Computational physics

Jona Ackerschott, Julian Mayr

Problem set 9

Problem 1

A linear congruential generator (lcg) for pseudo-random numbers is implemented in source code 1. By using the values $a = 106$, $c = 1283$ and $m = 6075$ with the initial states $I_0 = 110$, $J_0 = 883$ one can check the behavior of this lcg 'by eye'. The idea is to generate (in this case 100) x, y value pairs between 0 and 1, so that the result can be plotted as a distribution of points. This is done in source code 2 with the lcg defined in source code 1 and with the python method `random`, defined in the module `random`, which generates a pseudo-random real number between 0 and 1. The results for both methods are given in figure 1.

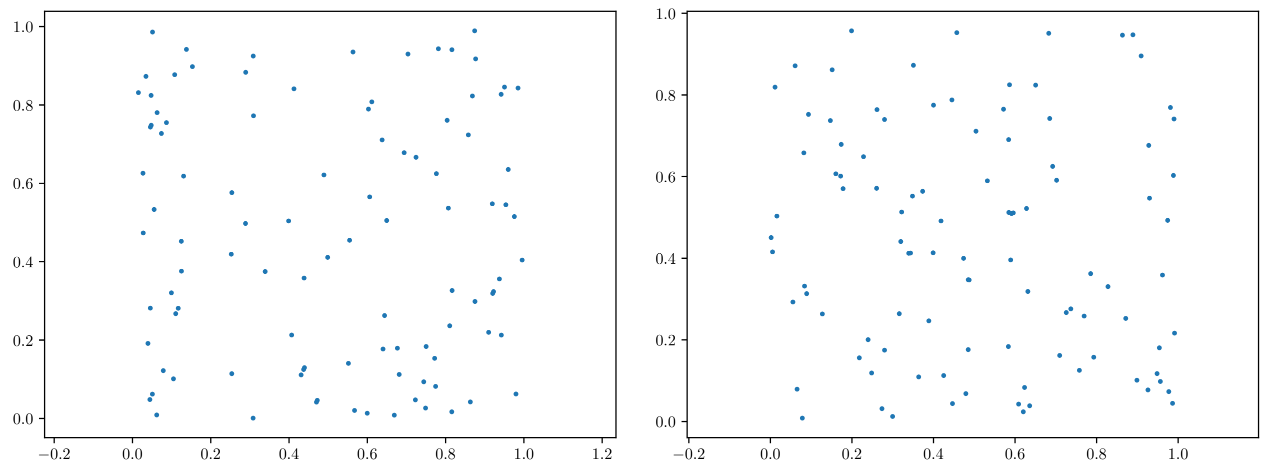

Figure 1: Distribution of 100 points, which x, y values (between 0 and 1) are generated by the lcg generator defined in source code 1 (left) and by the python method `random.random` (right)

Furthermore with an initial state of $I_0 = 110$, the I_{j+1},I_j-dependence of both random number generators is plotted by source code 3 and shown in figure 2
The expected deterministic character of this plot is not visible.
Lastly the lcg is used to simulate a dice roll. In source code 4 random numbers between 1 and 6 (or rather 0 and 5) are generated 10 times. Then, for 10,000 experiments, the sum of each random numbers is computed and plotted in a histogram. The result is given by figure 3
As one can see, the resulting distribution is approximately given by a gaussian distribution. At this point, it is somewhat odd, that there nearly half of the results are systematically missing. This also happens, by using the `random.random` method and i did not found an explanation for that, just yet.

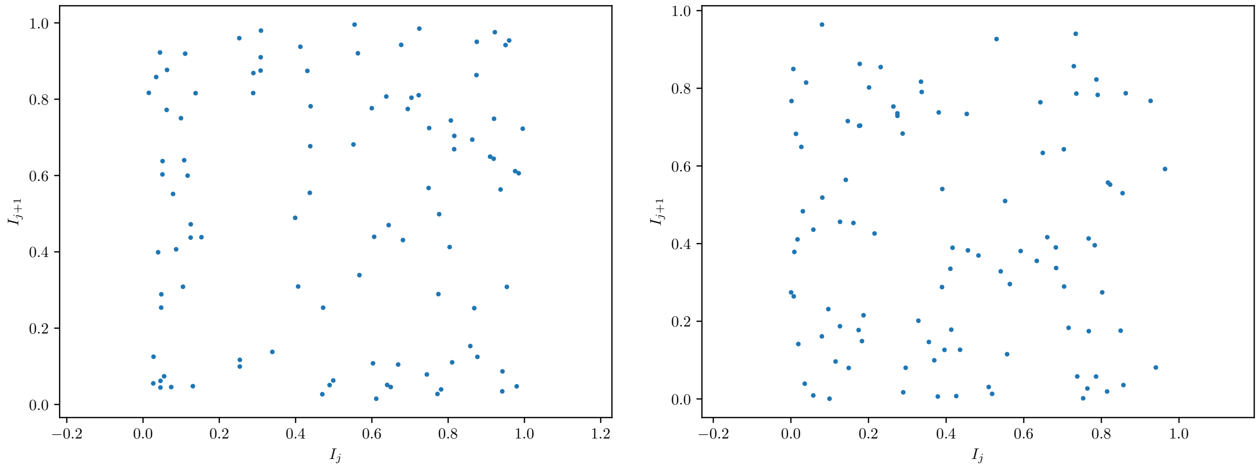

Figure 2: I_{j+1} in dependence of I_j

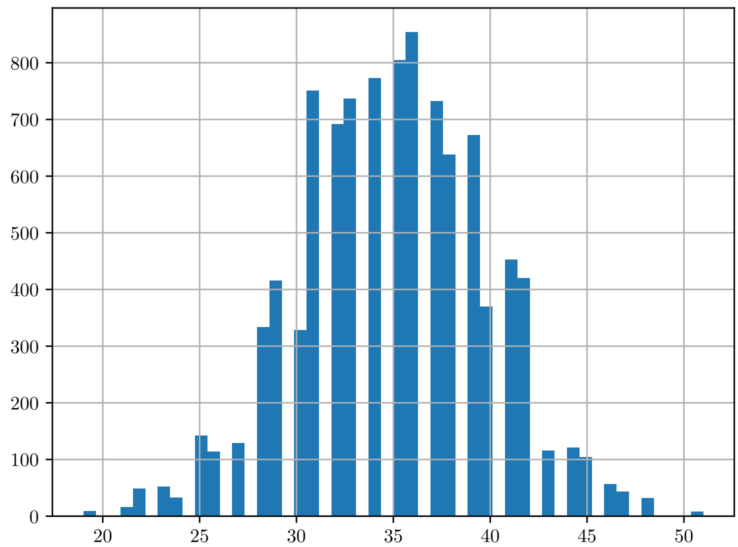

Figure 3: Sum of the results of 10 simulated dice rolls, for 10,000 total experiments. Plotted are the frequencies of each possible result.

```
class rng:
  def __init__(self, seed, a, c, m):
    self.__I = seed
    self.__a = a
    self.__c = c
    self.__m = m

  def next_int(self):
    self.__I = (self.__a * self.__I + self.__c) % self.__m
    return self.__I
```

Source code 1: rng.py

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

```python
import random

from rng import rng

figFolderPath = os.path.join(
  os.path.dirname(os.path.realpath(__file__)), 'figures'
)
if not os.path.exists(figFolderPath):
  os.makedirs(figFolderPath)

a, c, m = 106, 1283, 6075
I0, J0 = 110, 883

rng1 = rng(I0, a, c, m)
rng2 = rng(J0, a, c, m)

N = 100
x = np.array([rng1.next_int() / (m - 1) for i in range(N)])
y = np.array([rng2.next_int() / (m - 1) for i in range(N)])

x_ = np.array([random.random() for i in range(N)])
y_ = np.array([random.random() for i in range(N)])

plt.subplots()
plt.grid(False)
plt.axis('equal')
plt.scatter(x, y)

plt.savefig(os.path.join(figFolderPath, 'fig11.pdf'),
  bbox_inches='tight', pad_inches=0.6)
plt.savefig(os.path.join(figFolderPath, 'fig11.pgf'),
  bbox_inches='tight')

plt.subplots()
plt.grid(False)
plt.axis('equal')
plt.scatter(x_, y_)

plt.savefig(os.path.join(figFolderPath, 'fig12.pdf'),
  bbox_inches='tight', pad_inches=0.6)
plt.savefig(os.path.join(figFolderPath, 'fig12.pgf'),
  bbox_inches='tight')
```

Source code 2: problem1_1.py

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

```python
from rng import rng

figFolderPath = os.path.join(
  os.path.dirname(os.path.realpath(__file__)), 'figures'
)
if not os.path.exists(figFolderPath):
  os.makedirs(figFolderPath)

a, c, m = 106, 1283, 6075
I0 = 110

r = rng(I0, a, c, m)

N = 100
I = np.array([r.next_int() / (m - 1) for i in range(N)])
I_ = np.array([random.random() for i in range(N)])

plt.subplots()
plt.grid(False)
plt.axis('equal')
plt.xlabel(r'$I_j$')
plt.ylabel(r'$I_{j+1}$')
plt.scatter(I[:-1], I[1:])

plt.savefig(os.path.join(figFolderPath, 'fig13.pdf'),
  bbox_inches='tight', pad_inches=0.6)
plt.savefig(os.path.join(figFolderPath, 'fig13.pgf'),
  bbox_inches='tight')

plt.subplots()
plt.grid(False)
plt.axis('equal')
plt.xlabel(r'$I_j$')
plt.ylabel(r'$I_{j+1}$')
plt.scatter(I_[:-1], I_[1:])

plt.savefig(os.path.join(figFolderPath, 'fig14.pdf'),
  bbox_inches='tight', pad_inches=0.6)
plt.savefig(os.path.join(figFolderPath, 'fig14.pgf'),
  bbox_inches='tight')
```

Source code 3: problem1_2.py

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

from rng import rng
```

```
figFolderPath = os.path.join(
  os.path.dirname(os.path.realpath(__file__)), 'figures'
)
if not os.path.exists(figFolderPath):
  os.makedirs(figFolderPath)

a, c, m = 106, 1283, 6075
I0 = 329

r = rng(I0, a, c, m)

N = 10000
n = np.zeros(N)
for i in range(N):
  n[i] = np.sum([(r.next_int() % 6) + 1 for i in range(10)])

plt.hist(n, bins=50)

plt.savefig(os.path.join(figFolderPath, 'fig15.pdf'),
  bbox_inches='tight', pad_inches=0.6)
plt.savefig(os.path.join(figFolderPath, 'fig15.pgf'),
  bbox_inches='tight')
```

Source code 4: problem1_3.py

Problem 2

The Area A under $p(x)$ is given by

$$A = \int_0^a p(x)\,dx = \frac{a^2 b}{2} \tag{1}$$

Under the condition $A = 1$, one obtains $b = \frac{2}{a^2}$. The maximum of this distribution is $p_{\max} = \frac{2}{a}$. With this in mind, one can compute pseudo-random numbers which are distributed accordingly to $p(x)$, using the rejection method. This is done in soruce code 5. The uniform distributed random numbers needed for this are generated by the python method `random.random`. The result is given by the histogram in figure 4. One gets the first fit, which is pretty reasonable with a total number of generated samples of 1,000,000 as used below.

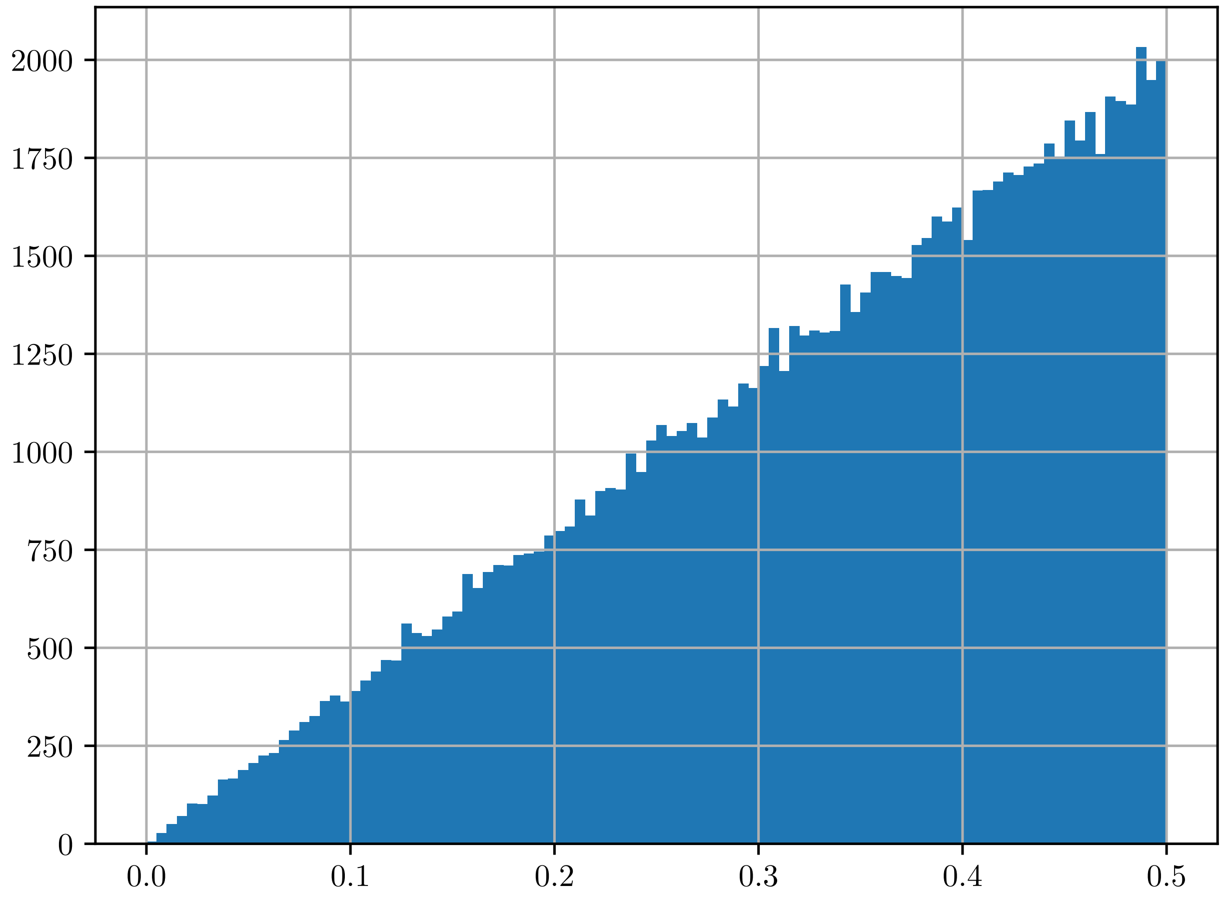

Figure 4: Histogram of 1,000,000 random numbers distributed accordingly to $p(x)$, computed using the rejection method.

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

figFolderPath = os.path.join(
  os.path.dirname(os.path.realpath(__file__)), 'figures'
)
if not os.path.exists(figFolderPath):
  os.makedirs(figFolderPath)

x_max = 0.5
y_max = 2.0 / x_max
b = 2 / x_max**2
def p(x):
  return b * x

N = 1000000
x_samples = np.zeros(N)
for i in range(N):
  while True:
    x = x_max * random.random()
    y = y_max * random.random()
    if y <= p(x):
      break
  x_samples[i] = x

plt.hist(x_samples, 100, range=(0, x_max))

plt.savefig(os.path.join(figFolderPath, 'fig2.pdf'),
  bbox_inches='tight', pad_inches=0.6)
plt.savefig(os.path.join(figFolderPath, 'fig2.pgf'),
  bbox_inches='tight')
```

Source code 5: problem2.py

Problem 3

The estimation of π is done in source code 6. The idea is to generate points within the square $[0,1] \times [0,1]$ randomly and count the number of accepted samples N_{acc}, which are under the function $f(x) = \sqrt{1-x^2}$. Then the ratio N_{acc}/N, with the total number N of the samples, is given by the ratio of the circle quadrant area $A_{\mathrm{circ}} = \pi/4$ divided by the total area of the square $A = 1$. So

$$\pi = 4\frac{N_{\mathrm{acc}}}{N} \tag{2}$$

The result of this estimation, which is the deviation of the estimations from π, is given in figure 5.

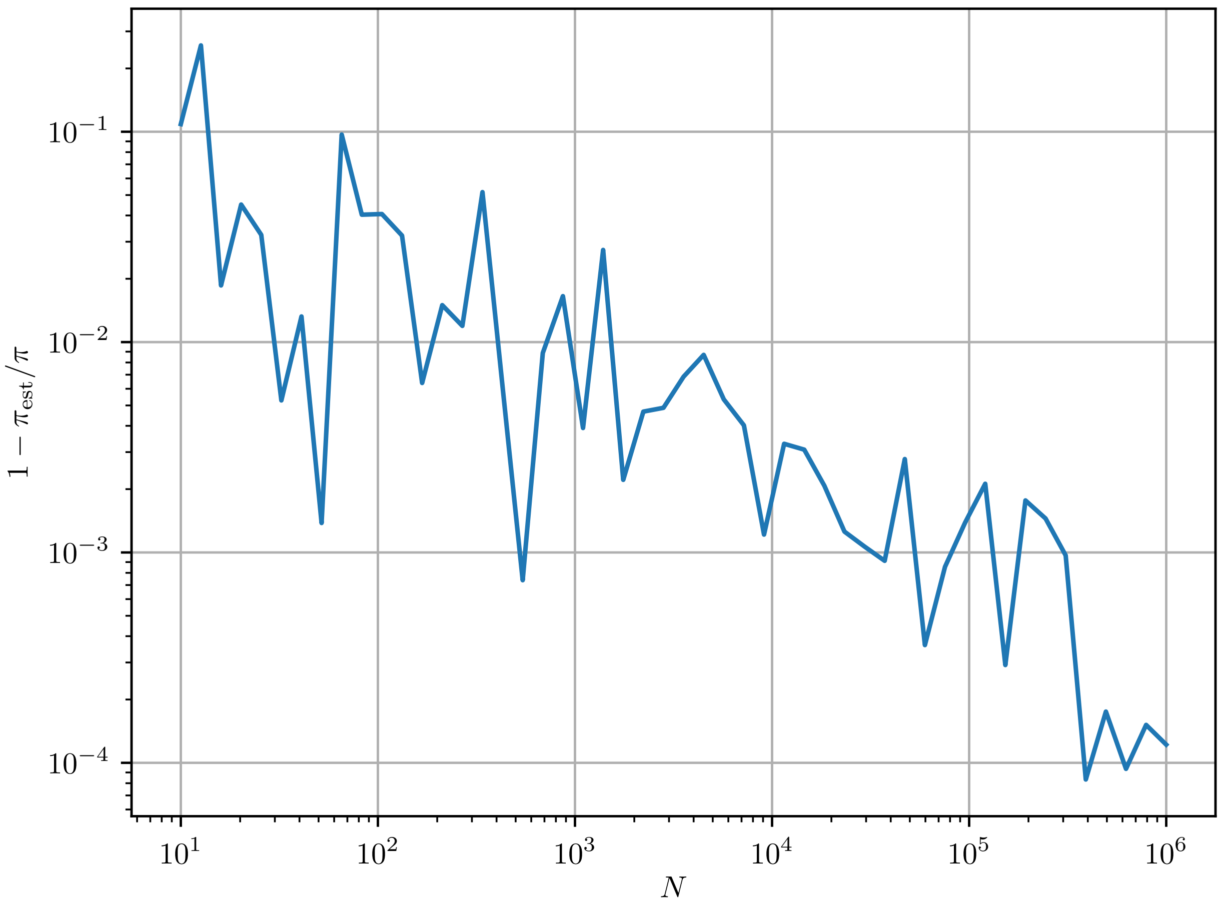

Figure 5: Deviation of the estimation π_{est} from π, in dependence of the number of samples N.

```python
import matplotlib.pyplot as plt
import numpy as np
from numpy import pi, sqrt
import os
import random

figFolderPath = os.path.join(
  os.path.dirname(os.path.realpath(__file__)), 'figures'
)
if not os.path.exists(figFolderPath):
  os.makedirs(figFolderPath)

def f(x):
  return sqrt(1 - x**2)

def pi_est(N):
  n_acc = 0
  for _ in range(N):
    x = random.random()
    y = random.random()
    if y <= f(x):
      n_acc += 1

  pi_est = 4 * n_acc / N
  return pi_est

N = np.logspace(1, 6, 50)
pi_ests = np.array([pi_est(int(N_)) for N_ in N])
err = np.array([np.abs(1 - pi_est_ / pi) for pi_est_ in pi_ests])

for pi_est_ in pi_ests:
  print(pi_est_)
print()
print(pi)

plt.xlabel(r'$N$')
plt.ylabel(r'$1 - \pi_\mathrm{est} / \pi$')
plt.loglog(N, err)

plt.savefig(os.path.join(figFolderPath, 'fig3.pdf'),
  bbox_inches='tight', pad_inches=0.6)
plt.savefig(os.path.join(figFolderPath, 'fig3.pgf'),
  bbox_inches='tight')
```

Source code 6: problem3.py