Table of Contents

Clear workspace

```
clear all;
clc;
```

A1, A2

```
% Create state-vector for LFSR with 22 bits
S = zeros(1,22);

% Set initial-state of LFSR to (0,0,1) decimal
S(1) = 1; % S(1) is LSB
S_initial = S; % store initial state of S

% Create large output vector to store output stream
DATA_OUT = zeros(1, 2^22-1);

% Observe LSFR for 5000000 clock ticks
for time = 1:5000000
    % Store bit to be shifted out (LSB found on LEFT end of vector)
    LSB = S(1);
    DATA_OUT(time) = LSB; % Store shifted bit into output

    % Shift bits that are not part of feedback polynomial
    S(1:20) = S(2:21);

    % Feedback polynomial is x^22 + x^21 + 1
    S(21) = xor(S(22),LSB); % XOR LSB with bit 22 to generate bit 21
    S(22) = LSB; % Shift LSB into MSB (bit 22)

    % check if we have returned to the initial state
    if S == S_initial
        period = time;
        fprintf('The state at time %d = the initial state, the period is %d.
\n', time, period);
        break;
    end
end
```

A3, A4

```
if exist('period', 'var') == 0
```

```
    fprintf(2,'DID NOT FIND PERIOD !!!! \n');
else
    % Write bits to file, as their decimal representation per byte
    fprintf('Printing random numbers to file my_random_numbers.m...\n');
    num_bytes = ceil(period / 8);

    % Open file for writing
    fid = fopen('my_random_numbers.m', 'w');

    % Write starting information
    fprintf(fid, 'num_bytes = %d;\n', num_bytes);
    fprintf(fid, 'RANDOM_DATA_OUT(1:num_bytes) = [...\n');

    % Write out only 1 period, and not the entire array
    BITS = DATA_OUT(1:period);

    % Append zeroes to LSB make the bitarray divisible by 8
    BITS = [BITS, zeros(1, 8-mod(period, 8))];

    % Reverse the array so the LSB is at index 1
    BITS = flip(BITS);

    byte = 1;

    % Convert each byte into decimal and write it out to file
    while byte <= num_bytes
        DEC = num2str(BITS(byte*8-7:byte*8)); % Get a new byte and convert to
 string
        DEC = bin2dec(DEC); % Convert to base 10
        fprintf(fid, '%d, ', DEC); % Write to file, comma separated

        % Write 16 bytes per line in the file
        if mod(byte,16) == 0
            fprintf(fid, '...\n');
        end
        byte = byte + 1; % Increment byte counter
    end

    % Close the file
    fprintf(fid, '...\n];');
    fclose(fid);
    fprintf('Finished printing random numbers to file my_random_numbers.m\n');
end
```

A5, A6

```
% Determine 0-runs and 1-runs and their lengths
changeBits = [1, diff(BITS(1:end-1))] ~= 0; % Determine indices where value
 changes from 0 to 1 or vice versa
% Get total lengths of runs by getting cumulative sum of changeBits and
% using histogram bin counts function to get number of elements in each bin
% (length of each run)
counts = histcounts(cumsum(changeBits), 1:sum(changeBits) + 1);
```

```
bitSequences = [BITS(changeBits)',counts']; % Put into 2 colum matrix with 0
 or 1 run and length (eg. 1 10 is 1-run of length 10)

% initialize 4 row column for A5, A6
zeroruns_table = zeros(4, 24); oneruns_table = zeros(4, 24);

% first row of table, set first row equal to k = 1, 2, 3...
zeroruns_table(1, 1:24) = 1:24;
oneruns_table(1, 1:24) = 1:24;

% second row of table, tally number of 0-runs or 1-runs of length k
for i = 1:size(bitSequences, 1)
    if (bitSequences(i, 2) <= 25) % check if k < 25 as table only goes to k =
 24
        % Increment the run count for the corresponding run length
        if bitSequences(i, 1) == 0 % check if is 0 run
            oneruns_table(2, bitSequences(i, 2)) = oneruns_table(2,
 bitSequences(i, 2)) + 1;
        else % if 1 run
            zeroruns_table(2, bitSequences(i, 2)) = zeroruns_table(2,
 bitSequences(i, 2)) + 1;
        end
    end
end

% third row of table
% Find the total number of zeros and ones from the runs tables by multiply
% number runs of each k and multiplying by k, then taking sum
totalZeros = sum(zeroruns_table(1,:) .* zeroruns_table(2,:));
totalOnes = sum(oneruns_table(1,:) .* oneruns_table(2,:));
for i = 1:24
   % Calculate the actual probability of each run length occurring by
   % dividing number of runs of length k by total observed runs
   oneruns_table(3, i) = oneruns_table(2, i) / sum(oneruns_table(2,:));
   zeroruns_table(3, i) = zeroruns_table(2, i) / sum(zeroruns_table(2,:));
end

% fourth row of table
for i = 1:24
   % Calculate the theoretical probability of each run length occurring
   % 2^(-k) where k is the run length
   oneruns_table(4, i) = 2^-oneruns_table(1, i);
   zeroruns_table(4, i) = 2^-zeroruns_table(1, i);
end
```

Published with MATLAB® R2022b