```
classdef MarkovChains
    methods(Static)
```

# MarkovChains

Written by Curtis Aquino (2019). Contains:

1. Rouwenhorst()
2. StationaryDistribution()
3. Moments()
4. Simulate()

```
function [Y,PTM]    = Rouwenhorst(N,shockvar,p,q)

%###########################################################################
% This function applies the Rouwenhorst method as suggested by Karen
% Kopecky (2010) to discretize a stationary AR(1) process with
 normally
% distributed errors of the form: zt = rho z_{t-1} + e_t
%###########################################################################

% *******************************************************
% Default is symmetry of p and q
% *******************************************************

if nargin < 5
    q           = p;
end

% *******************************************************
% Recursive formulation
% *******************************************************

Pi = cell(1,N);
for i = 2:N
    if i == 2
        Pi{i}   = [p,1-p;1-p,p];
    else
        Ze      = zeros(i-1,1);
        Pi{i}   = p*[Pi{i-1},Ze;Ze',0]+(1-p)*[Ze,Pi{i-1};0,Ze']+(1-
q)*[Ze',0;Pi{i-1},Ze]+q*[0,Ze';Ze,Pi{i-1}];
        Pi{i}(2:(end-1),:) = Pi{i}(2:(end-1),:)/2;
    end
end
PTM             = Pi{end};

% *******************************************************
% Generate the state space
% *******************************************************
```

```
Psi                = sqrt(N-1)*sqrt(shockvar);
Y                  = fliplr(linspace(-Psi,Psi,N))';

end

function F            = StationaryDistribution(PTM,maxIter)

%###########################################################################
% This function takes a probability transition matrix, PTM, and
 iterates on
% an equispaced initial guess until machine epsilon convergence. If no
% maximum number of iterations, maxIter, is specified,
% StationaryDistribution() will conclude that no stationary
 distribution
% exists if norm(pi_{2000}-pi_{1999}) > 10^(-16)
%###########################################################################

% *********************************************************
% Default argument
% *********************************************************

if nargin < 2
    maxIter = 2000;
end

% *********************************************************
% Ensures that input is a PTM
% *********************************************************

if range(size(PTM)) ~= 0
    error('This probability transition matrix is not a square')
end

% **********************************************************
% Finds the stationary distribution depending on data class
% **********************************************************

type                = class(PTM);
switch type
    case 'table'
        pi0         = ones(1,size(PTM,1))/size(PTM,1);
        pi1         = Inf;
        itr         = 1;
        thr         = Inf;
        while thr > eps
            pi1     = pi0 * PTM{:,:};
            pi0     = pi1;
            itr     = itr + 1;
            thr     = norm(pi1-pi0);
            if itr > maxIter
                error('No stationary distribution exists')
            end
        end
```

```
    case 'double'
        pi0         = ones(1,size(PTM,1))/size(PTM,1);
        pi1         = Inf;
        itr         = 1;
        thr         = Inf;
        while thr > eps
            pi1     = pi0 * PTM(:,:);
            thr     = norm(pi1-pi0);
            pi0     = pi1;
            itr     = itr + 1;
            if itr > maxIter
                error('No stationary distribution exists')
            end
        end
    otherwise
        error('Incorrect input type')
end

% ********************************************************
% Results
% ********************************************************

F                   = pi0';

end

function F          = Moments(PTM,X)

%###########################################################################
% Given a probability transition matrix, PTM, and a vector of values
 for
% each state, X, Moments() will produce a table output that displays
 the
% mean, variance, covariance, and autocorrelation associated with the
 PTM.
% Note that these are not based on simulation, but on the explicit
 formulas
% suggested by Paul Klein (2019).
%###########################################################################

% ********************************************************
% Fixes user input
% ********************************************************

if isrow(X)
    X = X';
end
if istable(PTM)
   PTM = PTM{:,:};
end

% ********************************************************
% Derives the stationary distribution, if it exists
% ********************************************************
```

```
StDs            = MarkovChains.StationaryDistribution(PTM);

% *********************************************************
% Computes moments following Klein (2019)
% *********************************************************

Mean            = sum(StDs.*X);
Variance        = sum(((X-Mean).^2).*StDs);
Covariance      = sum(StDs'.*sum(X(:)'*X.*PTM,2))-Mean^2;
Autocorrelation = Covariance/Variance;

% *********************************************************
% Produces output as a table
% *********************************************************

F               = array2table([Mean,Variance],'VariableNames',
{'Mean','Variance'});

end

function F          = Simulate(PTM,T,S0)

%############################################################################
% Given a probability transition matrix, PTM, and a number of periods,
 T,
% Simulate() will generate a time series of state variables based on
 the
% probability transition based with the first 5% of options burned to
% remove dependence on initial conditions. If no argument for an
 initial
% state, S0, is given, the default argument begin simulation from
 state 1.
%############################################################################

% *********************************************************
% Default argument
% *********************************************************

if nargin < 3
    S(1)    = 1;
else
    S(1)    = S0;
end

% *********************************************************
% Checks whether the PTM is valid
% *********************************************************

if range(size(PTM)) ~= 0
    error('This is not a probability transition matrix')
end

% *********************************************************
```

```
% Builds bounds for uniform draws
% ********************************************************

TT              = T + round(0.05*T);
for i = 1:size(PTM,1)
    Bounds      = cumsum(PTM(i,:));
    Bd(:,:,i)   = [[0;Bounds(1:(end-1))'],Bounds(:)];
end
UniformDraws    = rand(TT,1);

% ********************************************************
% Simulation
% ********************************************************

for i = 2:TT
    S(i)         = find(UniformDraws(i-1)>Bd(:,1,S(i-1)) &
 UniformDraws(i-1)<Bd(:,2,S(i-1)));
end

% ********************************************************
% Burns 5% to mitigate dependence on initial conditions
% ********************************************************

F = S(round(0.05*T)+1:end)';


end

    end
end
```

*Published with MATLAB® R2018b*