Exercise – Function – Halton

This custom **halton** function extends MATLAB's built-in capabilities by offering a flexible and simulation-ready framework for generating Halton sequences and transforming them into standard normal draws. Unlike MATLAB's default haltonset, this implementation allows users to specify prime bases, apply burn-in and leap adjustments, and incorporate randomization or scrambling techniques based on established methods from Train (2003) and Bhat (2003). It includes robust input validation, dimension-specific subsetting, and automatic transformation to the standard normal distribution, making it particularly well-suited for econometric simulations and quasi-Monte Carlo integration.

```matlab
function [H,Z] = halton(N,dimensions,draws,varargin)
%HALTON Generate Halton sequences and transform to standard normal.
% Version 2.0, Updated August 2025
% Compatible with MATLAB R2014a and later
%
% Original Author: Elisabeth Beusch
% Modified by: Tunga Kantarci, August 2025
%
% Description of Modifications:
%   - Adjusted comments and code to prevent edge cases and input
%     conflicts that could cause runtime errors or misbehavior.
%
% This function computes Halton sequences using the specified prime
% bases and transforms them into standard normal draws. The
% implementation follows chapters 9.3.3-9.3.5 in Train (2003) and
% includes options for randomization and scrambling with respect to
% Bhat (2003).
%
% Syntax:
%   [H, Z] = halton(N,dimensions,draws)
%   [H, Z] = halton(N,dimensions,draws,'Name',Value, ...)
%
% Inputs:
%   N            - Number of observational units.
%   dimensions   - Number of dimensions of the integral.
%   draws        - Number of draws per observational unit.
%
% Name-Value Pair Arguments:
%   'prime'      - Vector of prime numbers used as bases.
%                  Default: 0 (uses first 'dimensions' primes).
%   'burn'       - Number of initial Halton points to skip.
%                  Default: 50.
%   'leap'       - Number of points to skip between draws.
%                  Default: 0.
%   'random'     - Logical flag to apply randomization with respect to
%                  Bhat (2003).
%                  Set to 1 to enable. Default: 0.
%   'scramble'   - Logical flag to scramble the Halton sequence.
%                  Recommended for high-dimensional settings.
```

```
%                    Set to 1 to enable. Default: 0.
%
% Outputs:
%   H - Halton draws of size (N x dimensions x draws).
%   Z - Corresponding values from a standard normal distribution.
%
% Notes:
%   - The first Halton point (zero) is always dropped.
%   - The function performs basic checks on prime validity and
%     dimensionality.
%
% References:
%   Bhat, C. R., 2003. Simulation estimation of mixed discrete choice
%   models using randomized and scrambled Halton sequences.
%   Transportation Research Part B: Methodological, 37 (9), 837-855.
%
%   Train, K., 2003. Discrete Choice Methods with Simulation. Cambridge
%   University Press.
%
% ---------- BEGIN FUNCTION BODY BELOW ----------

%% Optional input arguments
opts = inputParser;
addParameter(opts,'prime',0,@isnumeric);
addParameter(opts,'burn',50,@isnumeric);
addParameter(opts,'leap',0,@isnumeric);
addParameter(opts,'random',0,@isnumeric);
addParameter(opts,'scramble',0,@isnumeric);
parse(opts, varargin{:});

prime      = opts.Results.prime;
burn       = opts.Results.burn;
leap       = opts.Results.leap;
randhalt   = opts.Results.random;
toscramble = opts.Results.scramble;

%% Define prime bases and dimensions
% If prime == 0, use first 'dimensions' primes implicitly
if isequal(prime,0)
    prime = primes(100); % Generate a pool of primes
    prime = prime(1:dimensions); % Select first 'dimensions' primes
end

% Force row vector
prime = prime(:)';

% Validate prime vector
if dimensions ~= numel(prime)
    error('Dimensions do not match number of primes supplied');
end
```

```
if any(~isprime(prime))
    error('Non-prime was supplied');
end
if dimensions == 1 && prime(1) <= 2
    error('For dimension == 1, a prime > 2 must be supplied');
end

% Set Halton set dimensionality
p = max(prime); % haltonset must cover all primes used

%% Warnings
if dimensions >= 7 && toscramble == 0
    warning(['Scrambling is recommended ' ...
        'for high-dimensional Halton draws.']);
end

if (burn - max(prime) <= 10) || (isscalar(prime) && dimensions >= 13)
    warning(['The default burn setting ' ...
        'might be too short for your primes']);
end

%% Generate Halton sequences
if leap == 0
    h = haltonset(p,'Skip',burn+1); % Sequence starts at zero; skip burn
else
    h = haltonset(p,'Skip',burn+1,'Leap',leap);
end

% Scramble if requested
if toscramble == 1
    h = scramble(h,'RR2');
end

hp = net(h,N*draws); % Generate Halton points

% Select dimensions based on supplied primes
all_primes = primes(p+1);
[~,primi] = ismember(prime,all_primes);
if any(primi == 0)
    error(['One or more supplied primes are not valid' ...
        'or not found in the prime list.']);
end
hp = hp(:, primi); % Subset to requested dimensions

%% Randomization with respect to Bhat (Train, 2003, p. 264)
if randhalt == 1
    mu = rand(1,dimensions);
    mu = repmat(mu,N*draws,1);
    hp = hp+mu;
    hp(hp>1) = hp(hp>1)-1; % Wrap values > 1 back into [0,1]
```

```
end

%% Reshape output
expected_cols = dimensions;

actual_cols = size(hp,2);

if actual_cols ~= expected_cols
    error(['Mismatch in Halton output dimensions: expected ' ...
        '%d,got %d.'],expected_cols,actual_cols);
end

H = reshape(hp',dimensions,draws,N); % dimensions x draws x N
H = permute(H,[3,1,2]); % N x dimensions x draws

%% Transform to standard normal
if nargout >= 2
    Z = norminv(H); % Requires Statistics and Machine Learning Toolbox
end
```