Stochastic Simulations

Autumn Semester 2024

Prof. Fabio Nobile *Assistant: Matteo Raviola*

Lab 01 – 12 September 2024

Random Number Generation

Exercise 1

Consider `Scipy` 's default **uniform** random number generator (RNG) `uniform` within the Statistics module `scipy.stats` and use it to generate a sequence of numbers $U_1, U_2, \dots, U_n$. See this for the module documentation and see this for the full list of available continuous distributions under `scipy.stats` . Consider different values of n, for example $n = 25, 100, 10^3, 10^5$, and address the following points:

1. Plot the cumulative distribution function (CDF) of the theorized $\mathcal{U}(0,1)$ distribution together with the empirical CDF of the data. Furthermore, produce a Q-Q plot of the data. Use both plots to assess the quality of the sequence with respect to the theorized $\mathcal{U}(0,1)$ distribution. Describe your observations.

2. Implement the Kolmogorov–Smirnov test to ascertain whether the empirical CDF of $U_1, U_2, \dots, U_n$ matches the theoretical CDF of the $\mathcal{U}(0,1)$ distribution at level $\alpha = 0.1$, i.e., we reject the null hypothesis H_0 at level $\alpha > 0$ that the sample $U_1, \dots, U_n \overset{\text{iid}}{\sim} \mathcal{U}(0,1)$ if $\sqrt{n}D_n > K_{\alpha,n}$, where $D_n = \sup_{x\in\mathbb{R}}|\hat{F}(x) - F(x)|$, and $K_{\alpha,n}$ is such that $\mathbb{P}(\sqrt{n}D_n > K_{\alpha,n}) < \alpha$.

   It is known that the appropriately scaled test statistic $D_n = \sup_{x\in\mathbb{R}}\left|\hat{F}_n(x) - F(x)\right|$ converges in distribution to a Kolmogorov random variable K independently of F, where $\mathbb{P}(K \le x) = 1 + 2\sum_{j=1}^{\infty}(-1)^j e^{-2j^2x^2}$, $x > 0$. This asymptotic result can then be used to compute the required $1-\alpha$ quantiles, $K_{\alpha,n} \simeq K_{\alpha,\infty}$. It is however also possible to characterize the distribution of D_n directly, which is useful for small values of n. Table 1 presents some of these pre-asymptotic $1-\alpha$ quantiles $K_{\alpha,n}$.

3. Implement the χ^2 goodness of fit test to ascertain whether the sequence $U_1, U_2, \dots, U_n$ is equidistributed. A description of such method can be found on section 8.7.4 of *Handbook of Monte Carlo Methods* (See also page 10 of the lecture notes).

   Hint: you can use the `ppf` *function of the* `scipy.stats.chi2` *class to compute quantiles of a* χ^2 *distribution.*

4. Repeat the tests in points 2. and 3. for different values of α. What do you observe? Explain your findings.

	α			
n	0.20	0.10	0.05	0.01
1	0.90	0.95	0.98	0.99
2	0.96	1.10	1.19	1.32
3	0.97	1.11	1.23	1.44
4	0.98	1.12	1.24	1.46
5	1.01	1.14	1.25	1.50
6	1.00	1.15	1.27	1.52
7	1.01	1.16	1.30	1.53
8	1.02	1.16	1.30	1.53
9	1.02	1.17	1.29	1.53
10	1.01	1.17	1.30	1.55
11	1.03	1.16	1.29	1.56
12	1.04	1.18	1.32	1.56
15	1.05	1.16	1.32	1.55
20	1.03	1.16	1.30	1.57
30	1.04	1.20	1.31	1.59
35	1.06	1.24	1.36	1.60
40	1.08	1.20	1.33	1.58
45	1.07	1.21	1.34	1.61
$n > 45$	1.07	1.22	1.36	1.63

Table 1: Critical values for the Kolmogorov–Smirnov test statistic $D_n = \sup_{x\in\mathbb{R}} \left|\hat{F}_n(x) - F(x)\right|$. The tabulated values $K_{n,\alpha}$ are such that $\mathbb{P}(\sqrt{n}D_n \geq K_{n,\alpha}) = \alpha$.

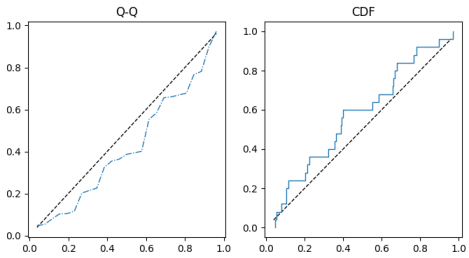
Solution

Implementation details concerning the tasks in this exercise are presented at the end of this Section. There we mention solutions without using built-in functions as well as indicate possible solutions based on built-in functions.

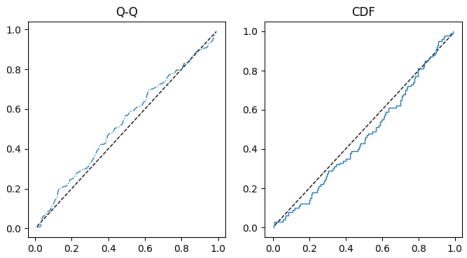
1.–3. Figure 1 illustrates the CDF comparisons and Q-Q plots for a `rand` sequence (seed is fixed for reproducibility; c.f. code). We see that increasing n results in a better linear behavior in the Q-Q plots as well as a better fit of the empirical CDF and the theorized CDF.

Furthermore, the test outcomes ($\alpha = 0.1$) for the same sequences are shown in Table 2, suggesting that `scipy.stats` 's default RNG produces a sequence with the desired uniform distribution indeed. Note that for some cases, the χ^2-test can reject the null hypothesis for 10^4. This is somehow an artifact of the selected seed.

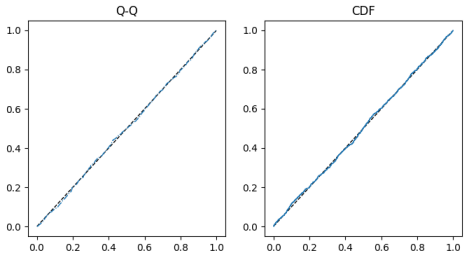
4. The smaller the values of α, the higher the confidence with which we require the test to hold. To account for an increased level of confidence, the value of a test statistic has to be even bigger before a test rejects the null hypothesis. This can be observed using the code in Section 1.



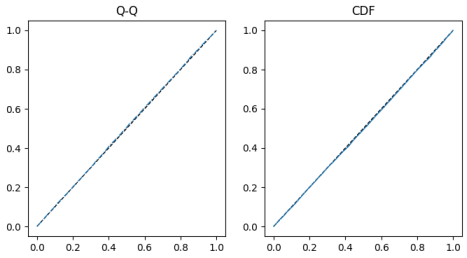
(a) $n = 25$



(b) $n = 100$



(c) $n = 10^3$



(d) $n = 10^4$

Figure 1: CDF (right) and Q-Q (left) plots for various values of n, based on sequences generated by `uniform.rvs()` (Exercise 1).

	n				
Test	25	100	1000	10000	100000
KS	not rejected	not rejected	not rejected	not rejected	not rejected
χ^2	not rejected	not rejected	not rejected	rejected	not rejected

Table 2: Test outcomes for `uniform.rvs()` at level $\alpha = 0.1$.

Exercise 2

Implement the linear congruential generator (LCG)

$$X_k = (aX_{k-1} + b) \bmod m \,, \quad U_k := \frac{X_k}{m} \,,$$

with $a = 3$, $b = 0$, and $m = 31$.

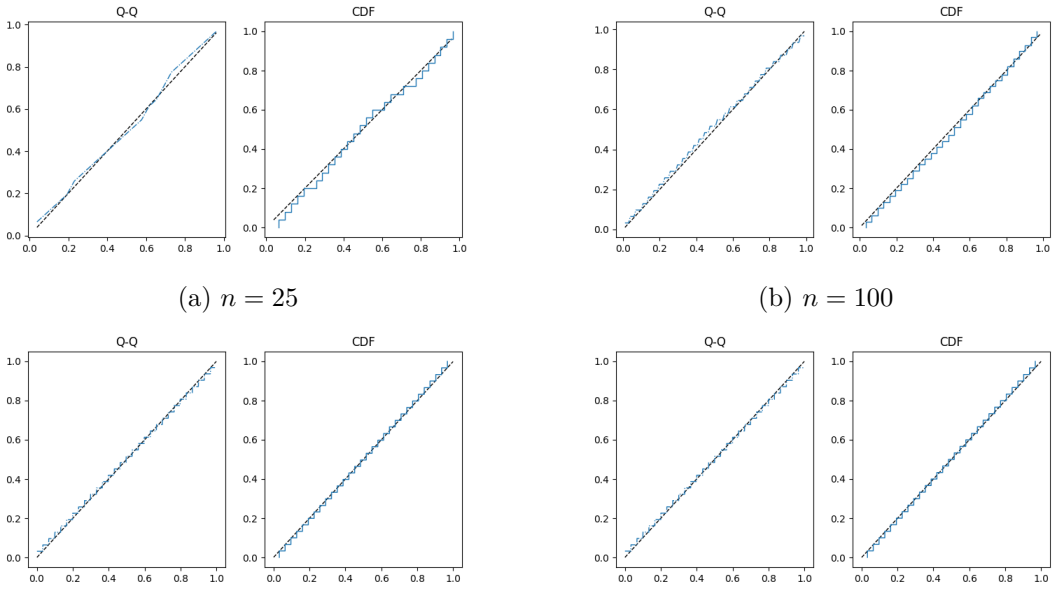
1. Use your LCG procedure to generate a sequence $U_1, U_2, \ldots, U_n$ and repeat Exercise 1. Discuss your results.

2. Explain why one would expect that the Serial test (with $d = 2$, say) is an appropriate test to scrutinize the LCG. Support your explanation by applying the Serial test at level $\alpha = 0.1$ to sequences (for various values of n) from both the LCG and from the default `scipy.stats` RNG `uniform`.

3. Implement the Gap test. Apply the test to both a sequence obtained from the default `scipy.stats` RNG `uniform` and to a sequence generated by the LCG. What do you observe?

Solution

1. Figure 2 illustrates the CDF comparisons and Q-Q plots for LCG sequences (seed `X0` is equal to 1). The test outcomes ($\alpha = 0.1$) for the same sequences are shown in Table 3. We see that the KS-test rejects the null hypothesis of a uniform distribution for large



(a) $n = 25$ (b) $n = 100$

(c) $n = 10^3$ (d) $n = 10^4$

Figure 2: CDF (right) and Q-Q plots for various values of n, based on sequences generated by LCG (Exercise 2).

	n				
Test	25	100	1000	10000	100000
KS	not rejected	not rejected	not rejected	rejected	rejected
χ^2	not rejected	not rejected	not rejected	not rejected	not rejected

Table 3: Test outcomes for the LCG at level $\alpha = 0.1$.

values of n. This is a consequence of the non-vanishing residual of $|\hat{F} - F|$ due to the periodicity of the LCG sequence ($m = 31$ is small). Notice that the χ^2-test with number of bins $K = 10$ does not reject the null hypothesis even for large values of n. This is due to the fact that K is small compared to the length of the sequence n. Increasing the value of K (e.g. $K = 20$) for large values of n changes this (see also the discussion on the value of r below).

2. The reason why we expect the Serial test to perform very well in rejecting the LCG sequence is the LCG's periodicity. Specifically, Figure 3 shows plots of the tuple (U_i, U_{i+1}) for both a `uniform.rvs` sequence (left) and a LCG sequence, each of length $n = 1000$. While we cannot recognize any systematic pattern in the plot for the `scipy.stats`

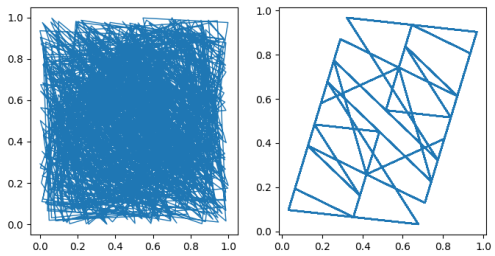


Figure 3: Plots of the tuple (U_i, U_{i+1}) for both a `uniform.rvs()` sequence (left) and a LCG sequence, each of length $n = 1000$.

generated numbers, the plot for the LCG shows a systematic pattern. As a consequence the full state space is not explored by the sequence. The serial test is, however, exactly designed for ascertain a uniform coverage of the state space, which suggests that the test will reject the LCG sequences. The implementation of the Serial test is simple extension of the χ^2-test used in the previous Exercise.

Here the serial test is implemented with $m = 10$ subdivisions along each dimension. The numerical tests confirm the motivation that this test is effective to identify the LCG sequence as not coming from a uniform random variable. In fact, the test rejects the null hypothesis at level $\alpha = 0.1$ already for $n \geq 100$; see Table 4. A rule-of-thumb

	n				
Test	26	100	1000	10000	100000
Serial	not rejected	rejected	rejected	rejected	rejected

Table 4: Serial test outcomes for the LCG at level $\alpha = 0.1$.

in practice is to use n and m such that $n \geq 5m^d$. This is satisfied here for $n \geq 1000$ ($m = 10$ and $d = 2$).

3. Here the Gap-test is implemented with $r = 5$ and the test outcomes are shown in Table 5. We observe that the Gap-test rejects the null hypothesis at the level $\alpha = 0.1$ for $n \geq 100$ of the LCG sequence, while the null hypothesis for the `uniform.rvs()` sequence was never rejected. Notice however that the smallest expected number per class is not bigger than 5 (which is the recommended threshold in practice) for approximately $n < 350$, so that the outcomes for smaller values of n may not be reliable.

	n			
RNG	25	100	1000	10000
`rand`	not rejected	not rejected	not rejected	not rejected
LCG	not rejected	rejected	rejected	rejected

Table 5: Gap-test for `uniform.rvs()` and LCG sequences of length n.

1 Python code

The Python code below exemplifies the implementation of the exercises of this lab. There the function getCriticalValueKS(n,alpha) computes (possibly via an interpolation step) the critical values needed for the KS test, which are based on the Table given in the exercise description.

1.1 getCriticalValuesKS.py script

```
import numpy as np
import math
from scipy import interpolate as intrp

def getCriticalValuesKS(n, alpha):
        #GETCRITICALVALUEKS compute the critical value for the two-sided KS test
        #statistic

        # References:
        #   Massey, F.J., (1951) "The Kolmogorov-Smirnov Test for Goodness of Fit",
        #         Journal of the American Statistical Association, 46(253):68-78.
        #   Miller, L.H., (1956) "Table of Percentage Points of Kolmogorov Statistics",
        #         Journal of the American Statistical Association, 51(273):111-121.
        #   Marsaglia, G., W.W. Tsang, and J. Wang (2003), "Evaluating Kolmogorov`s
        #         Distribution", Journal of Statistical Software, vol. 8, issue 18.

        # The critical value table used below is expressed in reference to a
        # 1-sided significance level.  Need to halve the significance level for
        # a basic two-sided test.

        alpha1 = alpha / 2.

        if n <= 20:  # Small sample exact values.
                # Exact K-S test critical values are solutions of an nth order polynomial.
                # Miller's approximation is excellent for sample sizes n > 20. For n <= 20,
                # Miller tabularized the exact critical values by solving the nth order
                # polynomial. These exact values for n <= 20 are hard-coded into the matrix
                # 'exact' shown below. Rows 1:20 correspond to sample sizes n = 1:20.
                a1 = np.array([0.00500,  0.01000,  0.02500,  0.05000,  0.10000])  # 1-sided significance level

                exact = np.array([[0.99500,  0.99000,  0.97500,  0.95000,  0.90000],
                                  [0.92929,  0.90000,  0.84189,  0.77639,  0.68377],
                                  [0.82900,  0.78456,  0.70760,  0.63604,  0.56481],
                                  [0.73424,  0.68887,  0.62394,  0.56522,  0.49265],
                                  [0.66853,  0.62718,  0.56328,  0.50945,  0.44698],
                                  [0.61661,  0.57741,  0.51926,  0.46799,  0.41037],
                                  [0.57581,  0.53844,  0.48342,  0.43607,  0.38148],
                                  [0.54179,  0.50654,  0.45427,  0.40962,  0.35831],
                                  [0.51332,  0.47960,  0.43001,  0.38746,  0.33910],
                                      [0.48893,  0.45662,  0.40925,  0.36866,  0.32260],
                                    [0.46770,  0.43670,  0.39122,  0.35242,  0.30829],
                                          [0.44905,  0.41918,  0.37543,  0.33815,  0.29577],
                                  [0.43247,  0.40362,  0.36143,  0.32549,  0.28470],
                                  [0.41762,  0.38970,  0.34890,  0.31417,  0.27481],
                                  [0.40420,  0.37713,  0.33760,  0.30397,  0.26588],
                                         [0.39201,  0.36571,  0.32733,  0.29472,  0.25778],
                                  [0.38086,  0.35528,  0.31796,  0.28627,  0.25039],
```

```
                                        [0.37062,  0.34569,  0.30936,  0.27851,  0.24360],
                                        [0.36117,  0.33685,  0.30143,  0.27136,  0.23735],
                                        [0.35241,  0.32866,  0.29408,  0.26473,  0.23156]])

                criticalValue  =  intrp.interp1d(a1 , exact[n-1,:], kind = 'cubic')(alpha1)
        else: # Large sample approximate values.
                #  alpha is a 1-sided significance level
                A = 0.09037 * (-math.log(alpha1, 10))**1.5 + 0.01515 * math.log(alpha1,10)**2 - 0.08467 * alpha1
                asymptoticStat =  np.sqrt(-0.5*np.log(alpha1)/n)
                criticalValue  =  asymptoticStat - 0.16693 / n - A / n**1.5

                criticalValue  =  np.min([criticalValue, 1-alpha1])

        return criticalValue
```

1.2 lab01.py script

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
from getCriticalValuesKS import *

#np.random.seed(12345) # Fix seed to allow rerpoducibility

def LCG(X0 = 1, n = 100, a = 3, b = 0, m = 31):
    """
    Linear Congruential Generator
    """
    X = np.zeros(n+1)
    X[0] = X0
    for i in range(1,n+1):
            X[i] = (a * X[i-1] + b) % m

    U = X / m
    return U

def KSTest(X, alpha = 0.1):
    """
    Kolmogorov Smirnov Test for data X and significance alpha
    """
    xgrid = np.linspace(0, 1, 10001)
    Fhat = [(X<=x0).sum()/float(n) for x0 in xgrid]
    Dn = np.max(np.abs(Fhat - F(xgrid)))
    #[Dn1, p] = st.kstest(X, F) # Comparison with the scipy.stats' builtin function
    print('KS Test statistic: ' + str(Dn) )

    val = getCriticalValuesKS(n, alpha)
    true = int( (Dn > val) ) # Convert boolean to int
    rej = ['cannot be', 'is']
    stat = {'Statistic': Dn, 'Quantile': val, 'Significance': alpha}
    message = 'KS test: the null hypothesis H0 ' + rej[true] + ' rejected at level alpha = ' + str(alpha)
    return message, stat

def ChiSquareTest(X, K = 10, alpha = 0.1):
    """
    Chi Squared Test for data X and significance alpha with K degrees of freedom
```

```
    """
    n = len(X)
    p = np.ones(K) / K
    N = np.array([np.sum((float(i)/K < X) & (X <= (i+1.)/K)) for i in range(K)])
    QK = np.sum( (N-n*p)**2. / (n*p)) # test statistic
    critval = st.chi2.ppf(1-alpha, K-1)
    true_chi = 1 * (QK > critval)
    rej = ['cannot be', 'is']
    stat = {'Statistic': QK, 'Quantile': critval, 'Significance': alpha}
    message = 'Chi2 test: the null hypothesis H0 ' + rej[true_chi] + ' rejected at level alpha = ' + str(alpha)
    return message, stat


def SerialTest(X, d = 2, alpha = 0.1):
    """
    Serial statistical test
    """
    assert X.shape[0] % 2 == 0, 'Random sample length should be even.'
    Y = X.reshape(2, int(X.shape[0]/2)).T
    nY = Y.shape[0]
    m = 10
    K = m**d
    Nm = np.zeros(K)
    p = np.ones(K) / float(K) # True probabilities (uniform partition)
    for k in range(K):
            xl = (k % m) / float(m)
            xu = xl + 1./m
            yl = np.max([0., np.floor(k/float(m))]) / m
            yu = yl + 1./m
            Nm[k] = np.sum( ((xl< Y[:,0]) & (Y[:,0] <= xu)) * ( (yl<Y[:,1]) & (Y[:,1]<= yu)))

    rej = ['cannot be', 'is']
    Qm = np.sum( (Nm - nY*p)**2 / (nY*p))
    serval = st.chi2.ppf(1-alpha, K-1)
    true_ser = 1 * (Qm > serval)
    stat = {'Statistic' : Qm, 'Quantile': serval, 'Significance': alpha}
    message = 'Serial test: the null hypothesis H0 ' + rej[true_ser] + ' rejected at level alpha = ' + str(alpha)
    return message, stat

def GapTest(X, alpha = 0.1, aa=0., bb=0.5, r=5):
    """
    Gap test
    """
    idx = list(set(np.where(X > aa)[0]) & set(np.where(X < bb)[0]))
    idx = np.hstack([0, np.array(idx) + 1])
    Z = idx[1:] - idx[:-1] - 1
    prob = bb - aa
    p = prob * (1 - prob) ** np.array([i for i in range(r)])
    p = np.hstack([p, (1 - prob)**r])

    Nr = np.zeros(r+1)
    for j in range(r):
            Nr[j] = np.sum( Z == j)

    nZ = len(Z)
    Nr[-1] = nZ - np.sum(Nr)
    Qr = np.sum( (Nr - p*nZ)**2. / (p*nZ))
```

```
    critval = st.chi2.ppf(1 - alpha, r)
    rej = ['cannot be', 'is']
    true = 1 * (Qr > critval)
    stat = {'Statistic': Qr, 'Quantile': critval, 'Significance' : alpha}
    message = 'Gap test: the null hypothesis H0 ' + rej[true] + ' rejected at level alpha = ' + str(alpha)
    return message, stat

def cdf(X):
    """
    Empirical CDF
    """
    Xs = np.sort(X) # Sorted version of data
    n = len(X)
    x = np.array(range(1, n+1)) / (n+1.)
    Nx = np.array([(x< x_i).sum() for x_i in x]) / float(n) # Computes empirical CDF
    xx =  np.hstack([Xs.reshape(n,1), Xs.reshape(n,1)]).flatten()
    Nxx = np.hstack([ np.hstack([Nx.reshape(n,1), Nx.reshape(n,1)]).flatten()[1:2*n], 1])
    return xx, Nxx

F = lambda u: u * (u>=0) * (u<=1) + (u>1) # CDF
Finv = lambda u: u * (u>=0) * (u<=1) + (u>1) # Inverse CDF

if __name__=='__main__':
    # Generate Data
    n = 10 # number of samples
    Y = st.uniform.rvs(size = n) # Using Scipy's built-in function
    X = LCG(n = n)[1:] # Using LCG function above

    Xs = np.sort(X) # Sorted version of data

    x = np.array(range(1, n+1)) / (n+1.)
    [xx, Nxx] = cdf(X)

    fig = plt.figure(figsize = (8,4))
    ax_qq = fig.add_subplot(121)
    ax_qq.plot(x, x, 'k--', linewidth = 1.)
    ax_qq.plot(Finv(x), Xs, '-.', linewidth = 1.)
    ax_qq.set_title('Q-Q')
    ax_cdf = fig.add_subplot(122)
    ax_cdf.plot(x, F(x), 'k--', linewidth = 1.)
    ax_cdf.plot(xx, Nxx, '-', linewidth = 1.)
    ax_cdf.set_title('CDF')

    #plt.savefig('../figures/qq_cdf_LCG_n1000.png')
    plt.show()

    [message_KS, stat_KS] = KSTest(X, 0.01)
    print(message_KS)

    [message_chi2, stat_chi] = ChiSquareTest(X, alpha=0.01)
    print(message_chi2)

    [message_ser, stat] = SerialTest(X)
    print(message_ser)

    [message_gap, stat_gap] = GapTest(X)
    print(message_gap)
```

```
fig1 = plt.figure(figsize = (8, 4))
ax1 = fig1.add_subplot(121)
ax1.plot(Y[:-1], Y[1:], '-', linewidth = 1.)
ax2 = fig1.add_subplot(122)
ax2.plot(X[:-1], X[1:], '-')
#plt.savefig('../figures/pairs.png')
plt.show()
```

Comment on built-in functions

Many of the functions that need to be implemented in this Lab already exist as `Python` built-in functions. For example, the empirical CDF can be conveniently plotted using the `ECDF` function that is available in the `statsmodels` package. The Kolmogorov-Smirnov test and χ^2 test are also available. There is, of course, very little reason to reinvent the wheel and we strongly encourage you to use these built-in functions in future Labs, if not stated otherwise. However, before naively relying on built-in functions, it is important to understand the underlying mathematical procedure.