from a wide variety of distributions. If we can calculate the integral of the probability density distribution we wish to pull random numbers from, it is possible to use uniformly distributed random numbers to generate random numbers from a general distribution.

Given the complexity of this problem, let’s take some time to learn about publicly available libraries to help us with a variety of computational tasks including random number generation. This will be useful for many problems you might encounter in the future.

Random Number Generation

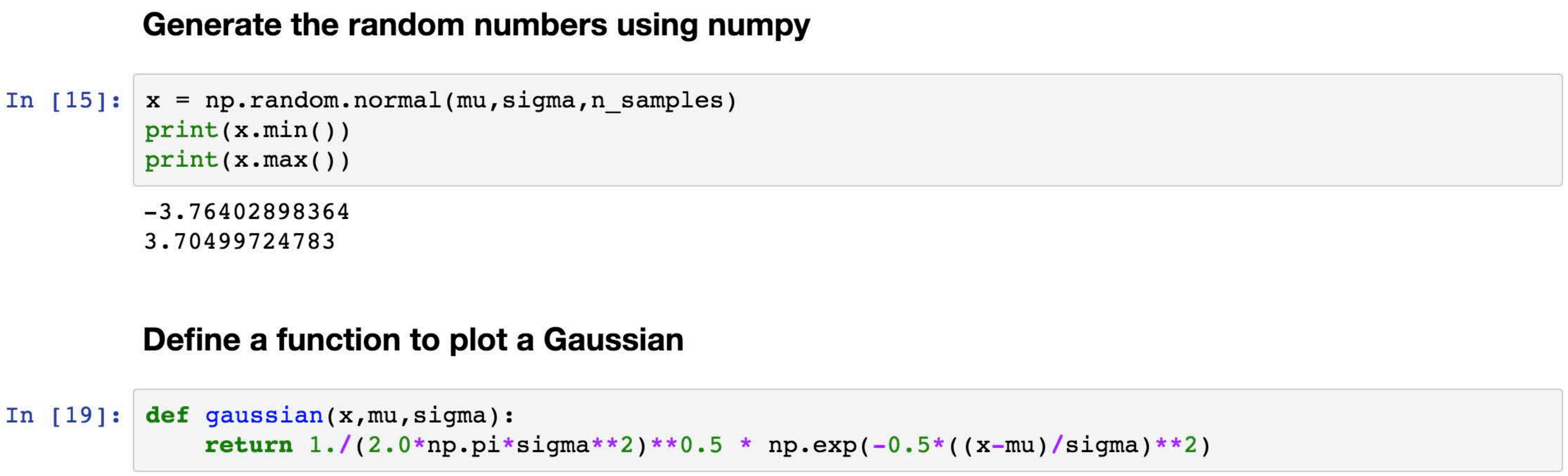
Use a random number generator to simulate a simple Gaussian process

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

Set some properties of the random system

```
In [5]: n_samples = 10000 #number of random samples to use
        n_bins    = 100   #number of bins for our histogram
        sigma     = 1.0   #rms width of the gaussian
        mu        = 0.0   #mean of the gaussian
```

Random Number Generation



Generate the random numbers using numpy

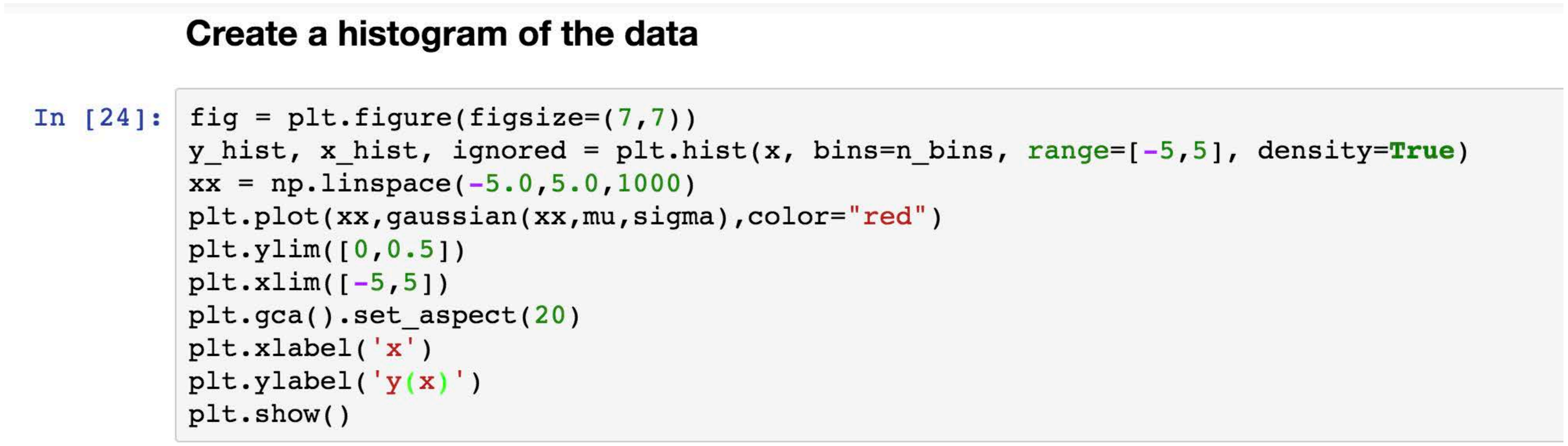
In [15]:
```
x = np.random.normal(mu,sigma,n_samples)
print(x.min())
print(x.max())
```

```
-3.76402898364
3.70499724783
```

Define a function to plot a Gaussian

In [19]:
```
def gaussian(x,mu,sigma):
    return 1./(2.0*np.pi*sigma**2)**0.5 * np.exp(-0.5*((x-mu)/sigma)**2)
```

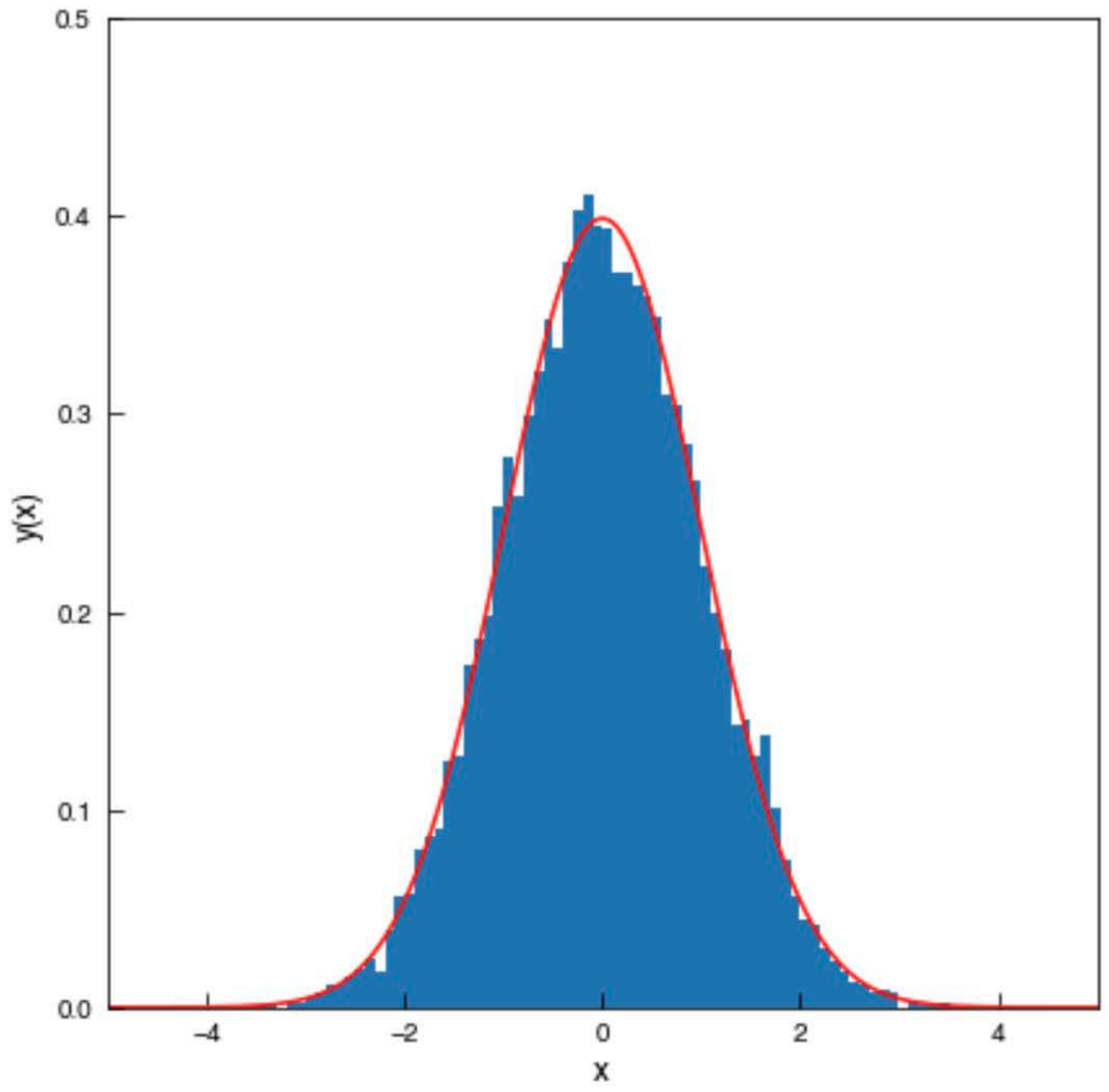
Random Number Generation



Create a histogram of the data

In [24]:
```
fig = plt.figure(figsize=(7,7))
y_hist, x_hist, ignored = plt.hist(x, bins=n_bins, range=[-5,5], density=True)
xx = np.linspace(-5.0,5.0,1000)
plt.plot(xx,gaussian(xx,mu,sigma),color="red")
plt.ylim([0,0.5])
plt.xlim([-5,5])
plt.gca().set_aspect(20)
plt.xlabel('x')
plt.ylabel('y(x)')
plt.show()
```

Random Number Generation

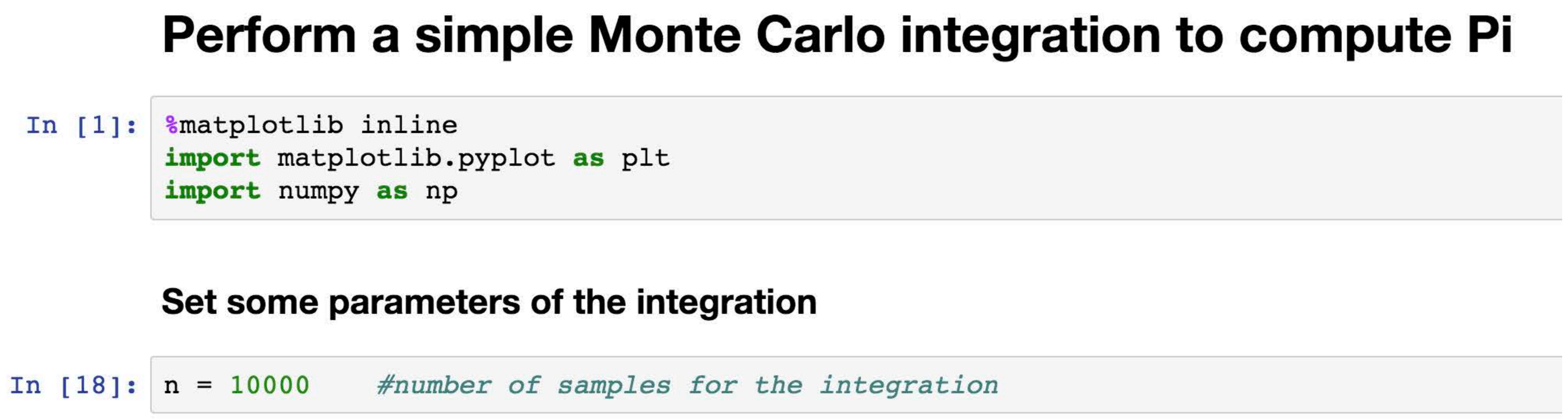


Monte Carlo Integration

One very useful application of random number generation is to perform Monte Carlo integration. Monte Carlo integration uses random numbers to sample a space and compute the integral of a given function by finding the ratio of the region under the curve to the total area probed by the samples.

I wanted to provide a simple example of Monte Carlo integration — the computation of Pi.

Monte Carlo Integration



Perform a simple Monte Carlo integration to compute Pi

```
In [1]:  %matplotlib inline
         import matplotlib.pyplot as plt
         import numpy as np
```

Set some parameters of the integration

```
In [18]: n = 10000     #number of samples for the integration
```

UC SANTA CRUZ

Monte Carlo Integration

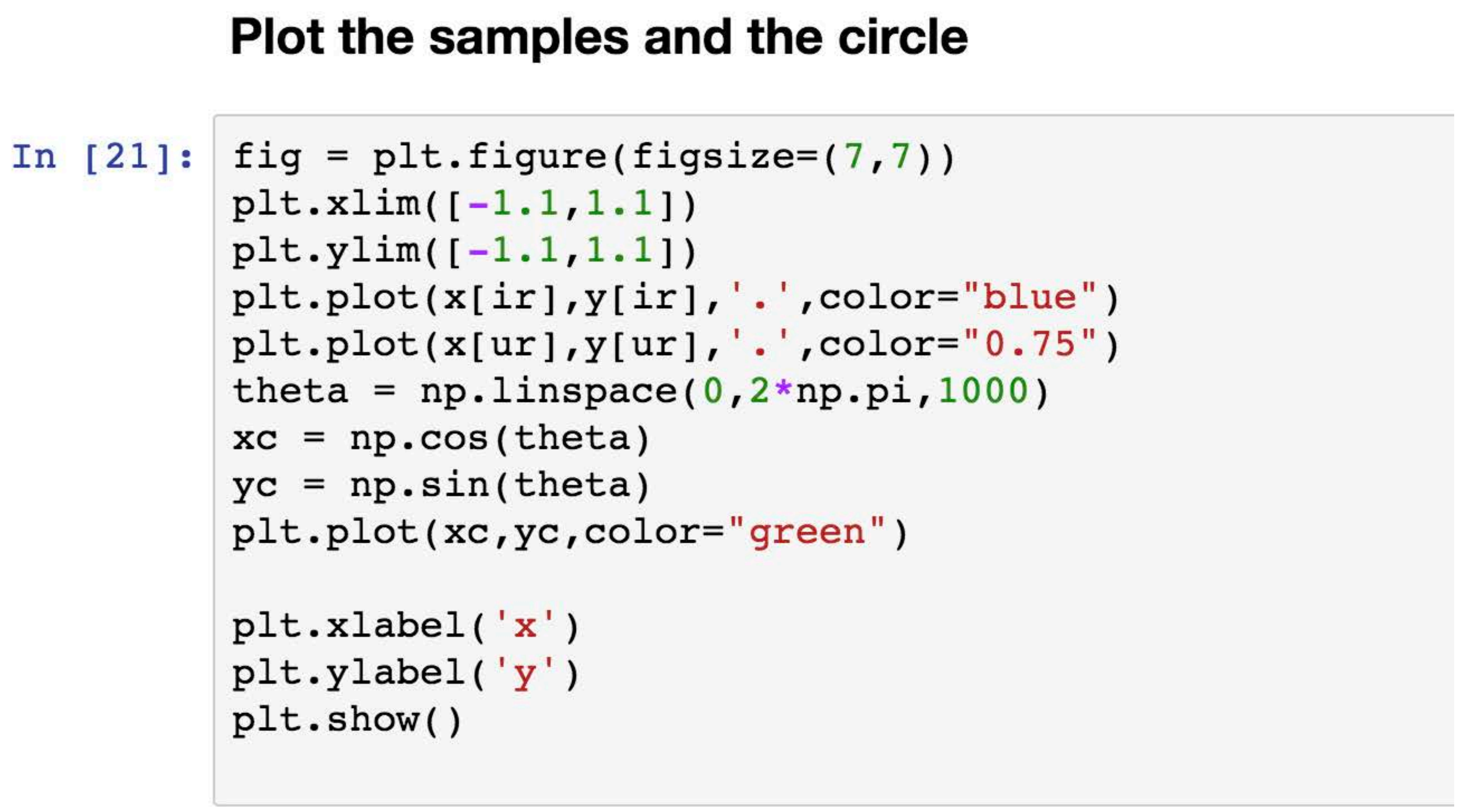
Make some uniformly sampled variables [-1,1]

```
In [19]: x = np.random.uniform(-1,1,n)
         y = np.random.uniform(-1,1,n)
```

Find the number of samples within the unit circle

```
In [20]: ir = np.where((x**2 + y**2)<1.0 )[0]
         ur = np.where((x**2 + y**2)>=1.0 )[0]
```

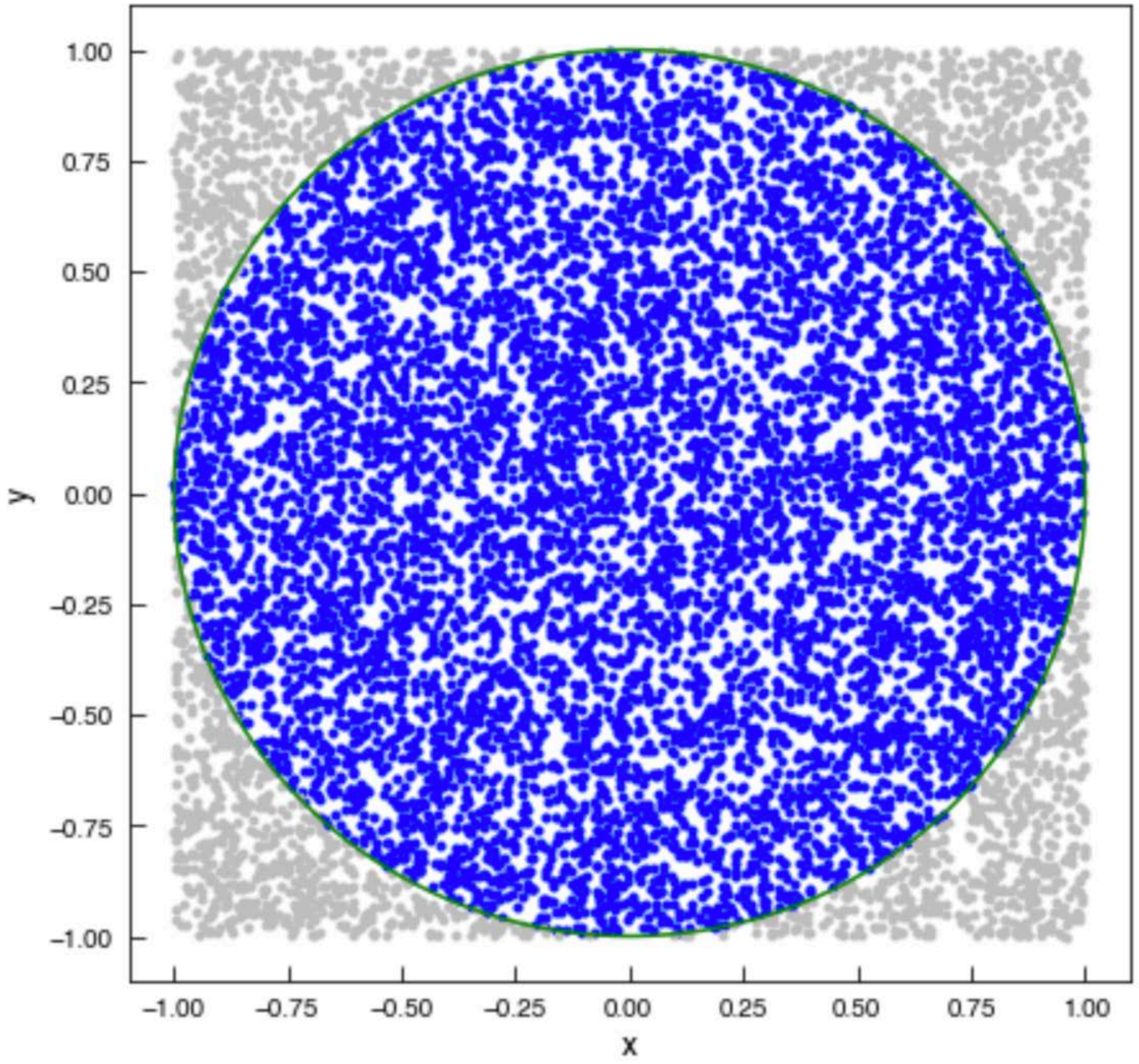
Monte Carlo Integration



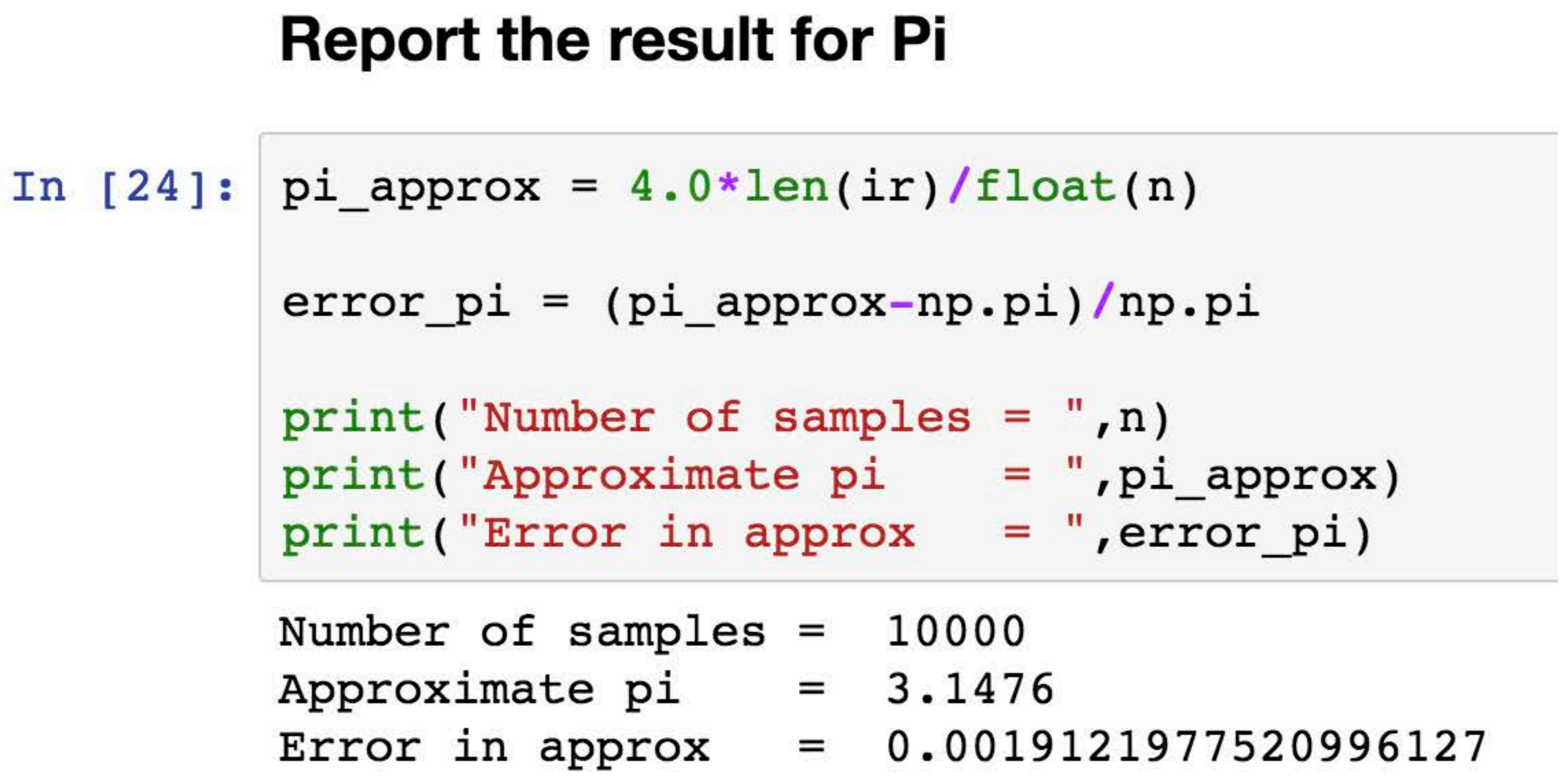
Plot the samples and the circle

In [21]:
```python
fig = plt.figure(figsize=(7,7))
plt.xlim([-1.1,1.1])
plt.ylim([-1.1,1.1])
plt.plot(x[ir],y[ir],'.',color="blue")
plt.plot(x[ur],y[ur],'.',color="0.75")
theta = np.linspace(0,2*np.pi,1000)
xc = np.cos(theta)
yc = np.sin(theta)
plt.plot(xc,yc,color="green")

plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Monte Carlo Integration



Monte Carlo Integration



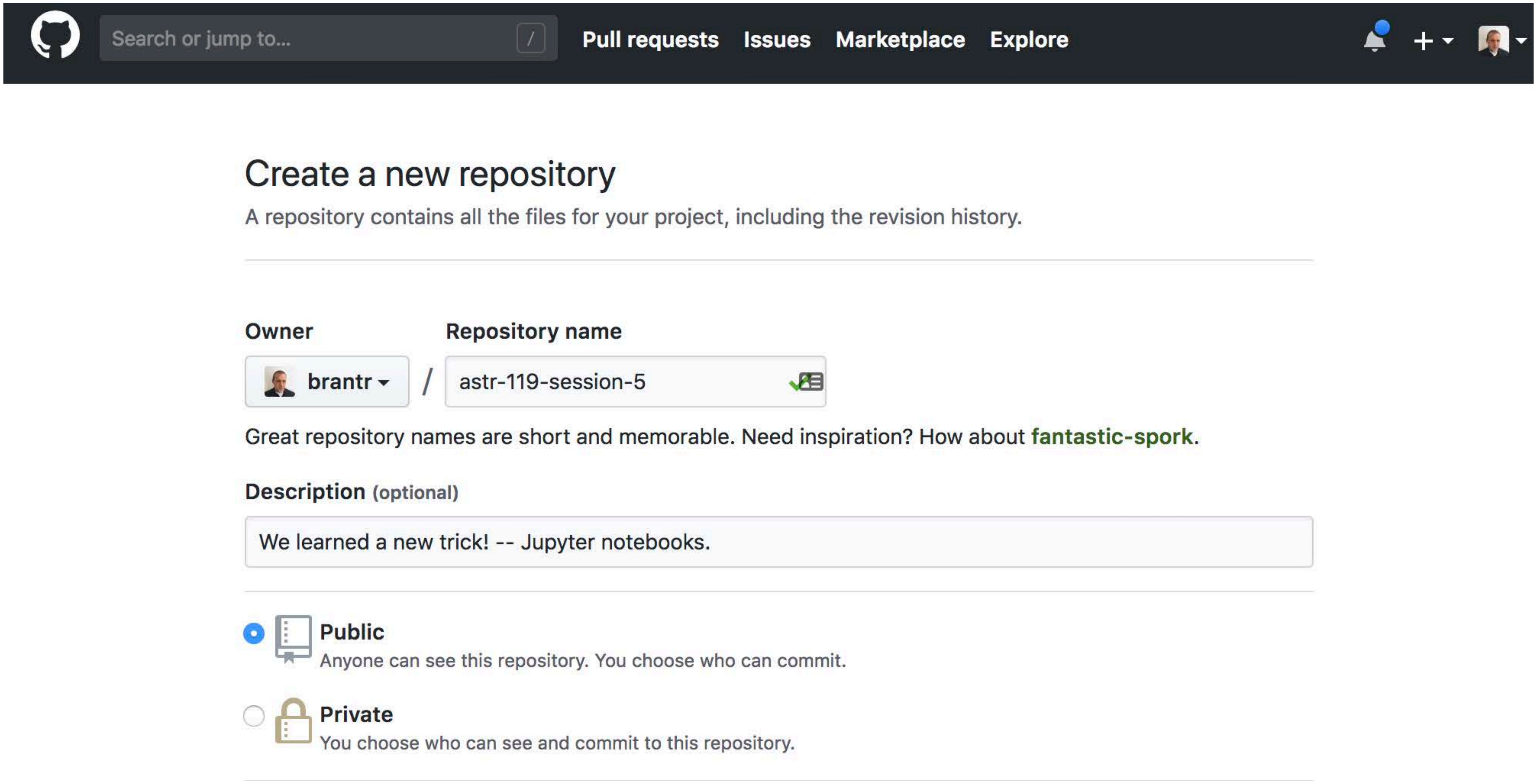
Report the result for Pi

```
In [24]: pi_approx = 4.0*len(ir)/float(n)

         error_pi = (pi_approx-np.pi)/np.pi

         print("Number of samples = ",n)
         print("Approximate pi    = ",pi_approx)
         print("Error in approx   = ",error_pi)
```

```
Number of samples =  10000
Approximate pi    =  3.1476
Error in approx   =  0.0019121977520996127
```

Save Your Work

Make a GitHub project “astr-119-session-17”, and commit the programs you made today.



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: brantr / Repository name: astr-119-session-5

Great repository names are short and memorable. Need inspiration? How about **fantastic-spork**.

Description (optional)

We learned a new trick! -- Jupyter notebooks.

[✓ **Public**] Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.