PAP334 – Exercises 3 – Model answers

Problem 1

(a) For any scalars a, m a *multiplicative linear congruential generator* (MLCG) generates a number n_i from the previous number n_i using the iterative formula:

$$n_i = (an_{i-1}) \cdot \mathrm{mod}\ m,$$

where the modulo operator gives the remainder (*jakojäännös*) of the integer division between its lhs. and rhs. components. These numbers a and m are conventionally named the *multiplier* and the *modulus*. They are defining the periodicity of the iterative procedure, hence the apparent randomness of the generator. This serie is fully defined with its initial value n_0, usually named the *seed*. This latter is an input to the random number generator, and fully determines the whole sequence to be generated.

In the particular case of the *L'Ecuyer* parameterisation they take the form $a = 40692$, $m = 2147483399$. A Python implementation of this latter is provided in the appendices. A mean value (resp. variance) of 0.502 (resp. 0.0829) are observed for a large $N = 10^4$ multiplicity. This is fairly compatible with the expected $1/2$ (resp. $1/12 \simeq 0.0833$) for a uniform distribution between 0 and 1.

In Figure 1 several monitors of this generator validity, such as the density histogram, and neighbouring values correlation are depicted. As observed, no periodic structures appear in the latter two figures, ensuring a relatively good randomness of this generator.

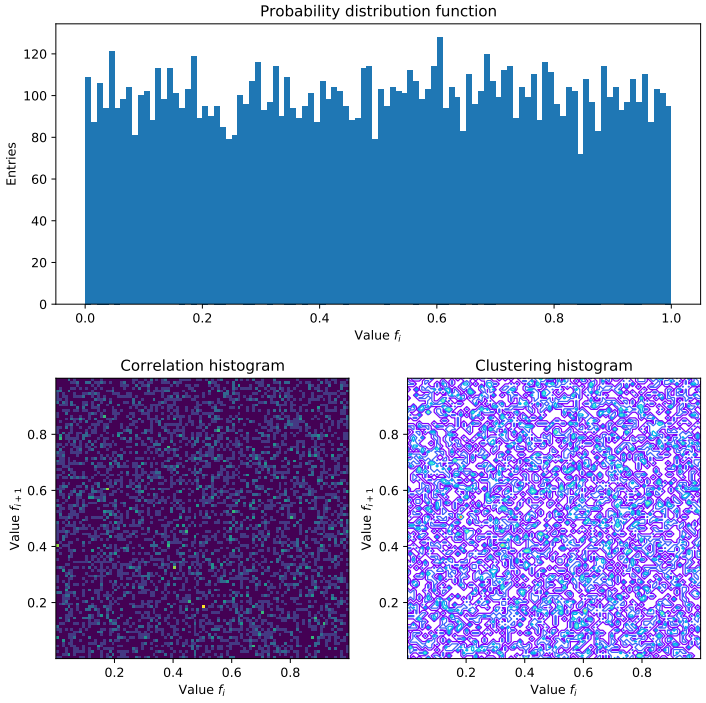

Figure 1: Top: distribution of 10^4 random numbers generated with the *L'Ecuyer* MLCG, and categorised in a 100-bin one-dimensional histogram. Bottom left (resp. right): neighbours correlation map (resp. clusters) for the same random numbers.

Several more formal tests can however be defined to ensure the validity. First would be a relatively small Pearson neighbours correlation coefficient, defined for a sample as:

$$\rho_{n,n+1} = \frac{\sum_{i=1}^{n}(x_i - \langle x_i \rangle)(x_{i+1} - \langle x_{i+1} \rangle)}{(n-1)\sigma_n \sigma_{n+1}},$$

where $\langle x_n \rangle$ and σ_n are the usual mean and variance of the full population considered.

In our particular case, this Pearson correlation coefficient is evaluated at -9.63×10^{-3}, hence a slight anti-correlation which can be safely neglected for a sufficiently low series multiplicity.

A possible other randomness test is the sample *binned flatness* ϕ defined as

$$\phi = \left\langle \frac{|f_i - \langle f \rangle|}{N} \right\rangle,$$

for a N-bins histogram with values $f_1, f_2, \ldots f_N$. As a normalised variation between the bin values and their average, it is expected to run between 0 (perfectly flat) and 1 (strongly peaked). Again in our particular implementation, this flatness is evaluated at 0.0832 (pretty flat)

(b) The central limit theorem (CLT) states that the sum of many random numbers following any distribution should follow a Gaussian. The test of the CLT resulted in Figure 2. The mean and standard deviation for this histogram is expected to be $\mu = 12 \cdot 0.5 = 6$ and $\sigma^2 = 12 \cdot (1/12) = 1$. The calculated results are $\mu = 5.990$ and $\sigma^2 = 0.9789$. Therefore the CLT seems to hold.

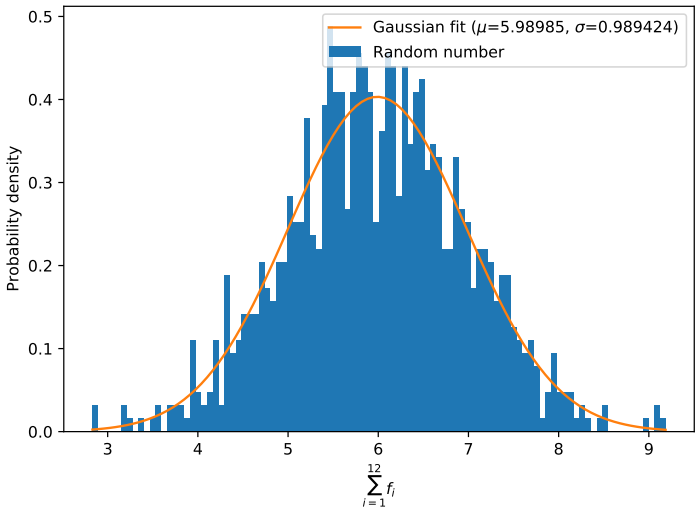

Figure 2: Distribution of the sum of 12 random numbers generated following a uniform distribution between 0 and 1.

Problem 2

The first step is to extract the normalisation constant for the Breit-Wigner distribution. This can be computed using

$$1 \equiv \frac{1}{N}\int_{-\infty}^{+\infty} \mathrm{d}E \, \frac{\Gamma_s^2}{(E-E_s)^2 + (\Gamma_s/2)^2}$$

$$N = \int_{-\infty}^{+\infty} \mathrm{d}E \, \frac{\Gamma_s^2}{(\Gamma_s/2)^2} \frac{1}{(2(E-E_s)/\Gamma_s)^2 + 1}.$$

Defining the variable change $E \mapsto x = 2(E-E_s)/\Gamma_s$, thus $\mathrm{d}x = 2\mathrm{d}E/\Gamma_s$, this gives

$$N = 2\Gamma_s \int_{-\infty}^{+\infty} \mathrm{d}x \, \frac{1}{x^2+1} = 2\Gamma_s \left[\arctan x\right]_{-\infty}^{+\infty} = 2\pi\Gamma_s.$$

Therefore the (normalised) probability density function for a Breit-Wigner distribution can be expressed as

$$f(E|E_s,\Gamma_s) = \frac{1}{\pi}\frac{\Gamma_s/2}{(E-E_s)^2+(\Gamma_s/2)^2}.$$

The inverse transform method allows to generate a given distribution using a simple uniform distribution. This technique requires to define a function $E(r)$ giving the desired PDF $f(E)$ when evaluated using r values uniformly distributed between 0 and 1.

Using this, the probability to obtain a value of r in $[r, r+\mathrm{d}r]$ is $g(r)\mathrm{d}r$. This latter should be equal to the probability to obtain E in $[E(r), E(r)+\mathrm{d}E(r)]$, which is $f(E)$. This can be obtained by finding $E(r)$ such that the cumulative distributions $F(E(r))$ and $G(r)$ relate by $F(E(r)) = G(r)$.

The cumulative distribution function (CDF) can therefore be expressed as:

$$F(E = E(r)) = \int_{-\infty}^{E(r)} \mathrm{d}E' f(E') = \int_{-\infty}^{r} \mathrm{d}r' g(r') = r.$$

Using the same change of variables as introduced above ($x = x(r)$), this can be rewritten as

$$r = \frac{1}{\pi}\int_{-\infty}^{x(r)} \mathrm{d}x' \frac{1}{x'^2+1} = \frac{1}{\pi}\left[\arctan x'\right]_{-\infty}^{x(r)} = \frac{1}{\pi}\left[\arctan\left(\frac{2(E(r)-E_s)}{\Gamma_s}\right)+\frac{\pi}{2}\right].$$

This latter allows to express $E(r)$ as

$$\left(r-\frac{1}{2}\right)\pi = \arctan\left(\frac{2(E(r)-E_s)}{\Gamma_s}\right)$$
$$E(r) = E_s + \frac{\Gamma_s}{2}\tan\left(\left(r-\frac{1}{2}\right)\pi\right).$$

This spectrum is generated for $E_s = 25$ GeV, $\Gamma_s = 5$ GeV in Figure 3. For comparison, the green curve shows the equivalent Gaussian distribution (using $\mu = E_s$, $\sigma = \Gamma_s$). Furthermore, a fit of the central part of this Breit-Wigner distribution ($17 < E < 33$ GeV) is performed using a Gaussian PDF. While the average is fully compatible with the generated E_s, the variance is seen to be smaller, compatible with the large distribution tails observed for the Breit-Wigner PDF.

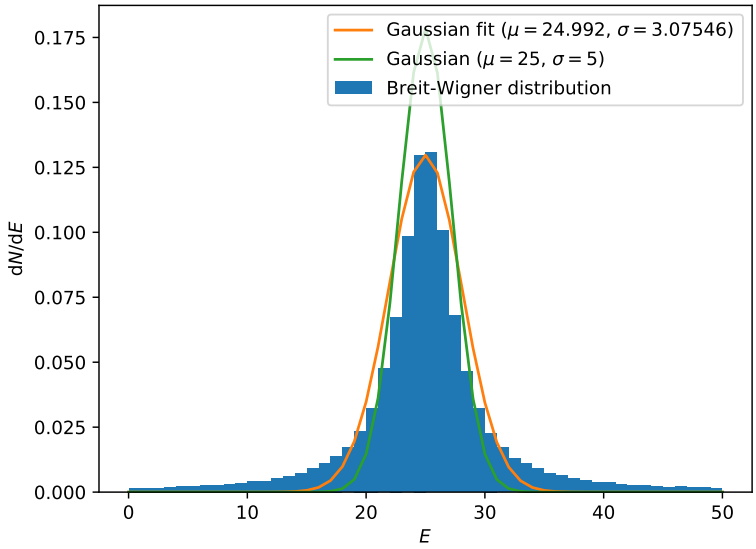

Figure 3: Breit-Wigner distribution as generated from the inverse transform method. Overlaid are the Gaussian fits of the central part of this distribution ($17 < E < 33$ GeV) and the Gaussian distribution with $\mu = E_s$, $\sigma = \Gamma_s$.

Code listing

Problem 1

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as grid
import numpy as np
from scipy import stats

# generator initialisation
class mlcg:
    def __init__(self, a=40692, m=2147483399, n=1235644):
        self._a = a
        self._m = m
        self._n = n
    def generate(self):
        self._n = (self._a*self._n) % self._m
        return self._n*1./self._m

# exercise (a)

# build a new generator object
gen = mlcg(n=10)

# will fill an array with N values
f = []
for i in range(10000):
    f.append(gen.generate())

# neighbour correlation
f1 = []
f2 = []
for i in range(int(len(f)/2)):
    f1.append(f[2*i-1])
    f2.append(f[2*i])

# a bit of printout
print('mean=', np.mean(f))
print('variance=', np.std(f)**2)
print('Pearson correl.coeff.=', np.corrcoef(f1, f2)[1,0])

# initialise the plotting part
fig = plt.figure()
ext = grid.GridSpec(ncols=1, nrows=2)

# first plot: PDF
ax_top = plt.Subplot(fig, ext[0])
counts, xbins, image = ax_top.hist(f, bins=100, label=['Random number'])
ax_top.set_title('Probability distribution function')
ax_top.set_xlabel('Value $f_i$')
ax_top.set_ylabel('Entries')
fig.add_subplot(ax_top)

mean_c = np.mean(counts)
print('flatness=', np.mean([abs(c_i-mean_c)/len(counts) for c_i in counts]))

# divide the lower part in two subfigures
bot = grid.GridSpecFromSubplotSpec(1, 2, subplot_spec=ext[1])

# second plot: neighbours correlation
ax_left = plt.Subplot(fig, bot[0])
counts, ybins, xbins, image = ax_left.hist2d(x=f1, y=f2, bins=(100, 100))
limits = [xbins.min(), xbins.max(), ybins.min(), ybins.max()]
ax_left.set_title('Correlation histogram')
ax_left.set_xlabel('Value $f_i$')
ax_left.set_ylabel('Value $f_{i+1}$')
fig.add_subplot(ax_left)

# third plot: identify clusters between neighbours
ax_right = plt.Subplot(fig, bot[1])
ax_right.contour(counts, extent=limits, linewidths=1, cmap=plt.cm.rainbow)
ax_right.set_title('Clustering histogram')
ax_right.set_xlabel('Value $f_i$')
ax_right.set_ylabel('Value $f_{i+1}$')
fig.add_subplot(ax_right)

fig.tight_layout()
fig.show()
plt.clf()

# exercise (b)
```

```
# will fill an array with N sums of 12 random numbers
sum_f = []
for i in range(1000):
    sum_f.append(np.sum([gen.generate() for j in range(12)]))

# a bit of printout
print('mean=', np.mean(sum_f))
print('variance=', np.std(sum_f)**2)

# initialise the plotting part

counts, xbins, image = plt.hist(sum_f, bins=100, density=True, label=['Random number'])
mean, sigma = stats.norm.fit(sum_f)
pdf_norm = stats.norm.pdf(xbins, mean, sigma)
plt.plot(xbins, pdf_norm, label='Gaussian fit ($\mu$=%g, $\sigma$=%g)' % (mean, sigma))
plt.xlabel('$\sum_{i=1}^{12} f_i$')
plt.ylabel('Probability density')
plt.legend(loc='best')
plt.show()
```

Problem 2

```
import matplotlib.pyplot as plt
import numpy as np
import random
from scipy import stats

# cumulative distribution function for B-W
def cdfBW(E, G_s, E_s):
    return 0.5+np.arctan(2.*(E-E_s)/G_s)/np.pi

# inverse cumulative distribution function for B-W
def cdfBWInv(x, G_s, E_s):
    return E_s+G_s/2.*np.tan((x-0.5)*np.pi)

# define distribution parameters and plot range
G_s = 5.
E_s = 25.

# compute the values of x for edges of the range of interest for E
min_E, max_E = (0., 50.)
min_x = cdfBW(min_E, G_s, E_s)
max_x = cdfBW(max_E, G_s, E_s)

# will fill an array with N values of the generated energy
E = []
for i in range(100000):
    rnd = random.random()
    # not very efficient method to define plotting range
    #E_i = cdfBWInv(rnd, G_s, E_s)
    #if E_i < 0. or E_i > 50.: continue
    # better method
    E_i = cdfBWInv(min_x+(max_x-min_x)*rnd, G_s, E_s)
    E.append(E_i)

# plotting part

counts, xbins, image = plt.hist(E, bins=50, density=True, label=['Breit-Wigner distribution'])
mean, sigma = stats.norm.fit([e for e in E if e > 17 and e < 33]) # normal (=Gaussian) fit
gaus_fit = stats.norm.pdf(xbins, mean, sigma)
gaus_pars = stats.norm.pdf(xbins, E_s, np.sqrt(G_s))
plt.plot(xbins, gaus_fit, label='Gaussian fit ($\mu={:g}$, $\sigma={:g}$)'.format(mean, sigma))
plt.plot(xbins, gaus_pars, label='Gaussian ($\mu={:g}$, $\sigma={:g}$)'.format(E_s, G_s))
plt.xlabel('$E$')
plt.ylabel('${\\rm d}N/{\\rm d}E$')
plt.legend(loc='best')
plt.show()
```