Exercise No. 10

David Bubeck, Pascal Becht, Patrick Nisblè

June 28, 2017

2 - Probability distribution functions

We have given a probability distribution function $p(x)$ in the domain $[0, a)$ by

$$p(x) = bx \tag{1}$$

Were b is the normalization factor. At first we need to calculate b as a fuction of a to normalize the given pdf. Therefore we integrate the pdf from 0 to a which has to be one.

$$\int_0^a bx\, dx \stackrel{!}{=} 1$$

For the solution we obtain

$$b = \frac{2}{a^2}$$

Our normalized pdf is now

$$p(x) = \frac{2}{a^2}x$$

For the calculation we used a random set of numbers which are uniformly distributed between 0 and 1. With the rejection method we calculated random numbers which obeyed equation 2.1 for $a = 0.5$. So we generated another set of random numbers uniformly distibuted as before to check if

$$v \leq \frac{p(x)}{g(x)}$$

with

$$g(x) = \frac{2}{a}$$

because $p(x)$ reaches it's maximum at $x = a$

```python
import numpy as np
import numpy.random as nprandom
import matplotlib.pyplot as plt

#maximal point for plotting invervall as given in the sheet
a = 0.5

#define linspace for the given intervall
x = np.linspace(0.001, a, 100)
#define function with normalization factor, therfore b = 2 / a^2
f = lambda x: (2 / a**2) * x
fx = f(x)

M = 2 / a                           #scale factor for rejection sampling g(x)
u1 = nprandom.rand(10000) * a       #uniform random samples scaled out in [0, 1)
u2 = nprandom.rand(10000)           #uniform random samples in [0, 1)
idx = np.where(u2 <= f(u1) / M)[0]  #rejection criterion
v = u1[idx]

#plot histogram
fig, ax = plt.subplots()
ax.hist(v, normed = 1, bins = 40, alpha = 0.7)
ax.plot(x, fx, 'r')
ax.set_title('estimated efficency = %3.1f%%'%(100 * len(v) / len(u1)))
plt.savefig('pdf.png')

#plot accepted and rejected areas
fig, ax = plt.subplots()
ax.plot(u1, u2, '.', c = 'r', label = 'rejected', alpha = 0.2)
ax.plot(u1[idx], u2[idx], '.', c = 'g', label = 'accepted', alpha = 0.4)
ax.legend(fontsize = 14)
plt.savefig('accepted_rejected.png')

plt.show()
```

Here are the plots for the solutions given with the code above. In figure 1 is show the histogram with the random number sample and the overplotting probability distribution function. In figure 2 is shown the individual random numbers which were accepted or rejected by the given algorithm. Also for this graph it was chosen a sampling of 10000 which gave us a reasonable fit (by eye) as you can see.

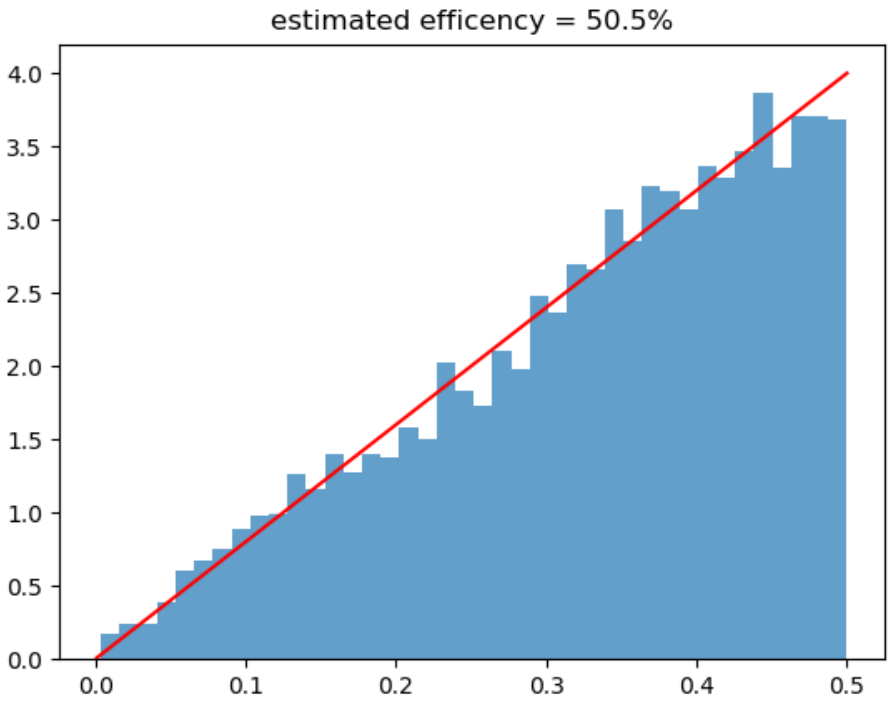

Figure 1: histogram for the pdf with overploting equation 2.1

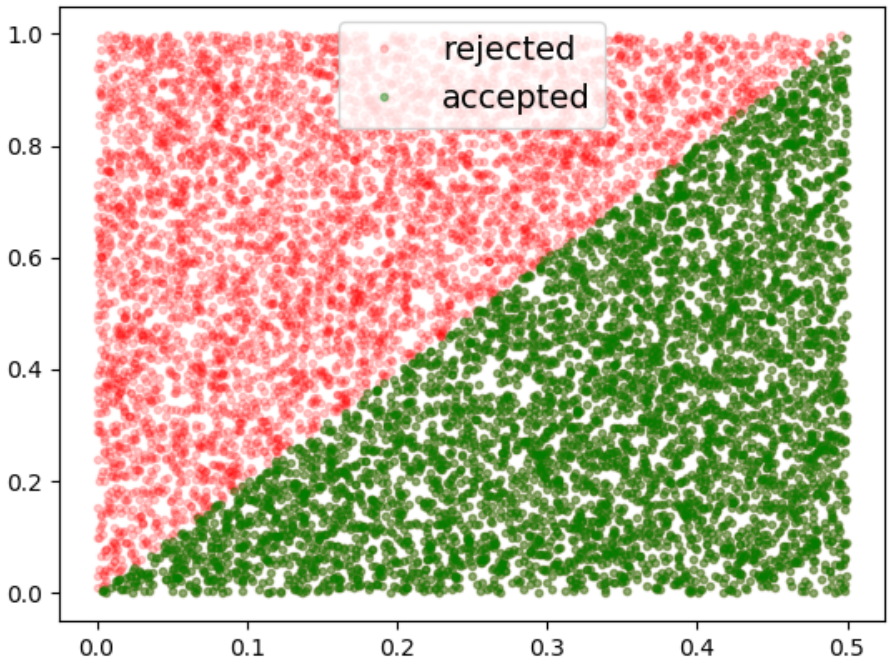

Figure 2: accepted and rejected random numbers

3 - Determine π with random numbers

To compute the number π we use the function

$$f(x) = \sqrt{1 - x^2} \quad for \quad 0 \leq x \leq 1 \tag{2}$$

Also we using the rejection method as above with a slightly adjustment for the further calculation of the accuracy of the result as a function of the random number. Furthermore, due to symmetry we only need to sample from the upper right corner of the box (one quarter of the circle). In this case the fomula to determine π can be explicitly written as:

$$\frac{pointsInCircle}{totalPoints} = \frac{\pi \cdot L^2}{4 \cdot L^2}$$

So π can be approximated with

$$\pi = \frac{pointsInCircle}{totalPoints} \cdot 4$$

Like above you can see in figure 3 histogram of the random number sample which follows the given function. Figure 4 show the accepted and rejected values. Also we have an approximation of π for this sample of $\pi_{approximation} = 3.14404$. This calculation was made with a random number sample of 100000.
The graph for the error calculation is shown in figure 5. You can see that the accuracy of the result will be better with higher random number sample.

```python
import numpy as np
import numpy.random as nprandom
import matplotlib.pyplot as plt

#define linspace for the given intervall
x = np.linspace(0.001, 1, 100)
#define function for pi calculation
f = lambda x: np.sqrt(1 - x**2)
fx = f(x)

#define fuction for rejection sampling
def approx(number_sample):
    M = 1                                   #scaling factor can be one for this
        function
    u1 = nprandom.rand(number_sample)    #uniform random samples in [0, 1)
    u2 = nprandom.rand(number_sample)    #uniform random samples in [0, 1)
    idx = np.where(u2 <= f(u1) / M)[0]   #rejection criterion
    v = u1[idx]

    #approximation of pi
    #due to symmetry we only need to sample from the upper right corner of the
        box
    #(one quarter of the circle). In this case the formula to determine pi can be
    #explicitly written as: pointsInCircle / totalPoints = pi * L^2 / (4 * L^2)
    approximation = len(v) / number_sample * 4

    return [v, idx, u1, u2, approximation]


w = approx(100000)
#plot histogram
fig, ax = plt.subplots()
ax.hist(w[0], normed = 1, bins = 40, alpha = 0.7)
ax.plot(x, fx, 'r')
ax.set_title('estimated efficency = %3.1f%%'%(100 * len(w[0]) / len(w[2])) +
                '\n approximation of pi = ' + str(w[4]))
plt.savefig('pi.png')

#plot accepted and rejected areas
fig, ax = plt.subplots()
ax.plot(w[2], w[3], '.', c = 'r', label = 'rejected', alpha = 0.2)
ax.plot(w[2][w[1]], w[3][w[1]], '.', c = 'g', label = 'accepted', alpha = 0.4)
ax.legend(fontsize = 14)
plt.savefig('accepted_rejected_pi.png')

#error calculation of the approximated pi to the pi of numpy
#in dependence of the magnitude of random numbers
sample_numbers = np.logspace(1, 6, num = 100)
pi_approximations = [approx(int(i))[4] for i in sample_numbers]
error = [(np.abs(i - np.pi) / np.pi) for i in pi_approximations]
fig, ax = plt.subplots()
ax.semilogx(sample_numbers, error)
ax.set_xlabel('magnitude random number')
ax.set_ylabel('error')
plt.savefig('error_calculation_pi.png')

plt.show()
```

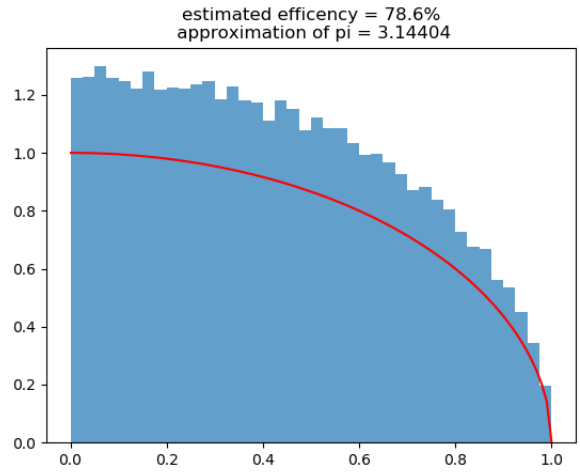

Figure 3: histogram for the function 2

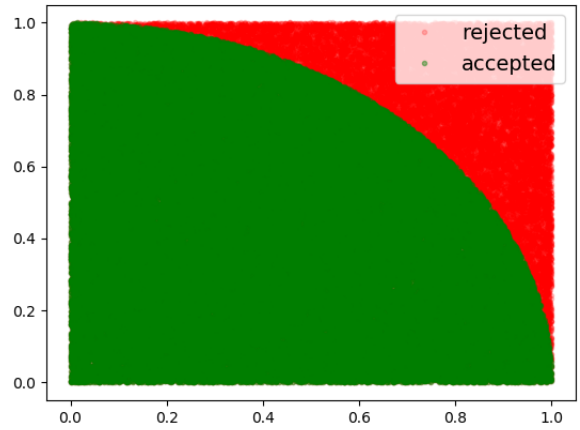

Figure 4: accepted and rejected random numbers

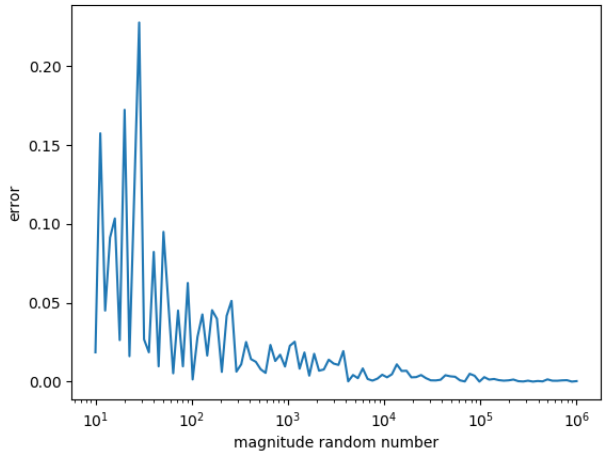

Figure 5: accuracy of the result as function of the number of samples