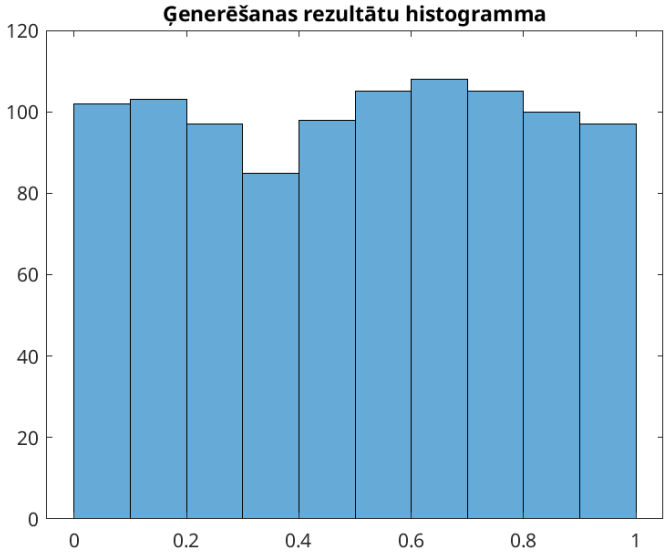
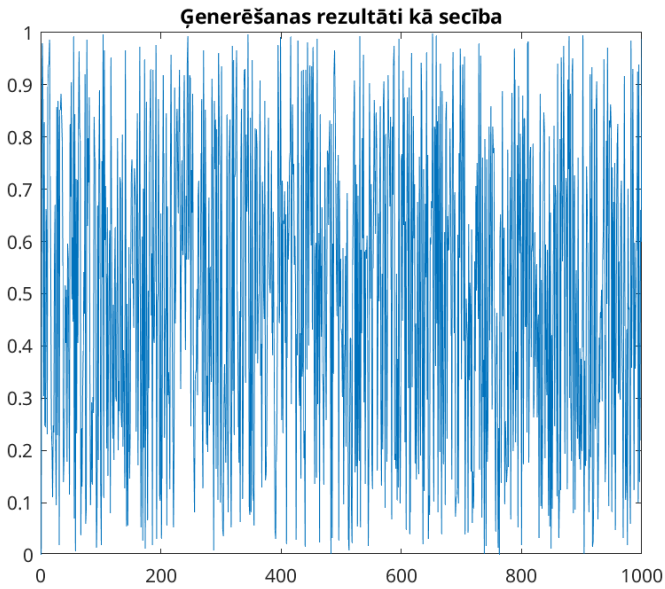
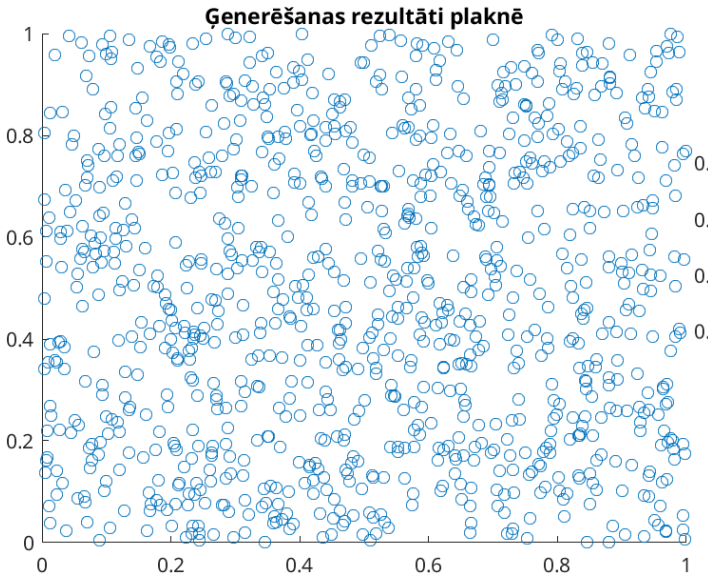
1.uzdevums

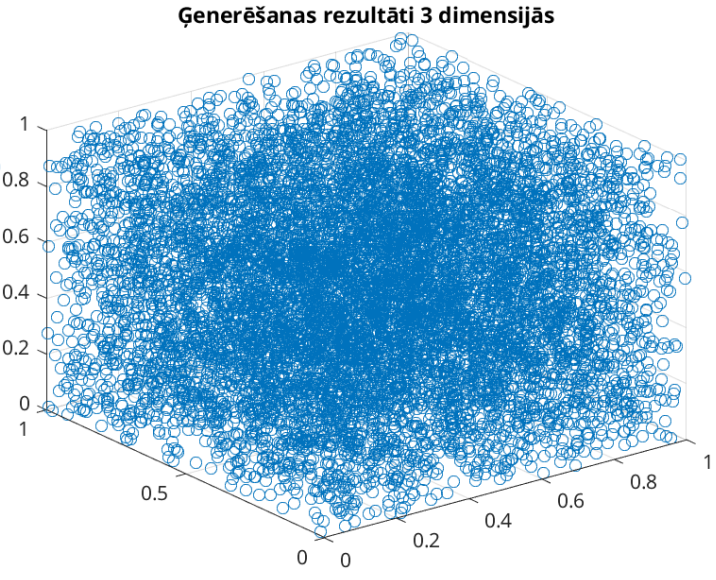
Izvēlēts multiplikatīvais kongruentais ģenerators ar parametriem no lekcijas 14. slaida, L'Ecuyer:

```
function result = custom_rng
  persistent current_x

    if isempty(current_x)
       current_x = 1;
    end
    % L'Ecuyer, 14. slaids
    m = 2147483399;
    a = 40692;

    current_x = rem(a * current_x, m);
    result = current_x / m;  % [0,1)
end
```









Attēlojot ģenerēšanas rezultātus grafiski, ģenerēšanas artifaktus – līnijas, bieži atkārtotas vērtības nenovēro.

Perioda un aperioda meklēšana

Saglabāju pirmās 2000 ģenerēšanas vērtības, meklē vismaz 100 vērtību atkārtojumu, pieņemot, ka atkārtojums var nesākties ar pirmo vērtību.

```
clear all, clc

P_length = 2000;
P = zeros(P_length, 1);
match_start = 0;
match_offset = 0;
match_count = 0;

tic
max_i = 2 ^ 33;
for i = 1:max_i
    rand = custom_rng();

    if (i < P_length)
        P(i) = rand;
    end

    if (match_start > 0)
        k = i - match_start + match_offset;
        if (P(k) == rand)
            match_count = match_count + 1;
            if (match_count > 100)
                disp(['sakritība sākot ar elementu: ' num2str(match_start)])
                disp(['aperiods: ' num2str(match_offset)])
                disp(['periods: ' num2str(match_start - match_offset)])
                break
            end
        else
            % atkārtojumu sērija pārtraukta
            match_count = 0;
            match_start = 0;
            match_offset = 0;
        end

    end

    if (i > P_length & match_start == 0)
        for j = 1:length(P)
            if (rand == P(j))
                match_start = i;
                match_offset = j;
                break;
            end
        end
    end
end
toc
```

Programmas izvads:

```
sakritība sākot ar elementu: 2147483399
aperiods: 1
periods: 2147483398
Elapsed time is 4896.515391 seconds.
```

Rezultāti sakrīt ar literatūrā norādīto – periods ir $2^{31} - 250$.

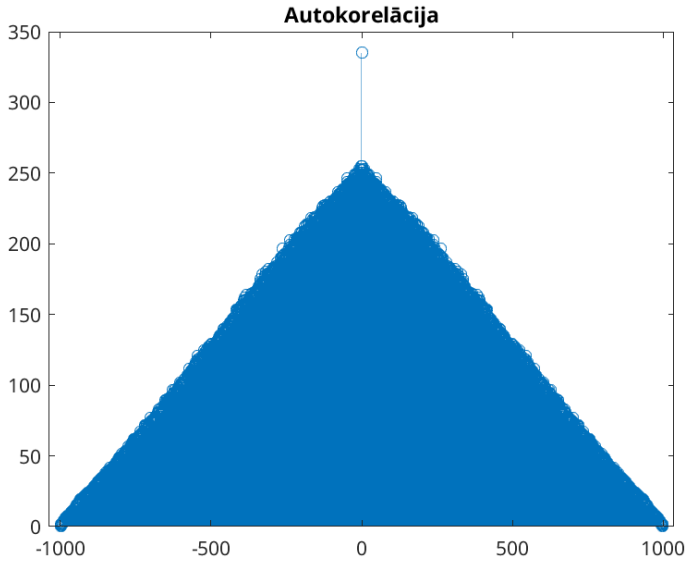
Statistiskās īpašības

```
%% g. korelācijas īpašības
n = 100000;
sample = zeros(n, 1);
for i=1:n
    sample(i) = custom_rng();
end;

nbins = 100; % number of bin
edges = linspace(0,1, nbins+1); % edges of the bins
E = n/nbins*ones(nbins,1); % expected value (equal for uniform dist)
[h,p,stats] = chi2gof(sample, 'Expected',E,'Edges',edges);
h
p
```

Rezultāti: h = 0, p = 0.9817

Secinājums – ar ļoti lielu varbūtību, sadalījums ir vienmērīgs.

```
%% g. korelācijas īpašības
clc, clear all
n = 1000;
sample = zeros(n, 1);
for i=1:n
    sample(i) = custom_rng();
end;
[c,lags] = xcorr(sample);
stem(lags,c)
title('Autokorelācija')
```



No grafika izskatās, ka pastāv korelācija ar iepriekšējām un nākamajām vērtībām, kas sagaidāms, jo katra nākamā vērtība $X_t = X_{i-1} * a \bmod m$. Iebūvētā ģeneratora, *rand*, autokorelācija izskatās identiski.

Iepriekšēja uzdevuma risināšana

Risinu integrāļa aprēķina uzdevumu no irpriekšējā mājasdarba.

```
%% h. uzdevuma risināšana
clc, clear all
syms x
fn2 = @(x) (x >= -7 & x <= -3).* (2 .* sqrt(x + 7)) + ...
     (x > -3 & x <= -1).*(0.5 .* (x + 1) .^ 2 + 2) + ...
     (x >= -1 & x < -0.75).*((x + 2) .* 8 - 6) + ...
     (x >= -0.75 & x < -0.5).*((-1.75-x) .* 4 + 8) + ...
     (x >= -0.5 & x < 0.5) .* 3 + ...
     (x >= 0.5 & x < 0.75) .* ((-1.75+x) .* 4 + 8) + ...
     (x >= 0.75 & x < 1) .* ((2 - x) .* 8 - 6) + ...
     (x >= 1 & x < 3) .* (0.5 .* (x - 1) .^ 2 + 2) + ...
     (x >= 3 & x <= 7) .* (2 .* sqrt(7-x));
ns = [ 100, 10000, 1000000 ];
for j = 1:length(ns)
    h = 0;
    n = ns(j);
    for i = 1:n
        x = custom_rng() * 14 - 7; % x ~ U[-7,7]
        y = custom_rng() * 4; % y ~ U[0, 4]

        if fn2(x) >= y
```

```
            h = h + 1;
        end;
    end;

    est_area = 14 * 4 * h / n; % -7 līdz 7 = 14
    disp( [ 'pie N = ' num2str(n) ' novērtētais laukums = ' num2str(est_area) ])
end;
% integrāļā precīzas vērtības aprēķins
syms fn(x)
fn(x) = piecewise( ...
    x >= -7 & x <= -3, 2 * sqrt(x + 7), ...
    x > -3 & x <= -1, 0.5 * (x + 1) ^ 2 + 2, ...
    x >= -1 & x < -0.75, (x + 2) * 8 - 6, ...
    x >= -0.75 & x < -0.5, (-1.75-x) * 4 + 8, ...
    x >= -0.5 & x < 0.5, 3, ...
    x >= 0.5 & x < 0.75, (-1.75+x) * 4 + 8, ...
    x >= 0.75 & x < 1, (2 - x) * 8 - 6, ...
    x >= 1 & x < 3, 0.5 * (x - 1) ^ 2 + 2, ...
    x >= 3 & x <= 7, 2 * sqrt(7-x), ...
    0);
exact_area = double(int(fn, -7, 7));
disp([ 'Precīza laukuma vērtība = ' num2str(exact_area) ])
```

Izvads:

```
iebūvētais rand
pie N = 100 novērtētais laukums = 39.76
pie N = 10000 novērtētais laukums = 37.9176
pie N = 1000000 novērtētais laukums = 38.2412
Precīza laukuma vērtība = 38.25
custom_rand:
pie N = 100 novērtētais laukums = 38.64
pie N = 10000 novērtētais laukums = 38.276
pie N = 1000000 novērtētais laukums = 38.2725
```

Secinājums

Izstrādātais ģenerators ir diezgan tuvs iebūvētajam, šajā pielietojumā iespējams pat labāks par iebūvēto ģeneratoru – no iegūtajiem mērījumiem šķiet, ka rezultāts konverģē mazliet ātrāk.

2.uzdevums

Risinājumā izmantota Shapiro-Wilk testa implementācija *swtest* no https://github.com/antagomir/scripts/blob/master/matlab/swtest.m

```
clear all, clc
syms x
n = 100;
polya_sample = zeros(n, 1);
builtin_sample = randn(n, 1);
polya_inv = @(x) (1 + (x >= 0.5) .* -2) .* sqrt(-pi ./ 2 .* log(1 - (x .* 2 - 1)
.^ 2));
for i=1:n
    polya_sample(i) = polya_inv(rand);
end;
disp([ 'Kolmogorova-Smirnova tests' ])
[h, p] = kstest(polya_sample); % Kolmogorova-Smirnova tests, noklusējuma
parametri - salīdzina ar normālsadalījumu
disp([ 'Polya: h = ' num2str(h) ', p = ' num2str(p) ])
[h, p] = kstest(builtin_sample);
disp([ 'Iebūvētais: h = ' num2str(h) ', p = ' num2str(p) ])
disp([ 'Shapiro-Wilks tests' ])
[h, p, W] = swtest(polya_sample);
disp([ 'Polya: h = ' num2str(h) ', p = ' num2str(p) ', W = ' num2str(W) ])
[h, p, W] = swtest(builtin_sample);
disp([ 'Iebūvētais: h = ' num2str(h) ', p = ' num2str(p) ', W = ' num2str(W) ])
```

Rezultāts:

```
Kolmogorova-Smirnova tests
Polya: h = 0, p = 0.84824
Iebūvētais: h = 0, p = 0.56303
Shapiro-Wilks tests
Polya: h = 0, p = 0.48266, W = 0.98765
Iebūvētais: h = 0, p = 0.89679, W = 0.99315
```

Secinājums

Nulles hipotēzi, ka ģenerētais sadalījums atbilst normālajam, nevar noraidīt. P-vērtības variē diezgan plašā diapazonā – dažos atkārtojumos iegūts arī rezultāts, kur nulles hipotēzi noraida. Iebūvētā ģeneratora statistiskās īpašības ir diezgan tuvas, šo *Polya's* shēmu tātad noteikti var izmantot citās valodās/programmās, kur nav pieejams iebūvēts/labāks normāli sadalītu skaitļu ģenerators.

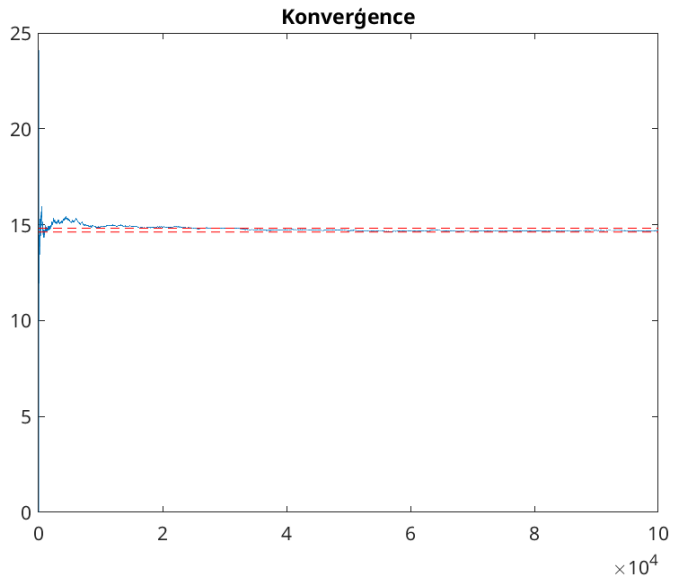
3.uzdevums

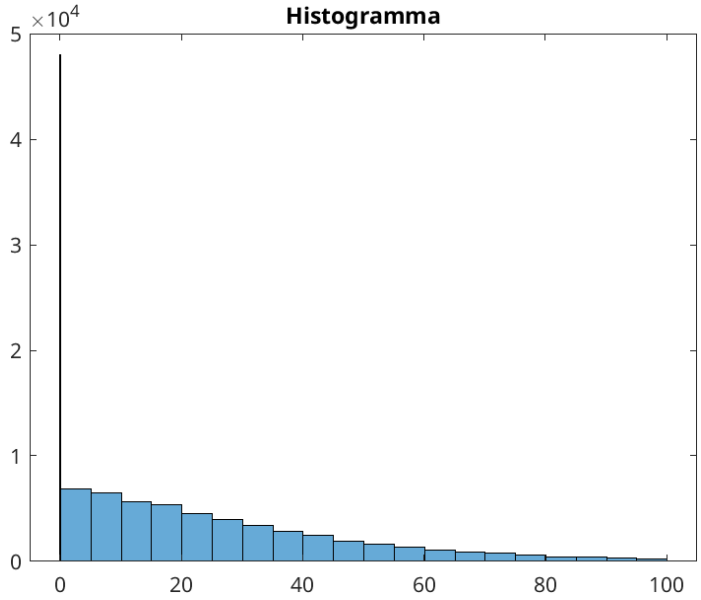
```
strike_price = 100;
T = 1;
s_0 = 100;
volatility = 0.3;
risk_free_rate = 0.06;
n = 100000;
sample = zeros(n, 1);
estimates = zeros(n - 1, 1);
for i = 1:n
    % akcijas cena
    S_T = s_0 .* exp((risk_free_rate - 0.5 .* (volatility .^ 2)) .* T +
volatility .* randn .* sqrt(T));
    % diskontēta starpība starp vienošanās cenu un akcijas cenu
    optPrice = exp (- risk_free_rate * T) * (S_T - strike_price);
    if (optPrice < 0)
        optPrice = 0;
    end
    sample(i) = optPrice;

    estimates(i) = mean(sample(1:i)); % novērtējums - vidējā vērtība no
līdzšinējām
end

% iebūvētā funkcija šim aprēķinam
[Call,Put] = blsprice(s_0,strike_price,risk_free_rate,T,volatility);
Call
```

```
plot(estimates)
hold on
eps = 0.10;
convLine = ones(length(estimates), 1) * Call;
plot(convLine - eps, '--r')
plot(convLine + eps, '--r')
% plot(convLine, '--g')
hold off
mean(sample)
figure
histogram(sample, [0 eps:5:100 100])
```

Izvads

```
Call = 14.7171
ans = 14.6962
```





Secinājumi

Simulācijas dotais novērtējums ir ļoti tuvs analītiski aprēķinātajam. Attēlojot rezultātus grafiski interesanti, ka apmēram pusē gadījumu līguma vērtība ir 0.