Doutorado em Economia - EPGE/FGV *MDPEMF024 - Métodos Numéricos*

Lista #1

2 de maio de 2022

Aluno: Rafael Vetromille *Professor: Cézar Santos / TA: Ana Paula Ruhe*

Informações: Para essa lista, você trabalhará com o seguinte processo estocástico AR(1):

$$z_t = \rho z_{t-1} + \varepsilon_t$$

com $\varepsilon_t \sim N(0, \sigma^2)$. Assuma, por enquanto, que $\rho = 0.95$ e $\sigma = 0.007$, calibração de Cooley and Prescott (1995).

Questão #1

Discretize o processo acima usando o método de Tauchen (1986). Use 9 pontos.

Resposta. O código a seguir representa uma das possibilidades de se implementar o método de Tauchen no `Matlab`. No código em anexo, as funções estão no final do arquivo `.m`. Primeiramente, definimos um vetor de zeros e uma matriz diagonal para depois alocarmos os valores que serão obtidos. Agora devemos determinar a imagem do grid, a localização e o número de pontos. O "upper bound" será definido como θ_N e será igual a

$$\theta_N = +m \times \sqrt{\frac{\sigma^2}{1-\rho^2}}$$

em que $\sigma(\sqrt{1-\rho^2})^{-1}$ é o desvio padrão incondicional de θ e m é um parâmetro de escala. O limite inferior, por sua vez, será definido como

$$\theta_1 = -m \times \sqrt{\frac{\sigma^2}{1-\rho^2}}$$

Os demais $N-2$ pontos do grid serão calculados de forma equidistantes entre θ_1 e θ_N, isto é, cada passo poderá ser calculado como:

$$\texttt{step} = \frac{\theta_N - \theta_1}{N-1}$$

e podemos fazer um for loop para calcular os $N-2$ pontos restantes, conforme descrito nas linhas 15 a 17 do código abaixo.

Em seguida, precisamos saber qual é a probabilidade de passar para o estado θ_j condicional de estar no estado θ_i com $i, j \in [1, 2, \ldots, N]$ em que N represente o número de pontos no grid. Para tanto, denote esta probabilidade por $p_{i,j} = P(\theta_{t+1} = \theta_j \mid \theta_t = \theta_i)$ e seja P a matriz de $\{p_{i,j}\}_{i=1,j=1}^{N,N}$. A abordagem de Tauchen discretiza a distribuição de probabilidade condicional.

Suponha que o estado atual seja θ_i . Qual é a probabilidade de se mover para θ_j? Após algumas contas, podemos calculá-la da seguinte forma:

$$p_{i,j} = \Phi\left(\frac{\theta_j + \Delta\theta/2 - \rho\theta_i}{\sigma}\right) - \Phi\left(\frac{\theta_j - \Delta\theta/2 - \rho\theta_i}{\sigma}\right)$$

A transição para os pontos do canto deve ser tratada de forma diferente

$$p_{i,1} = \Phi\left(\frac{\theta_1 + \Delta\theta/2 - \rho\theta_i}{\sigma}\right) \quad \wedge \quad p_{i,N} = 1 - \Phi\left(\frac{\theta_N - \Delta\theta/2 - \rho\theta_i}{\sigma}\right)$$

Agora, podemos calcular as probabilidades de transição $p_{i,j}$ para todo i e j. O resultado desta discretização é um grid $\{\theta_i\}_{i=1}^N$ e a matriz de probabilidade de transição P. A função no quadro abaixo implementa exatamente esse algoritmo e retorna os dois objetos da discretização:

```
% tauchen.m
function [theta, Pi] = tauchen(m, rho, sig, N)

% Setting the grid and the trasition matrix
theta    = zeros(N,1)';
Pi       = eye(N,N);

% Specifying the grid
theta(N)     = + m * sqrt(sig^2/(1-rho^2));
theta(1)     = - m * sqrt(sig^2/(1-rho^2));

% Computing the remaining (N-2) points [equidistantly distributed]
step      = (theta(N)-theta(1))/(N-1);

for i = 2:(N-1)
   theta(i) = theta(i-1) + step;
end

% Computing the transition probabilities
for j = 1:N
    for k = 1:N
        % The transition to the corner points have to be treated differently
        if k == 1
            Pi(j,k) = normcdf((theta(1) - rho*theta(j) + step/2) / sig);
        % The transition to the corner points have to be treated differently
        elseif k == N
            Pi(j,k) = 1 - normcdf((theta(N) - rho*theta(j) - step/2) / sig);
        else
        % The other points will be calculated by:
           Pi(j,k) = normcdf((theta(k) - rho*theta(j) + step/2) / sig) - ...
               normcdf((theta(k) - rho*theta(j) - step/2) / sig);
        end
    end
end
end
```

Agora, podemos aplicar a função anterior e obter os resultados da discretização. Antes disso, precisamos definir as variáveis calibradas, o número de estados e o parâmetro de escala (ad-hoc). Assim, temos:

```
%% Formal beggining of the document
clearvars;
close all;
clc;

%% Exercise 1.

% Calibration
rho      = 0.95;
sig      = 0.007;

N        = 9;        % Number of points (states);
m        = 3;        % Scaling parameter (I don't know why!);

% Tauchen's Method

% FUNCTION: [S, P] = tauchen(m, rho, sig, N)
%
% INPUTS:
%     m:    scalar, scaling parameter
%   rho:    scalar, AR-coefficient
%   sig:    scalar, standard deviation of innovations
%     n:    scalar, number of grid points for the discretized process
%
% OUTPUTS:
%    S:     column vector of size Nx1, contains all possible states in ascending order
%    P:     matrix of size NxN, contains the transition proabilities.
%           Rows are current state and columns future state

[S9T, P9T] = tauchen(m, rho, sig, N)
```

Nesse código, as variáveis S9T e P9T representam o grid de estados ($N \times 1$) e a matriz de transição ($N \times N$) obtida pelo método de Tauchen, respectivamente. O resultado via Matlab (%.4f) obtido será:

```
S9T =

   -0.0673   -0.0504   -0.0336   -0.0168         0    0.0168    0.0336    0.0504    0.0673

P9T =

    0.7644    0.2347    0.0009    0.0000    0.0000         0         0         0         0
    0.0592    0.7405    0.1997    0.0006    0.0000    0.0000         0         0         0
    0.0001    0.0747    0.7569    0.1679    0.0004    0.0000    0.0000         0         0
    0.0000    0.0001    0.0931    0.7669    0.1396    0.0002    0.0000    0.0000         0
    0.0000    0.0000    0.0002    0.1147    0.7702    0.1147    0.0002    0.0000         0
    0.0000    0.0000    0.0000    0.0002    0.1396    0.7669    0.0931    0.0001    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0004    0.1679    0.7569    0.0747    0.0001
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0006    0.1997    0.7405    0.0592
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0009    0.2347    0.7644
```

■

Questão #2

Discretize o processo acima usando o método de Rouwenhorst. Use 9 pontos.

Resposta. O método de Rouwenhorst, recentemente reintroduzido por Kopechy and Suen (RED, 2010), não depende da discretização da distribuição condicional de θ. O algoritmo pode ser descrito da seguinte forma:

1. Escolha o tamanho do grid e defina esse tamanho como N. Agora, calcule seus pontos finais da seguinte forma:

$$\theta_N = +\sigma_\theta \times \sqrt{N-1} \quad \wedge \quad \theta_1 = -\sigma_\theta \times \sqrt{N-1}$$

onde $\sigma_\theta^2 = \sigma^2/(1-\rho^2)$.

2. Agora, calcule a matriz de transição de forma recursiva da seguinte forma:

$$P = p \times \begin{bmatrix} P_{N-1} & \mathbf{0} \\ \mathbf{0}' & 0 \end{bmatrix} + (1-p) \times \begin{bmatrix} \mathbf{0} & P_{N-1} \\ 0 & \mathbf{0}' \end{bmatrix} + (1-p) \times \begin{bmatrix} \mathbf{0}' & 0 \\ P_{N-1} & \mathbf{0} \end{bmatrix} + p \times \begin{bmatrix} 0 & \mathbf{0}' \\ \mathbf{0} & P_{N-1} \end{bmatrix}$$

onde `p = (1+rho)/2`, `P2 = [p 1-p; 1-p p]` e $\mathbf{0}$ é um vetor coluna de zeros de dimensão $(N-1) \times 1$.

3. Normalize as linhas para somarem 1.

A função a seguir representa uma das formas de implementar o método de Rouwenhorst no `Matlab`:

```
% Rouwenhorst.m
function [theta, Pi] = rouwenhorst(rho, sig, N)

% Specifying the grid
theta(N)    = + (sig / sqrt(1-rho^2)) * sqrt(N-1);
theta(1)    = - (sig / sqrt(1-rho^2)) * sqrt(N-1);

% Computing the remaining (N-2) points [equidistantly distributed]
step        = (theta(N)-theta(1))/(N-1);

for i = 2:(N-1)
   theta(i) = theta(i-1) + step;
end

% Computing p and P2
p  = (1+rho)/2;
Pi = [p 1-p; 1-p p];

% Compute the transition matrix P recursively
for n = 3:N
    Pi =    p * [Pi zeros(n-1,1); zeros(1,n-1) zeros(1,1)] + ...
         (1-p) * [zeros(n-1,1) Pi; zeros(1,1) zeros(1,n-1)] + ...
         (1-p) * [zeros(1,n-1) zeros(1,1); Pi zeros(n-1,1)] + ...
             p * [zeros(1,1) zeros(1,n-1); zeros(n-1,1) Pi];
end

% Normalize rows to add up to 1
for i = 1:N, Pi(i,:) = Pi(i,:)/sum(Pi(i,:)); end
end
```

Agora, podemos aplicar a função anterior e obter os resultados da discretização. Note que já definimos as variáveis calibradas no início do código e, portanto, não precisaríamos repetir, mas para fins didáticos coloquei a calibração novamente. O número de estados continua a ser $N = 9$. Assim, temos:

```
% Calibration
rho      = 0.95;
sig      = 0.007;

% Number of points (states)
N        = 9;

% Rouwenhorst's Method

% FUNCTION: [S, Pi] = rouwenhorst(m, rho, sig, N)
% At the end of the document.
%
% INPUTS:
%   rho:    scalar, AR-coefficient
%   sig:    scalar, standard deviation of innovations
%   n:      scalar, number of grid points for the discretized process
%
% OUTPUTS:
%   S:      column vector of size Nx1, contains all possible states in ascending order
%   Pi:     matrix of size NxN, contains the transition proabilities.
%           Rows are current state and columns future state

[S9R, P9R] = rouwenhorst(rho, sig, N)
```

Nesse código, as variáveis `S9R` e `P9R` representam o grid de estados ($N \times 1$) e a matriz de transição ($N \times N$) obtida pelo método de Rouwenhorst, respectivamente. O resultado via Matlab (%.4f) obtido será:

```
S9R =

   -0.0634   -0.0476   -0.0317   -0.0159    0.0000    0.0159    0.0317    0.0476    0.0634

P9R =

    0.8167    0.1675    0.0150    0.0008    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0209    0.8204    0.1469    0.0113    0.0005    0.0000    0.0000    0.0000    0.0000
    0.0005    0.0420    0.8231    0.1261    0.0081    0.0003    0.0000    0.0000    0.0000
    0.0000    0.0016    0.0630    0.8247    0.1051    0.0054    0.0001    0.0000    0.0000
    0.0000    0.0001    0.0032    0.0841    0.8253    0.0841    0.0032    0.0001    0.0000
    0.0000    0.0000    0.0001    0.0054    0.1051    0.8247    0.0630    0.0016    0.0000
    0.0000    0.0000    0.0000    0.0003    0.0081    0.1261    0.8231    0.0420    0.0005
    0.0000    0.0000    0.0000    0.0000    0.0005    0.0113    0.1469    0.8204    0.0209
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0008    0.0150    0.1675    0.8167
```

■

Questão #3

Simule o processo contínuo para 10000 períodos. Faça o mesmo para os processos discretizados (lembre-se de usar as mesmas realizações para os choques). Compare os caminhos para cada processo (gráficos serão úteis aqui). Se eles não estiverem muito próximos, utilize mais pontos.

Resposta. Primeiramente, implementemos a função que simula o AR(1) no Matlab. Nesse caso, colocamos `Z[0] = 0` (estado inicial), simulamos um vetor aleatório com $T = 10000$ observações obtido de uma distribuição normal com a função `normrnd()` com parâmetros $\sigma = 0.007$ e $\mu = 0$. A seguinte função implementa o processo:

```
% ar1_simulation.m
function [eps, Z] = ar1_simulation(rho, sig, N)

    Z   = zeros(N,1)';
    eps = normrnd(0, sig, N, 1)';

    for i=2:N
        Z(i) = rho*Z(i-1) + eps(i);
    end
end
```

O resultado pode ser obtido utilizando-se a função acima da seguinte forma:

```
%%%% Simulation of the AR(1) process %%%%

% Setting seed, for replicate the results
rng(1234)

% Number of periods
T = 10000;

% Simulation AR(1) process and shocks

% FUNCTION: [eps, Z] = ar1_simulation(rho, sig, N)
% At the end of the document.
%
% INPUTS:
%   rho:    scalar, AR-coefficient
%   sig:    scalar, standard deviation of innovations
%   N:      scalar, number of periods of time
%
% OUTPUTS:
%   eps:   column vector of size Nx1, contains all shocks
%   Z:     column vector of size Nx1, contains all realizations of Z(t)

[eps, Z] = ar1_simulation(rho, sig, T);
```

O comando `rnd(1234)` tem por objetivo deixar o código replicável para outros usuários obterem sempre os mesmos resultados. Essa função tem como *output* dois vetores como descrito no comentário.

Para retornarmos a simulação para os processos discretizados, primeiramente criamos um vetor chamado `idx` que alocará todos os índices do vetor de estados `theta`, isto é, a posição no vetor $N \times 1$. Após isso, definimos o estado inicial. O primeiro estado (estado inicial) é simplesmente a mediana, pois o grid é centrado na média e tem $N = 9$ elementos que é um número ímpar. Agora, para cada período dos $T = 10000$ choques simulados, registramos a probabilidade de ir para o primeiro estado. Após isso, percorremos cada um dos estados do grid discreto (via colunas da matriz) da seguinte forma: pegamos a linha da matriz de transição do espaço de estado anterior e verificamos quais colunas (índices) a CDF do choque `i` é menor do que a probabilidade acumulada obtida pela soma acumulada das linhas da matriz de transição com a função `cum = cumsum(Pi')';` e, por fim, salvamos o primeiro elemento do vetor resultante. Esse será o nosso próximo estado, podendo permanecer no mesmo ou mudar. No Matlab, uma forma de implementar esse código é da seguinte maneira:

```
% shock.m
function idx = shock(th0, sig, eps, Pi, T)

idx    = ones(T,1);
idx(1) = th0;
cum    = cumsum(Pi')';

for i = 2:T
    x     = find(normcdf(eps(i), 0, sig) ≤ cum(idx(i-1),:));
    idx(i) = x(1);
end
end
```

O resultado será o vetor `idx` com 10000 elementos dos índices do vetor de estados. Para obtermos esse resultado basta rodarmos as seguintes linhas de código:

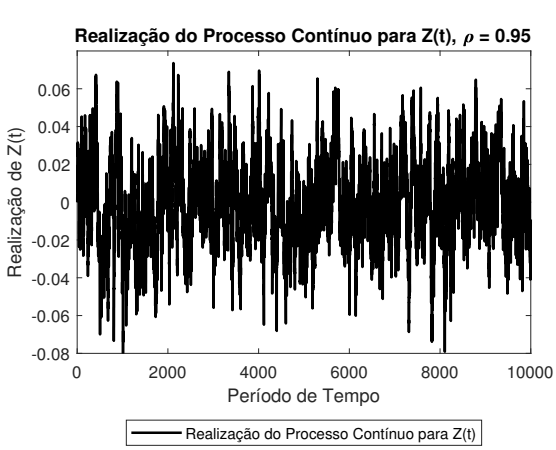
```
% Initial state (median)
th0 = find(S9T == median(S9T(:)));

% Correspondents indices
IDXT = shock(th0, sig, Shock, P9T, T);
IDXR = shock(th0, sig, Shock, P9R, T);

% Discretized AR(1)
AR1D_T99 = S9T(IDXT);
AR1D_R99 = S9R(IDXR);
```

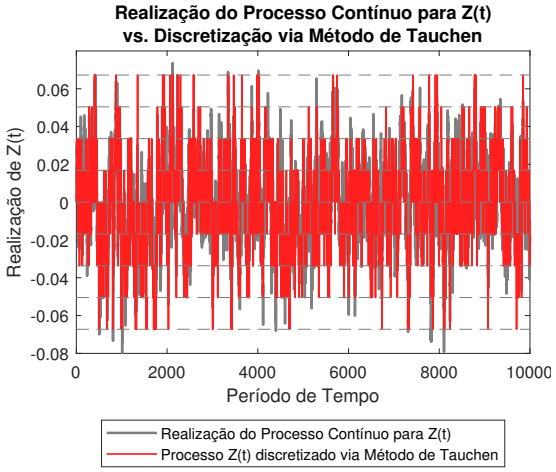
As últimas duas linhas transformam os vetores `IDXT` (posição dos estados do grid obtido pelo método de Tauchen) e `IDXR` (posição dos estados do grid obtido pelo método de Rouwenhorst) nos valores de fato que os estados assumem. Podemos ver os primeiros dez resultados de ambos os métodos discretizados da seguinte maneira:

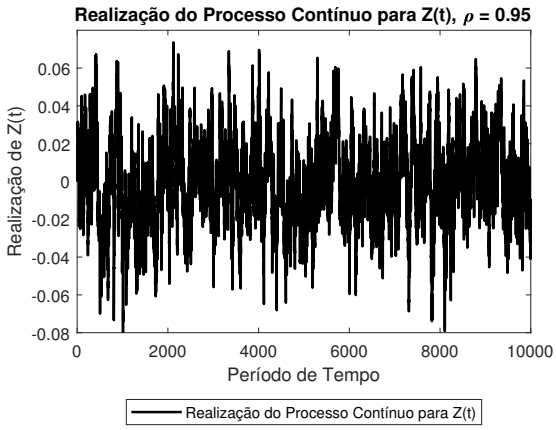
```
>> AR1D_T95(1:10)

ans =

          0         0         0         0    0.0168    0.0168    0.0168    0.0336    0.0336    0.0336

```

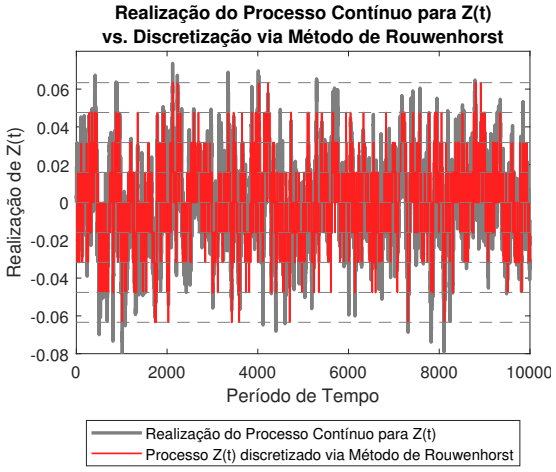
```
>> AR1D_R95(1:10)

ans =

     0     0     0     0     0     0     0     0     0     0
```

Agora, comparando os caminhos via gráfico (veja o código para gerar os gráficos abaixo no documento `lista1.m`), temos:









Note que o processo discretizado estão relativamente próximos do processo contínuo. Uma forma melhor de comparar isso graficamente seria se o exercício pedisse um T menor. Dessa forma, conseguiríamos comparar com maior precisão. ■

Questão #4

Estime processos AR(1) com base nos dados simulados, tanto a partir do Tauchen quanto o de Rouwenhorst. Quão próximo eles estão do processo gerador de dados real? Se eles não estiverem muito próximos, utilize mais pontos.

Resposta. Para estimarmos bastar utilizarmos a função `fitlm` para rodar a regressão e a função `lagmatrix` para calular o lag da variável dependente. Assim, temos:

```
% For simplicity, let's call the variables as Y1 (Tauchen) and Y2 (Rouwenhorst)
Y1 = AR1D_T95;
Y2 = AR1D_R95;

% Run the regressions
lm1_95 = fitlm(lagmatrix(Y1,1), Y1, 'Intercept', false);  % Tauchen, rho = 0.95
lm2_95 = fitlm(lagmatrix(Y2,1), Y2, 'Intercept', false);  % Rouwenhorst, rho = 0.95
```

O resultado obtido é:

```
>> lm1_95

lm1_95 =

Linear regression model:
    y ~ x1

Estimated Coefficients:
           Estimate       SE        tStat     pValue
           ________    _________    ______    ______

    x1     0.95239     0.0030497    312.29      0

Number of observations: 9999, Error degrees of freedom: 9998
Root Mean Squared Error: 0.00814

>> lm2_95

lm2_95 =

Linear regression model:
    y ~ x1

Estimated Coefficients:
           Estimate       SE        tStat     pValue
           ________    _________    ______    ______

    x1     0.95339     0.0030187    315.83      0

Number of observations: 9999, Error degrees of freedom: 9998
Root Mean Squared Error: 0.00693
```

Note que em ambas as regressões os valores do coeficiente defasado é bem próximo de 0.95. No entanto, o mais correto é considerar o erro padrão pois não estamos trabalhando com estimativa pontual e sim intervalar. Nesse caso, podemos rodar um teste de hipótese $H_0 : \rho = 0.95$ vs. $H_1 : \rho \neq 0.95$. Assim, a estatística do teste pode ser calculada como:

$$t_{\text{calc}} = \frac{\hat{\beta}_{\text{OLS}} - 0.95}{\text{ep}(\hat{\beta}_{\text{OLS}})}$$

No Matlab, podemos computá-las da seguinte maneira:

```
% Compute t-statistic
tStat_tauchen = (lm1_95.Coefficients.Estimate - 0.95)/lm1_95.Coefficients.SE;
tStat_rouwenh = (lm2_95.Coefficients.Estimate - 0.95)/lm2_95.Coefficients.SE;
```

Os resultados obtidos, serão:

```
>> tStat_tauchen

tStat_tauchen =

    0.7844

>> tStat_rouwenh

tStat_rouwenh =

    1.1215
```

Ambos menores que o valor crítico de 1.96. Logo, não rejeitamos a nula. E, portanto, devemos ter cuidado na interpretação desse resultado. Se simularmos 100 processos AR(1) e discretizá-los via método de Tauchen ou Rouwenhorst, o parâmetro verdadeiro $\rho = 0.99$ pertencerá a 95 deles. ■

Questão #5

Refaça os exercícios acima quando $\rho = 0.99$.

5.1. Discretize o processo acima usando o método de Tauchen (1986). Use 9 pontos.

Resposta. No Matlab, temos o seguinte:

```
clearvars;
clc;

% Re-calibration
rho      = 0.99;
sig      = 0.007;

N        = 9;        % Number of points (states);
m        = 3;        % Scaling parameter (I don't know why!);

% Tauchen's method
[S9T, P9T] = tauchen(m, rho, sig, N)
```

O resultado será:

```
S9T =

   -0.1489   -0.1116   -0.0744   -0.0372    0.0000    0.0372    0.0744    0.1116    0.1489


P9T =

    0.9928    0.0072    0.0000         0         0         0         0         0         0
    0.0024    0.9914    0.0062    0.0000         0         0         0         0         0
    0.0000    0.0028    0.9918    0.0054    0.0000         0         0         0         0
    0.0000    0.0000    0.0033    0.9921    0.0046    0.0000         0         0         0
    0.0000    0.0000    0.0000    0.0039    0.9921    0.0039    0.0000         0         0
    0.0000    0.0000    0.0000    0.0000    0.0046    0.9921    0.0033    0.0000         0
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0054    0.9918    0.0028    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0062    0.9914    0.0024
         0    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0072    0.9928
```

■

5.2. Discretize o processo acima usando o método de Rouwenhorst. Use 9 pontos.

Resposta. No Matlab, temos o seguinte:

```
% Re-calibration
rho      = 0.99;
sig      = 0.007;
N        = 9;        % Number of points (states);

% Rouwenhorst's method
[S9R, P9R] = rouwenhorst(rho, sig, N)
```

O resultado será:

```
S9R =

   -0.1404   -0.1053   -0.0702   -0.0351         0    0.0351    0.0702    0.1053    0.1404


P9R =

    0.9607    0.0386    0.0007    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0048    0.9609    0.0338    0.0005    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0097    0.9610    0.0290    0.0004    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0001    0.0145    0.9611    0.0241    0.0002    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0001    0.0193    0.9611    0.0193    0.0001    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0002    0.0241    0.9611    0.0145    0.0001    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0004    0.0290    0.9610    0.0097    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0005    0.0338    0.9609    0.0048
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0007    0.0386    0.9607
```

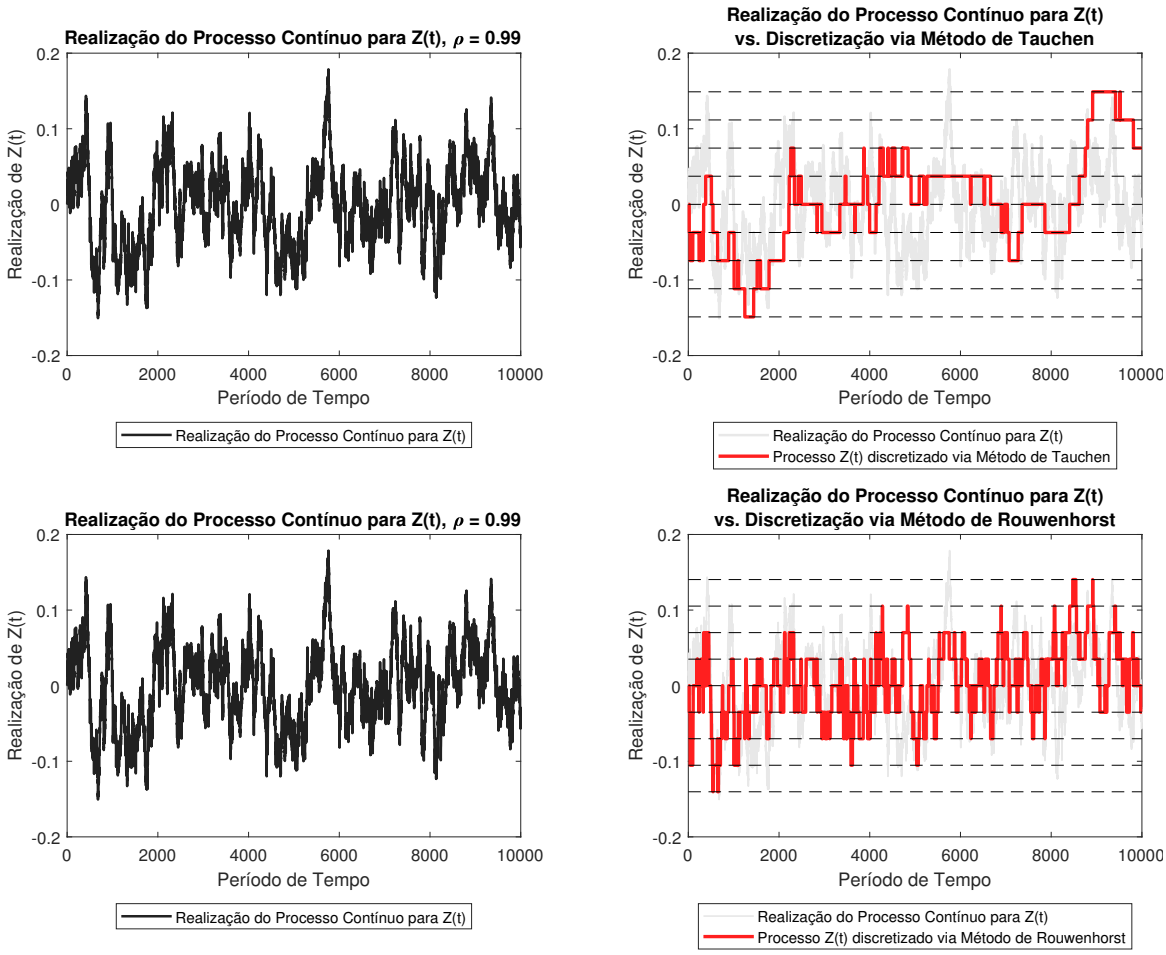
■

5.3. Simule o processo contínuo para 10000 períodos. Faça o mesmo para os processos discretizados (lembre-se de usar as mesmas realizações para os choques). Compare os caminhos para cada processo (gráficos serão úteis aqui). Se eles não estiverem muito próximos, utilize mais pontos.

Resposta. Igualmente ao item 3, temos:

```
%%%% Simulation of the AR(1) process, rho = 0.99 %%%%

% Setting seed, for replicate the results
rng(1234)

% Number of periods
T = 10000;

% Simulation AR(1) process and shocks
[Shock, Z] = ar1_simulation(rho, sig, T);

%%%% Simulation of the discretized AR(1) process, rho = 0.99 %%%%

% Initial state (median)
th0 = find(S9T == median(S9T(:)));

% Correspondents indices
IDXT = shock(th0, sig, Shock, P9T, T);
IDXR = shock(th0, sig, Shock, P9R, T);

% Discretized AR(1)
AR1D_T99 = S9T(IDXT);
AR1D_R99 = S9R(IDXR);
```

Graficamente (veja o código do gráfico no arquivo `lista1.m`), temos:



Note que quando $\rho = 0.99$, a discretização via método de Tauchen fica muito ruim, enquanto que a discretização via método de Rouwenhorst aproxima melhor o processo contínuo, conforme evidenciado por Kopecky e Suen (2010). Sendo assim, quando há uma maior persistência do choque, isto é, quando ρ se aproxima de 1, o método de Rouwenhorst torna-se mais confiável. ■

5.4. Estime processos AR(1) com base nos dados simulados, tanto a partir do Tauchen quanto o de Rouwenhorst.

Resposta. Igualmente ao item 4, temos:

```
% For simplicity, let's call the variables as Y1 (Tauchen) and Y2 (Rouwenhorst)
Y1 = AR1D_T99;
Y2 = AR1D_R99;

% Run the regressions
lm1_99 = fitlm(lagmatrix(Y1,1), Y1, 'Intercept', false);
lm2_99 = fitlm(lagmatrix(Y2,1), Y2, 'Intercept', false);

% Compute t-statistic (hyphotesis test)
tStat_tauchen = (lm1_99.Coefficients.Estimate - 0.99)/lm1_99.Coefficients.SE;
tStat_rouwenh = (lm2_99.Coefficients.Estimate - 0.99)/lm2_99.Coefficients.SE;
```

O resultado das regressões pode ser conferido abaixo:

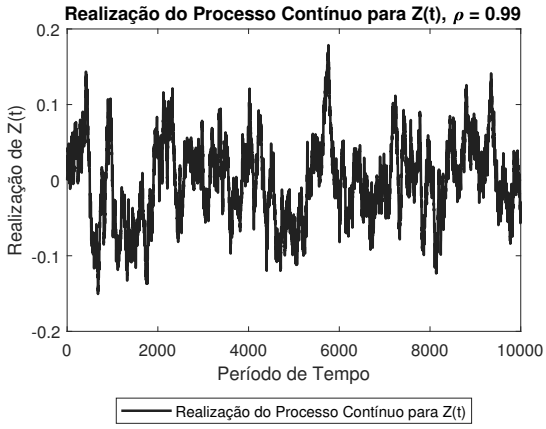
```
>> lm1_99

lm1_99 =

Linear regression model:
    y ~ x1

Estimated Coefficients:
           Estimate        SE        tStat     pValue
           ________    __________    ______    ______

    x1     0.99876     0.00051122    1953.7      0


Number of observations: 9999, Error degrees of freedom: 9998
Root Mean Squared Error: 0.00333
>> lm2_99

lm2_99 =

Linear regression model:
    y ~ x1

Estimated Coefficients:
           Estimate       SE        tStat     pValue
           ________    _________    ______    ______

    x1     0.99046     0.0013801    717.69      0


Number of observations: 9999, Error degrees of freedom: 9998
Root Mean Squared Error: 0.00676
```

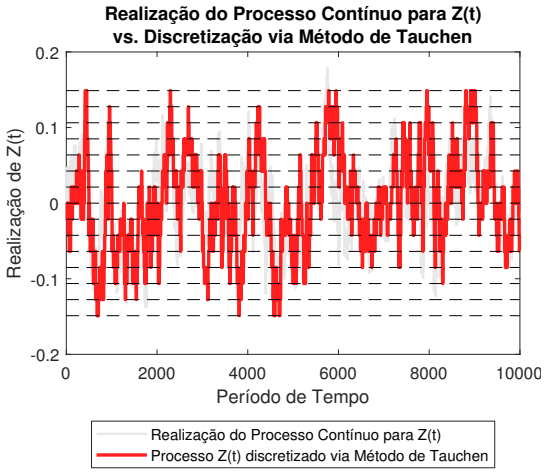
Já o resultado dos testes de hipóteses nos dizem que:

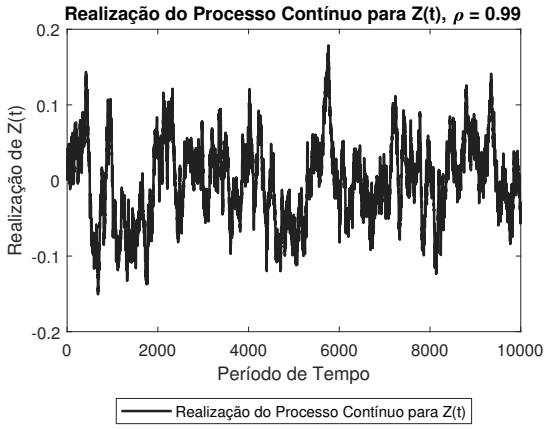
```
>> tStat_tauchen

tStat_tauchen =

   17.1319

>> tStat_rouwenh

tStat_rouwenh =

    0.3327
```

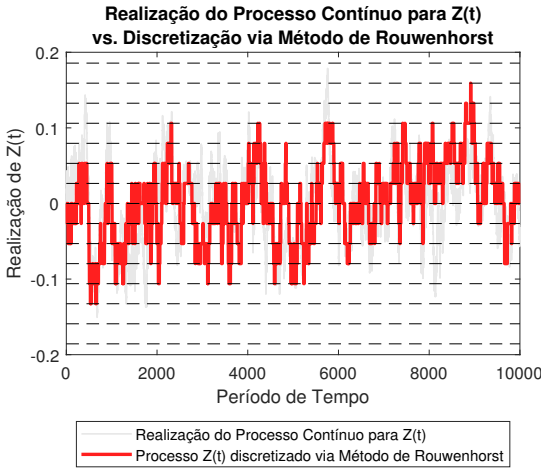
Note que o teste de hipóteses que testa a nula $H_0 : \rho = 0.99$ para o método de Tauchen rejeita a hipótese nula (`abs(tStat_tauchen)≥1.96`), enquanto que o método de Rouwenhorst não rejeita (`abs(tStat_tauchen)< 1.96`). Isso indica que ao simularmos 100 processos contínuos e discretizarmos via ambos os métodos, no método de Tauchen 95 intervalos não conterão o parâmetro ρ verdadeiro, enquanto que no método de Rouwenhorst 95 intervalos conterão o parâmetro ρ verdadeiro.

Note que se aumentarmos o grid para $N = 15$, a discretização via método de Tauchen fica bem melhor. Graficamente, temos:









Agora na regressão, a estatísitica t será:

```
>> tStat_tauchen

tStat_tauchen =

    2.5875

>> tStat_rouwenh

tStat_rouwenh =

    1.1073
```

ou seja, continuamos a rejeitar a hipótese nula com o método de Tauchen. ■

Observação

Para a primeira lista o professor pediu que fizessemos o código em duas linguagens de programação diferentes. A linguagem de programação principal escolhida será o Matlab, de modo a atender ao requisito do professor segue todos os códigos em R também. Os valores não serão exatamente os mesmos pois a função de simulação do AR(1) envolve a geração de números aleatórios.

Questão #1

Discretize o processo acima usando o método de Tauchen (1986). Use 9 pontos.

Resposta. O código a seguir propõe uma solução para o exercício.

```
library(magrittr)

# Exercise 1 -------------------------------------------------------------

## Calibration
rho = 0.95
sig = 0.007

N = 9         # Number of points (states)
m = 3         # Scaling parameter (I don't know why!)

t = 10000

## Grid and Transition matrix function of Tauchen's Method

tauchen = function(N, sig, rho, m) {

  # Grid
  grid = rep(0, N)
  grid[1] = - m * sqrt(sig^2/(1-rho^2))
  grid[N] = + m * sqrt(sig^2/(1-rho^2))
  step = (grid[N] - grid[1]) / (N - 1)
  for (i in 2:(N-1)) {
    grid[i] = grid[i-1] + step
  }

  # Transition matrix
  if (N > 1) {
    step = grid[2] - grid[1]
    P = array(0, dim = c(N, N))
    for (j in 1:N) {
      for (k in 1:N) {
        if (k == 1) {
```

```
          P[j, k] = pnorm((grid[k] - rho * grid[j] + (step / 2)) / sig)
        }
        else if (k == N) {
          P[j, k] = 1 - pnorm((grid[k] - rho * grid[j] - (step / 2)) / sig)
        }
        else {
          P[j, k] = pnorm((grid[k] - rho * grid[j] + (step / 2)) / sig) -
            pnorm((grid[k] - rho * grid[j] - (step / 2)) / sig)
        }
      }
    }
  } else {
    P = 1
  }
  return(list(zgrid = round(grid, 4), P = round(P, 4)))
}

## Grid and Transition matrix (Tauchen's Method)
Tauchen95 = tauchen(N, sig, rho, m)
Tauchen95
```

```
## $zgrid
## [1] -0.0673 -0.0504 -0.0336 -0.0168  0.0000  0.0168  0.0336  0.0504  0.0673
##
## $P
##         [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
##  [1,] 0.7644 0.2347 0.0009 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##  [2,] 0.0592 0.7405 0.1997 0.0006 0.0000 0.0000 0.0000 0.0000 0.0000
##  [3,] 0.0001 0.0747 0.7569 0.1679 0.0004 0.0000 0.0000 0.0000 0.0000
##  [4,] 0.0000 0.0001 0.0931 0.7669 0.1396 0.0002 0.0000 0.0000 0.0000
##  [5,] 0.0000 0.0000 0.0002 0.1147 0.7702 0.1147 0.0002 0.0000 0.0000
##  [6,] 0.0000 0.0000 0.0000 0.0002 0.1396 0.7669 0.0931 0.0001 0.0000
##  [7,] 0.0000 0.0000 0.0000 0.0000 0.0004 0.1679 0.7569 0.0747 0.0001
##  [8,] 0.0000 0.0000 0.0000 0.0000 0.0000 0.0006 0.1997 0.7405 0.0592
##  [9,] 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0009 0.2347 0.7644
```

Questão #2

Discretize o processo acima usando o método de Rouwenhorst. Use 9 pontos.

Resposta. O código a seguir propõe uma solução para o exercício.

```
## Grid and Transition matrix function of Rouwenhorst's Method
rouwen <- function(rho, sigma, n){

  zgrid <- seq(
```

```
    from = - (sig / sqrt(1-rho^2)) * sqrt(N-1),
    to = + (sig / sqrt(1-rho^2)) * sqrt(N-1),
    length = n
  )

  p  <- (rho+1)/2
  nu <- ((n-1)/(1-rho^2))^(1/2) * sigma
  P  <- matrix(c(p, 1 - p, 1 - p, p), nrow = 2, ncol = 2)

  for (i in 3:n){
    zeros <- matrix(0, nrow = i-1, ncol = 1)
    zzeros <- matrix(0, nrow = 1, ncol = i-1)

    P        <- p * rbind(cbind(P, zeros), cbind(zzeros, 0)) +
                (1-p) * rbind(cbind(zeros, P), cbind(0, zzeros)) +
                (1-p) * rbind(cbind(zzeros, 0), cbind(P, zeros)) +
                 p * rbind(cbind(0, zzeros), cbind(zeros, P))
  }

  for (j in 1:N){
    P[j,] = P[j,]/sum(P[j,])
  }

  return(list(zgrid = round(zgrid, 4), P = round(P, 4)))
}

## Grid and Transition matrix (Rouwenhorst's Method)
Rouwen95 <- rouwen(rho, sig, N)
Rouwen95
```

```
## $zgrid
## [1] -0.0634 -0.0476 -0.0317 -0.0159  0.0000  0.0159  0.0317  0.0476  0.0634
##
## $P
##         [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
##  [1,] 0.8167 0.1675 0.0150 0.0008 0.0000 0.0000 0.0000 0.0000 0.0000
##  [2,] 0.0209 0.8204 0.1469 0.0113 0.0005 0.0000 0.0000 0.0000 0.0000
##  [3,] 0.0005 0.0420 0.8231 0.1261 0.0081 0.0003 0.0000 0.0000 0.0000
##  [4,] 0.0000 0.0016 0.0630 0.8247 0.1051 0.0054 0.0001 0.0000 0.0000
##  [5,] 0.0000 0.0001 0.0032 0.0841 0.8253 0.0841 0.0032 0.0001 0.0000
##  [6,] 0.0000 0.0000 0.0001 0.0054 0.1051 0.8247 0.0630 0.0016 0.0000
##  [7,] 0.0000 0.0000 0.0000 0.0003 0.0081 0.1261 0.8231 0.0420 0.0005
##  [8,] 0.0000 0.0000 0.0000 0.0000 0.0005 0.0113 0.1469 0.8204 0.0209
##  [9,] 0.0000 0.0000 0.0000 0.0000 0.0000 0.0008 0.0150 0.1675 0.8167
```

Questão #3

Simule o processo contínuo para 10000 períodos. Faça o mesmo para os processos discretizados (lembre-se de usar as mesmas realizações para os choques). Compare os caminhos para cada processo (gráficos serão úteis aqui). Se eles não estiverem muito próximos, utilize mais pontos.

Resposta. O código a seguir propõe uma solução para o exercício.

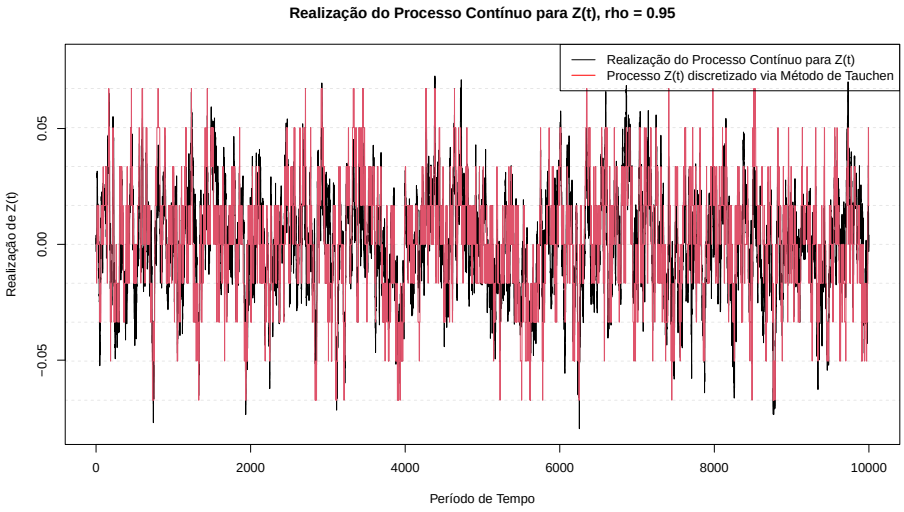
```
## Function of a AR(1) continuous process
ar1_simulation <- function(rho, sig, t){

  Z = rep(0, t); Z[1] = 0

  eps = rnorm(t, mean = 0, sd = sig)

  for(i in 2:t){
    Z[i] = rho*Z[i-1] + eps[i]
  }

  return(list(Z = round(Z, 4), eps = round(eps, 4)))
}

## Simulation of the process
## (rho = 0.95, mu = 0, sig = 0.007, t = 10000, Z[1] = 0)
set.seed(1347)
ar1_95 <- ar1_simulation(rho, sig, t)

## Show the first 10 numbers
ar1_95 %>% purrr::map(.f = ~ head(.x, 10))
```

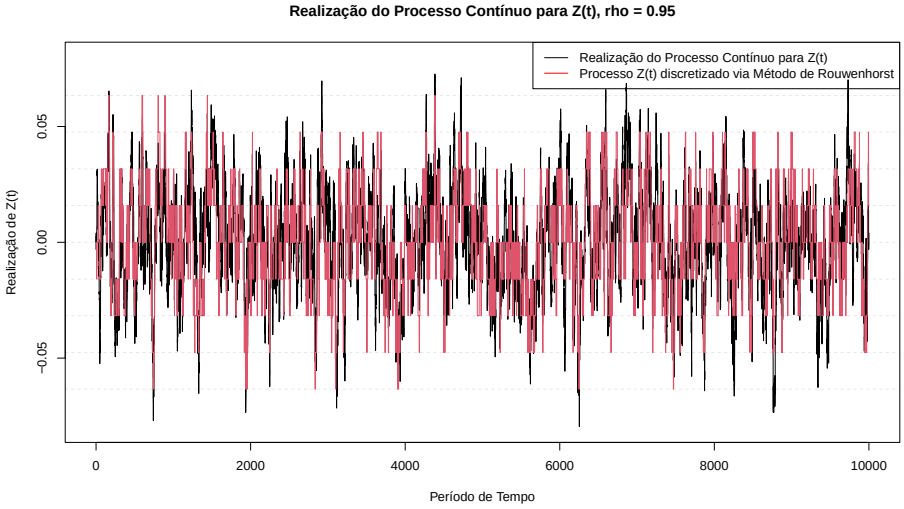
```
## $Z
##  [1]  0.0000  0.0040 -0.0032 -0.0009  0.0027 -0.0010 -0.0115 -0.0076 -0.0023
## [10]  0.0061
##
## $eps
##  [1]  0.0036  0.0040 -0.0070  0.0022  0.0035 -0.0035 -0.0106  0.0034  0.0049
## [10]  0.0083
```

```
## Function for discretize the process
discret <- function(th0, sig, eps, P, t){

  idx = rep(1, t)
  idx[1] = th0
  cum = t(apply(P, 1, cumsum))

  for(i in 2:t){
    x = which(pnorm(eps[i], mean = 0, sd = sig) <= cum[idx[i-1],])
```

```
    idx[i] = x[1]
  }

  return(idx)
}

## Tauchen's Method, rho = 0.95 ####

## Defining the initial state
th0 <- which(Tauchen95$zgrid == median(Tauchen95$zgrid))

## Returning the indices of the grid
idx = discret(th0, sig, ar1_95$eps, Tauchen95$P, t)

## Simulation the discretized process
ztauchen95 <- Tauchen95$zgrid[idx]

## Plotting
par(mfrow=c(2,1))

plot(ar1_95$Z, type = 'S', col = 1,
     main = "Realização do Processo Contínuo para Z(t), rho = 0.95",
     xlab = "Período de Tempo", ylab = "Realização de Z(t)",
     ylim = c(-0.08, 0.08))
lines(ztauchen95, col = 2)
abline(h = Tauchen95$zgrid, col = scales::alpha('black', 0.1), lty = 2)
legend("topright",
  legend = c("Realização do Processo Contínuo para Z(t)",
             "Processo Z(t) discretizado via Método de Tauchen"),
  col = c("black", "red"),
  lty = c(1, 1)
)

## Rouwenhorst's Method, rho = 0.95 ####

## Defining the initial state
th0 <- which(Rouwen95$zgrid == median(Rouwen95$zgrid))

## Returning the indices of the grid
idx = discret(th0, sig, ar1_95$eps, Rouwen95$P, t)

## Simulation the discretized process
zrouwen95 <- Rouwen95$zgrid[idx]

## Plotting
```

```r
plot(ar1_95$Z, type = 'S', col = 1,
     main = "Realização do Processo Contínuo para Z(t), rho = 0.95",
     xlab = "Período de Tempo", ylab = "Realização de Z(t)",
     ylim = c(-0.08, 0.08))
lines(zrouwen95, col = 2)
abline(h = Rouwen95$zgrid, col = scales::alpha('black', 0.1), lty = 2)
legend("topright",
       legend = c("Realização do Processo Contínuo para Z(t)",
                  "Processo Z(t) discretizado via Método de Rouwenhorst"),
       col = c("black", "red"),
       lty = c(1, 1)
)
```





Questão #4

Estime processos AR(1) com base nos dados simulados, tanto a partir do Tauchen quanto o de Rouwenhorst. Quão próximo eles estão do processo gerador de dados real? Se eles não estiverem muito próximos, utilize mais pontos.

Resposta. O código a seguir propõe uma solução para o exercício.

```
## Compute the lag (Tauchen's method)
ztauchen95_lag1 <- dplyr::lag(ztauchen95, 1)

## Run the regression and show the results
lm_tauchen95 <- lm(ztauchen95 ~ 0 + ztauchen95_lag1); broom::tidy(lm_tauchen95)
```

```
## # A tibble: 1 x 5
##   term            estimate std.error statistic p.value
##   <chr>              <dbl>     <dbl>     <dbl>   <dbl>
## 1 ztauchen95_lag1    0.946   0.00324      292.       0
```

```
## Hyphotesis test (H0: rho = 0.95 vs H1: rho \neq 0.95)
tauchen95_tstat <- (lm_tauchen95$coefficients - 0.95)/sqrt(diag(vcov(lm_tauchen95)))
tauchen95_tstat
```

```
## ztauchen95_lag1
##       -1.191776
```

```
## Confidence interval (if inside, we do not reject the null)
qt(c(.025, .975), df = lm_tauchen95$df.residual)
```

```
## [1] -1.960201  1.960201
```

```
## Compute the lag (Tauchen's method)
zrouwen95_lag1 <- dplyr::lag(zrouwen95, 1)

## Run the regression and show the results
lm_rouwen95 <- lm(zrouwen95 ~ 0 + zrouwen95_lag1); broom::tidy(lm_rouwen95)
```

```
## # A tibble: 1 x 5
##   term           estimate std.error statistic p.value
##   <chr>             <dbl>     <dbl>     <dbl>   <dbl>
## 1 zrouwen95_lag1    0.951   0.00310      307.       0
```

```
## Hyphotesis test (H0: rho = 0.95 vs H1: rho \neq 0.95)
rouwen95_tstat <- (lm_rouwen95$coefficients - 0.95)/sqrt(diag(vcov(lm_rouwen95)))
rouwen95_tstat
```

```
## zrouwen95_lag1
##      0.2725319
```

```
## Confidence interval (if inside, we do not reject the null)
qt(c(.025, .975), df = lm_rouwen95$df.residual)
```

```
## [1] -1.960201  1.960201
```

Questão #5

Refaça os exercícios acima quando $\rho = 0.99$.

Resposta. O código a seguir propõe uma solução para o exercício.

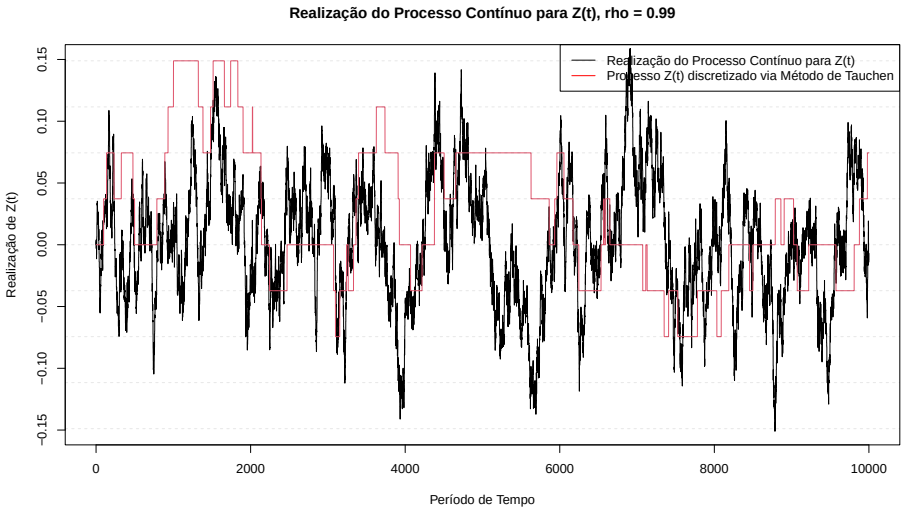
```
## Recalibration
rho = 0.99
sig = 0.007

N = 9         # Number of points (states);
m = 3         # Scaling parameter (I don't know why!);

T = 10000

## 5.1 Tauchen's Method ####
Tauchen99 <- tauchen(N, sig, rho, m)

## 5.2 Rouwenhorst's Method ####
Rouwen99 <- rouwen(rho, sig, N)

## 5.3 Process simulation and discretize ####

## Simulation of the process
## (rho = 0.99, mu = 0, sig = 0.007, T = 10000, Z[1] = 0)
set.seed(1347)
ar1_99 <- ar1_simulation(rho, sig, T)

## Show the first 10 numbers
ar1_99 %>% purrr::map(.f = ~ head(.x, 10))
```

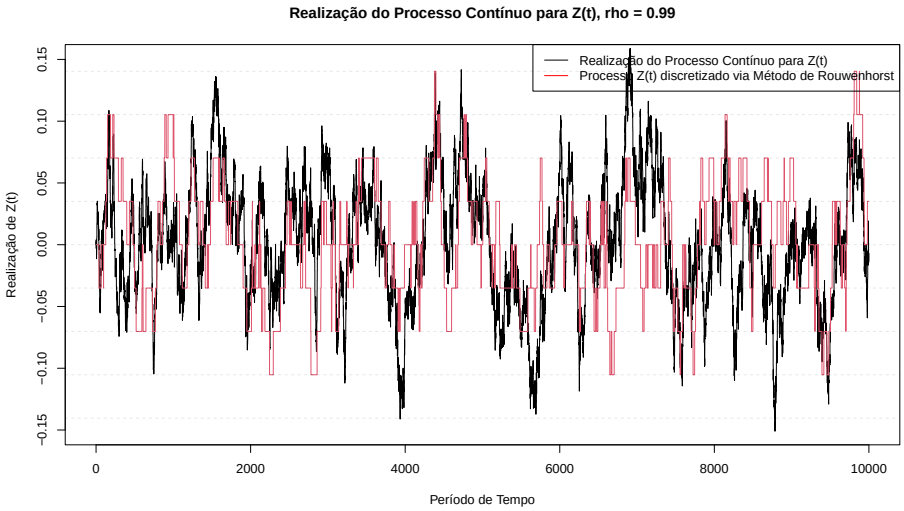
```
## $Z
##  [1]  0.0000  0.0040 -0.0031 -0.0009  0.0027 -0.0009 -0.0114 -0.0080 -0.0030
## [10]  0.0053
##
## $eps
##  [1]  0.0036  0.0040 -0.0070  0.0022  0.0035 -0.0035 -0.0106  0.0034  0.0049
## [10]  0.0083
```

```
## Tauchen's Method, rho = 0.99 ####

## Defining the initial state
th0 <- which(Tauchen99$zgrid == median(Tauchen99$zgrid))

## Returning the indices of the grid
idx = discret(th0, sig, ar1_99$eps, Tauchen99$P, T)
```

```r
## Simulation the discretized process
ztauchen99 <- Tauchen99$zgrid[idx]

## Plotting
par(mfrow = c(2, 1))

plot(ar1_99$Z, type = 'S', col = 1,
     main = "Realização do Processo Contínuo para Z(t), rho = 0.99",
     xlab = "Período de Tempo", ylab = "Realização de Z(t)",
     ylim = c(-0.15, 0.15))
lines(ztauchen99, col = 2)
abline(h = Tauchen99$zgrid, col = scales::alpha('black', 0.1), lty = 2)
legend("topright",
       legend = c("Realização do Processo Contínuo para Z(t)",
                  "Processo Z(t) discretizado via Método de Tauchen"),
       col = c("black", "red"),
       lty = c(1, 1)
)

## Rouwenhorst's Method, rho = 0.99 ####

## Defining the initial state
th0 <- which(Rouwen99$zgrid == median(Rouwen99$zgrid))

## Returning the indices of the grid
idx = discret(th0, sig, ar1_99$eps, Rouwen99$P, T)

## Simulation the discretized process
zrouwen99 <- Rouwen99$zgrid[idx]

## Plotting
plot(ar1_99$Z, type = 'S', col = 1,
     main = "Realização do Processo Contínuo para Z(t), rho = 0.99",
     xlab = "Período de Tempo", ylab = "Realização de Z(t)",
     ylim = c(-0.15, 0.15))
lines(zrouwen99, col = 2)
abline(h = Rouwen99$zgrid, col = scales::alpha('black', 0.1), lty = 2)
legend("topright",
       legend = c("Realização do Processo Contínuo para Z(t)",
                  "Processo Z(t) discretizado via Método de Rouwenhorst"),
       col = c("black", "red"),
       lty = c(1, 1), text.font = 1)
```

```r
## 5.4 Regressions ####
```


Realização do Processo Contínuo para Z(t), rho = 0.99
Realização do Processo Contínuo para Z(t)
Processo Z(t) discretizado via Método de Tauchen
Realização de Z(t)
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
2000
4000
6000
8000
10000
Período de Tempo


Realização do Processo Contínuo para Z(t), rho = 0.99
Realização do Processo Contínuo para Z(t)
Processo Z(t) discretizado via Método de Rouwenhorst
Realização de Z(t)
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
2000
4000
6000
8000
10000
Período de Tempo

```
## Compute the lag (Tauchen's method)
ztauchen99_lag1 <- dplyr::lag(ztauchen99, 1)

## Run the regression and show the results
lm_tauchen99 <- lm(ztauchen99 ~ 0 + ztauchen99_lag1)
broom::tidy(lm_tauchen99)
```

```
## # A tibble: 1 x 5
##   term            estimate std.error statistic p.value
##   <chr>              <dbl>     <dbl>     <dbl>   <dbl>
## 1 ztauchen99_lag1    0.999  0.000534     1870.       0
```

```
## Hyphotesis test (H0: rho = 0.99 vs H1: rho \neq 0.99)
tauchen99_tstat <- (lm_tauchen99$coefficients - 0.99)/sqrt(diag(vcov(lm_tauchen99)))
tauchen99_tstat %>% as.vector()
```

```
## [1] 16.19571
```

```
## Confidence interval (if inside, we do not reject the null)
qt(c(.025, .975), df = lm_tauchen99$df.residual)
```

```
## [1] -1.960201  1.960201
```

```
## Compute the lag (Tauchen's method)
zrouwen99_lag1 <- dplyr::lag(zrouwen99, 1)

## Run the regression and show the results
lm_rouwen99 <- lm(zrouwen99 ~ 0 + zrouwen99_lag1)
broom::tidy(lm_rouwen99)
```

```
## # A tibble: 1 x 5
##   term           estimate std.error statistic p.value
##   <chr>             <dbl>     <dbl>     <dbl>   <dbl>
## 1 zrouwen99_lag1    0.990   0.00142      695.       0
```

```
## Hyphotesis test (H0: rho = 0.99 vs H1: rho \neq 0.99)
rouwen99_tstat <- (lm_rouwen99$coefficients - 0.99)/sqrt(diag(vcov(lm_rouwen99)))
rouwen99_tstat %>% as.vector()
```

```
## [1] -0.1108278
```

```
## Confidence interval (if inside, we do not reject the null)
qt(c(.025, .975), df = lm_rouwen99$df.residual)
```

```
## [1] -1.960201  1.960201
```