

Lista #1

16 de maio de 2022

Aluno: Rafael Vetromille

Professor: César 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:

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

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 $i, j \in [1, 2, \dots, N]$ 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:

```

1 % tauchen.m
2 function [theta, Pi] = tauchen(m, rho, sig, N)
3
4 % Setting the grid and the transition matrix
5 theta = zeros(N,1)';
6 Pi = eye(N,N);
7
8 % Specifying the grid
9 theta(N) = + m * sqrt(sig^2/(1-rho^2));
10 theta(1) = - m * sqrt(sig^2/(1-rho^2));
11
12 % Computing the remaining (N-2) points [equidistantly distributed]
13 step = (theta(N)-theta(1))/(N-1);
14
15 for i = 2:(N-1)
16     theta(i) = theta(i-1) + step;
17 end
18
19 % Computing the transition probabilities
20 for j = 1:N
21     for k = 1:N
22         % The transition to the corner points have to be treated differently
23         if k == 1
24             Pi(j,k) = normcdf((theta(1) - rho*theta(j) + step/2) / sig);
25         % The transition to the corner points have to be treated differently
26         elseif k == N
27             Pi(j,k) = 1 - normcdf((theta(N) - rho*theta(j) - step/2) / sig);
28         else
29             % The other points will be calculated by:
30             Pi(j,k) = normcdf((theta(k) - rho*theta(j) + step/2) / sig) - ...
31                 normcdf((theta(k) - rho*theta(j) - step/2) / sig);
32         end
33     end
34 end
35 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:

```

1 %% Formal beginning of the document
2 clearvars;
3 close all;
4 clc;
5
6 %% Exercise 1.
7
8 % Calibration
9 rho      = 0.95;
10 sig      = 0.007;
11
12 N        = 9;          % Number of points (states);
13 m        = 3;          % Scaling parameter (I don't know why!);
14
15 % Tauchen's Method
16
17 % FUNCTION: [S, P] = tauchen(m, rho, sig, N)
18 %
19 % INPUTS:
20 %   m:    scalar, scaling parameter
21 %   rho:   scalar, AR-coefficient
22 %   sig:   scalar, standard deviation of innovations
23 %   n:     scalar, number of grid points for the discretized process
24 %
25 % OUTPUTS:
26 %   S:     column vector of size Nx1, contains all possible states in ascending order
27 %   P:     matrix of size NxN, contains the transition probabilities.
28 %          Rows are current state and columns future state
29
30 [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:

```

1 S9T =
2
3     -0.0673    -0.0504    -0.0336    -0.0168         0     0.0168     0.0336     0.0504     0.0673
4
5 P9T =
6
7     0.7644     0.2347     0.0009     0.0000     0.0000         0         0         0         0
8     0.0592     0.7405     0.1997     0.0006     0.0000     0.0000         0         0         0
9     0.0001     0.0747     0.7569     0.1679     0.0004     0.0000     0.0000         0         0
10    0.0000     0.0001     0.0931     0.7669     0.1396     0.0002     0.0000     0.0000         0
11    0.0000     0.0000     0.0002     0.1147     0.7702     0.1147     0.0002     0.0000         0
12    0.0000     0.0000     0.0000     0.0002     0.1396     0.7669     0.0931     0.0001     0.0000
13    0.0000     0.0000     0.0000     0.0000     0.0004     0.1679     0.7569     0.0747     0.0001
14    0.0000     0.0000     0.0000     0.0000     0.0000     0.0006     0.1997     0.7405     0.0592
15    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 $p = (1+\rho)/2$, $P_2 = [p \ 1-p; \ 1-p \ p]$ 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:

```
1 % Rouwenhorst.m
2 function [theta, Pi] = rouwenhorst(rho, sig, N)
3
4 % Specifying the grid
5 theta(N) = + (sig / sqrt(1-rho^2)) * sqrt(N-1);
6 theta(1) = - (sig / sqrt(1-rho^2)) * sqrt(N-1);
7
8 % Computing the remaining (N-2) points [equidistantly distributed]
9 step = (theta(N)-theta(1))/(N-1);
10
11 for i = 2:(N-1)
12     theta(i) = theta(i-1) + step;
13 end
14
15 % Computing p and P2
16 p = (1+rho)/2;
17 Pi = [p 1-p; 1-p p];
18
19 % Compute the transition matrix P recursively
20 for n = 3:N
21     Pi = p * [Pi zeros(n-1,1); zeros(1,n-1) zeros(1,1)] + ...
22         (1-p) * [zeros(n-1,1) Pi; zeros(1,1) zeros(1,n-1)] + ...
23         (1-p) * [zeros(1,n-1) zeros(1,1); Pi zeros(n-1,1)] + ...
24         p * [zeros(1,1) zeros(1,n-1); zeros(n-1,1) Pi];
25 end
26
27 % Normalize rows to add up to 1
28 for i = 1:N, Pi(i,:) = Pi(i,:)/sum(Pi(i,:)); end
29 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:

```

1 % Calibration
2 rho      = 0.95;
3 sig      = 0.007;
4
5 % Number of points (states)
6 N        = 9;
7
8 % Rouwenhorst's Method
9
10 % FUNCTION: [S, Pi] = rouwenhorst(m, rho, sig, N)
11 % At the end of the document.
12 %
13 % INPUTS:
14 %   rho:    scalar, AR-coefficient
15 %   sig:    scalar, standard deviation of innovations
16 %   n:      scalar, number of grid points for the discretized process
17 %
18 % OUTPUTS:
19 %   S:      column vector of size Nx1, contains all possible states in ascending order
20 %   Pi:     matrix of size NxN, contains the transition probabilities.
21 %           Rows are current state and columns future state
22
23 [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:

```

1 S9R =
2
3     -0.0634    -0.0476    -0.0317    -0.0159     0.0000     0.0159     0.0317     0.0476     0.0634
4
5 P9R =
6
7     0.8167     0.1675     0.0150     0.0008     0.0000     0.0000     0.0000     0.0000     0.0000
8     0.0209     0.8204     0.1469     0.0113     0.0005     0.0000     0.0000     0.0000     0.0000
9     0.0005     0.0420     0.8231     0.1261     0.0081     0.0003     0.0000     0.0000     0.0000
10    0.0000     0.0016     0.0630     0.8247     0.1051     0.0054     0.0001     0.0000     0.0000
11    0.0000     0.0001     0.0032     0.0841     0.8253     0.0841     0.0032     0.0001     0.0000
12    0.0000     0.0000     0.0001     0.0054     0.1051     0.8247     0.0630     0.0016     0.0000
13    0.0000     0.0000     0.0000     0.0003     0.0081     0.1261     0.8231     0.0420     0.0005
14    0.0000     0.0000     0.0000     0.0000     0.0005     0.0113     0.1469     0.8204     0.0209
15    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:

```
1 % ar1_simulation.m
2 function [eps, Z] = ar1_simulation(rho, sig, N)
3
4     Z = zeros(N,1)';
5     eps = normrnd(0, sig, N, 1)';
6
7     for i=2:N
8         Z(i) = rho*Z(i-1) + eps(i);
9     end
10 end
```

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

```
1 %%% Simulation of the AR(1) process %%%
2
3 % Setting seed, for replicate the results
4 rng(1234)
5
6 % Number of periods
7 T = 10000;
8
9 % Simulation AR(1) process and shocks
10
11 % FUNCTION: [eps, Z] = ar1_simulation(rho, sig, N)
12 % At the end of the document.
13 %
14 % INPUTS:
15 %   rho:    scalar, AR-coefficient
16 %   sig:    scalar, standard deviation of innovations
17 %   N:      scalar, number of periods of time
18 %
19 % OUTPUTS:
20 %   eps:    column vector of size Nx1, contains all shocks
21 %   Z:      column vector of size Nx1, contains all realizations of Z(t)
22
23 [eps, Z] = ar1_simulation(rho, sig, T);
```

O comando `rng(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 aloca 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:

```
1 % shock.m
2 function idx = shock(th0, sig, eps, Pi, T)
3
4 idx = ones(T,1);
5 idx(1) = th0;
6 cum = cumsum(Pi')';
7
8 for i = 2:T
9     x = find(normcdf(eps(i), 0, sig) ≤ cum(idx(i-1),:));
10    idx(i) = x(1);
11 end
12 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:

```
1 % Initial state (median)
2 th0 = find(S9T == median(S9T(:)));
3
4 % Correspondents indices
5 IDXT = shock(th0, sig, Shock, P9T, T);
6 IDXR = shock(th0, sig, Shock, P9R, T);
7
8 % Discretized AR(1)
9 AR1D_T99 = S9T(IDXT);
10 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:

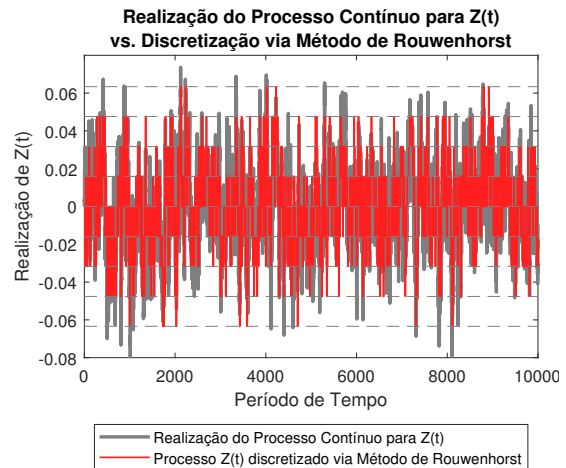
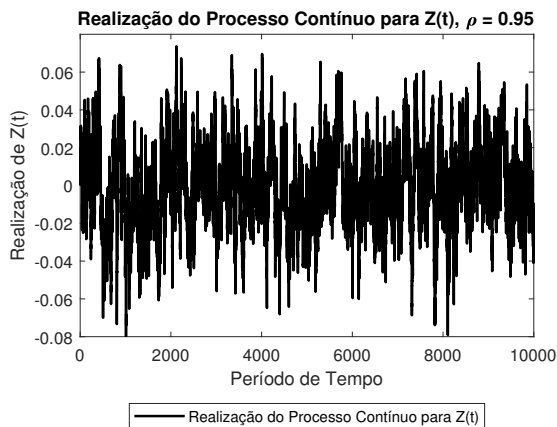
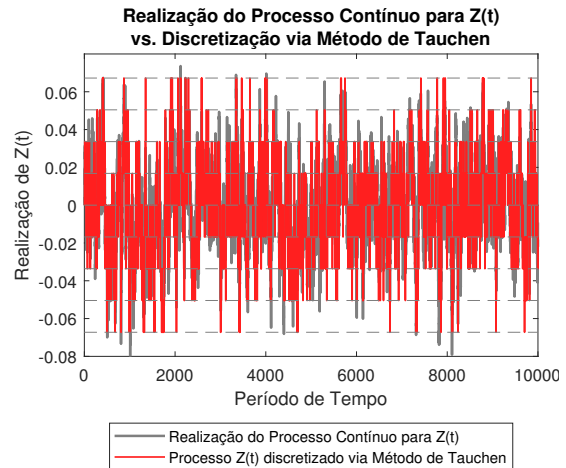
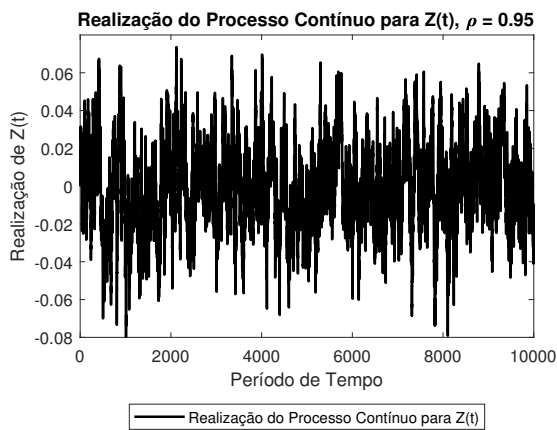
```
1 >> AR1D_T95(1:10)
2
3 ans =
4
5      0      0      0      0      0.0168      0.0168      0.0168      0.0336      0.0336      0.0336
6
```

```

7 >> AR1D_R95(1:10)
8
9 ans =
10
11      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 basta utilizarmos a função `fitlm` para rodar a regressão e a função `lagmatrix` para calcular o lag da variável dependente. Assim, temos:

```
1 % For simplicity, let's call the variables as Y1 (Tauchen) and Y2 (Rouwenhorst)
2 Y1 = AR1D_T95;
3 Y2 = AR1D_R95;
4
5 % Run the regressions
6 lm1_95 = fitlm(lagmatrix(Y1,1), Y1, 'Intercept', false); % Tauchen, rho = 0.95
7 lm2_95 = fitlm(lagmatrix(Y2,1), Y2, 'Intercept', false); % Rouwenhorst, rho = 0.95
```

O resultado obtido é:

```
1 >> lm1_95
2
3 lm1_95 =
4
5 Linear regression model:
6     y ~ x1
7
8 Estimated Coefficients:
9             Estimate      SE      tStat      pValue
10            _____  _____  _____  _____
11
12     x1      0.95239      0.0030497    312.29      0
13
14 Number of observations: 9999, Error degrees of freedom: 9998
15 Root Mean Squared Error: 0.00814
16
17 >> lm2_95
18
19 lm2_95 =
20
21 Linear regression model:
22     y ~ x1
23
24 Estimated Coefficients:
25             Estimate      SE      tStat      pValue
26            _____  _____  _____  _____
27
28     x1      0.95339      0.0030187    315.83      0
29
30 Number of observations: 9999, Error degrees of freedom: 9998
31 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:

```
1 % Compute t-statistic
2 tStat_taugchen = (lm1_95.Coefficients.Estimate - 0.95)/lm1_95.Coefficients.SE;
3 tStat_rouwenh = (lm2_95.Coefficients.Estimate - 0.95)/lm2_95.Coefficients.SE;
```

Os resultados obtidos, serão:

```
1 >> tStat_taugchen
2
3 tStat_taugchen =
4
5     0.7844
6
7 >> tStat_rouwenh
8
9 tStat_rouwenh =
10
11     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

Refça 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:

```
1 clearvars;
2 clc;
3
4 % Re-calibration
5 rho      = 0.99;
6 sig      = 0.007;
7
8 N        = 9;          % Number of points (states);
9 m        = 3;          % Scaling parameter (I don't know why!);
10
11 % Tauchen's method
12 [S9T, P9T] = tauchen(m, rho, sig, N)
```

O resultado será:

```
1 S9T =
2
3     -0.1489    -0.1116    -0.0744    -0.0372     0.0000     0.0372     0.0744     0.1116     0.1489
4
5
6 P9T =
7
8     0.9928     0.0072     0.0000         0         0         0         0         0         0
9     0.0024     0.9914     0.0062     0.0000         0         0         0         0         0
10    0.0000     0.0028     0.9918     0.0054     0.0000         0         0         0         0
11    0.0000     0.0000     0.0033     0.9921     0.0046     0.0000         0         0         0
12    0.0000     0.0000     0.0000     0.0039     0.9921     0.0039     0.0000         0         0
13    0.0000     0.0000     0.0000     0.0000     0.0046     0.9921     0.0033     0.0000         0
14    0.0000     0.0000     0.0000     0.0000     0.0000     0.0054     0.9918     0.0028     0.0000
15    0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0062     0.9914     0.0024
16         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:

```
1 % Re-calibration
2 rho      = 0.99;
3 sig      = 0.007;
4 N        = 9;          % Number of points (states);
5
6 % Rouwenhorst's method
7 [S9R, P9R] = rouwenhorst(rho, sig, N)
```

O resultado será:

```
1 S9R =
2
3     -0.1404    -0.1053    -0.0702    -0.0351         0     0.0351     0.0702     0.1053     0.1404
4
5
6 P9R =
7
8     0.9607     0.0386     0.0007     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
9     0.0048     0.9609     0.0338     0.0005     0.0000     0.0000     0.0000     0.0000     0.0000
10    0.0000     0.0097     0.9610     0.0290     0.0004     0.0000     0.0000     0.0000     0.0000
11    0.0000     0.0001     0.0145     0.9611     0.0241     0.0002     0.0000     0.0000     0.0000
12    0.0000     0.0000     0.0001     0.0193     0.9611     0.0193     0.0001     0.0000     0.0000
13    0.0000     0.0000     0.0000     0.0002     0.0241     0.9611     0.0145     0.0001     0.0000
14    0.0000     0.0000     0.0000     0.0000     0.0004     0.0290     0.9610     0.0097     0.0000
15    0.0000     0.0000     0.0000     0.0000     0.0000     0.0005     0.0338     0.9609     0.0048
16    0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0007     0.0386     0.9607
```

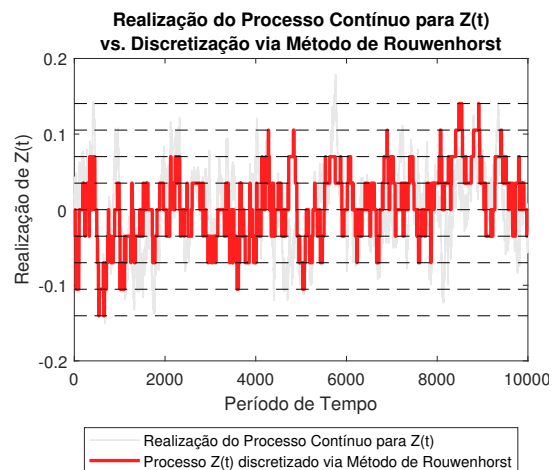
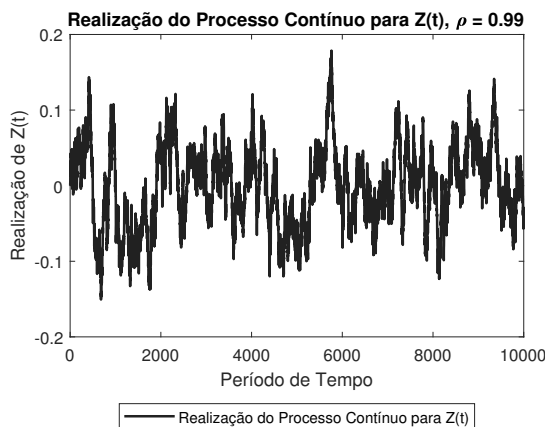
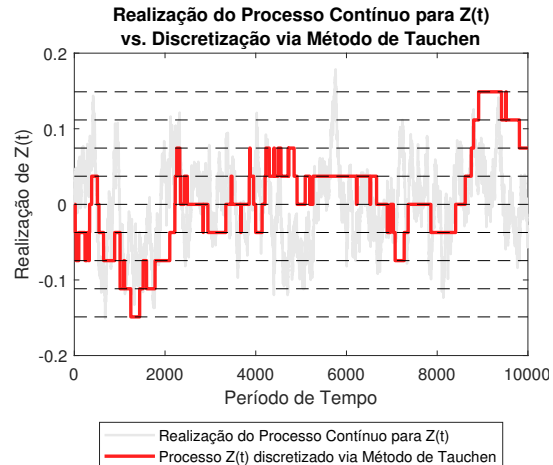
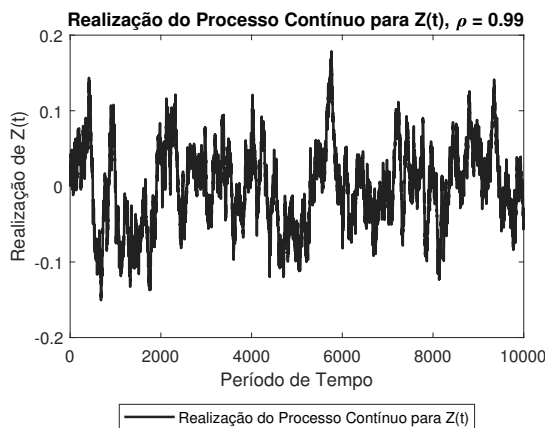


5.3. Simule o processo contínuo para 10000 períodos. Faça o mesmo para os processos discretizados (lembre-se de usar as mesmas realizações para os choques). Compare os caminhos para cada processo (gráficos serão úteis aqui). Se eles não estiverem muito próximos, utilize mais pontos.

Resposta. Igualmente ao item 3, temos:

```
1  %%% Simulation of the AR(1) process, rho = 0.99 %%%
2
3  % Setting seed, for replicate the results
4  rng(1234)
5
6  % Number of periods
7  T = 10000;
8
9  % Simulation AR(1) process and shocks
10 [Shock, Z] = ar1_simulation(rho, sig, T);
11
12 %%% Simulation of the discretized AR(1) process, rho = 0.99 %%%
13
14 % Initial state (median)
15 th0 = find(S9T == median(S9T(:)));
16
17 % Correspondents indices
18 IDXT = shock(th0, sig, Shock, P9T, T);
19 IDXR = shock(th0, sig, Shock, P9R, T);
20
21 % Discretized AR(1)
22 AR1D_T99 = S9T(IDXT);
23 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:

```
1 % For simplicity, let's call the variables as Y1 (Tauchen) and Y2 (Rouwenhorst)
2 Y1 = AR1D_T99;
3 Y2 = AR1D_R99;
4
5 % Run the regressions
6 lm1_99 = fitlm(lagmatrix(Y1,1), Y1, 'Intercept', false);
7 lm2_99 = fitlm(lagmatrix(Y2,1), Y2, 'Intercept', false);
8
9 % Compute t-statistic (hypothesis test)
10 tStat_tauchen = (lm1_99.Coefficients.Estimate - 0.99)/lm1_99.Coefficients.SE;
11 tStat_rouwenh = (lm2_99.Coefficients.Estimate - 0.99)/lm2_99.Coefficients.SE;
```

O resultado das regressões pode ser conferido abaixo:

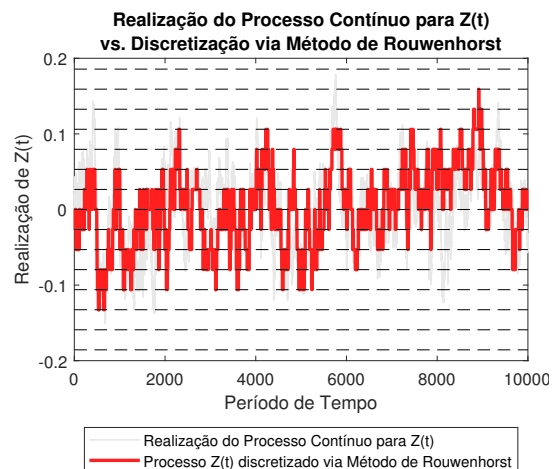
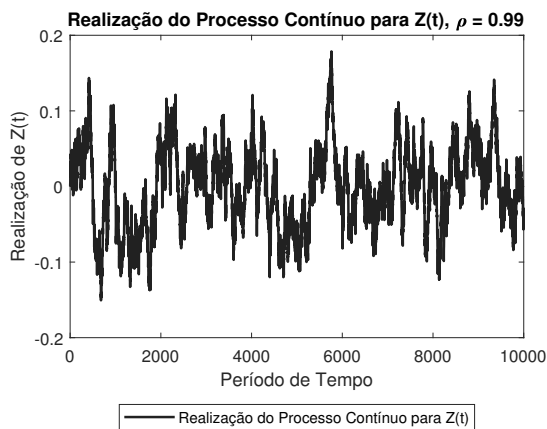
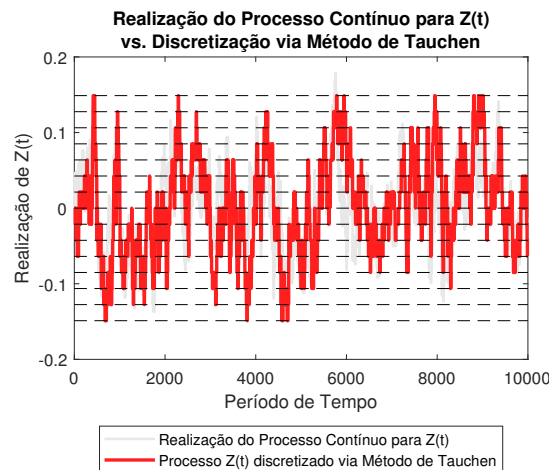
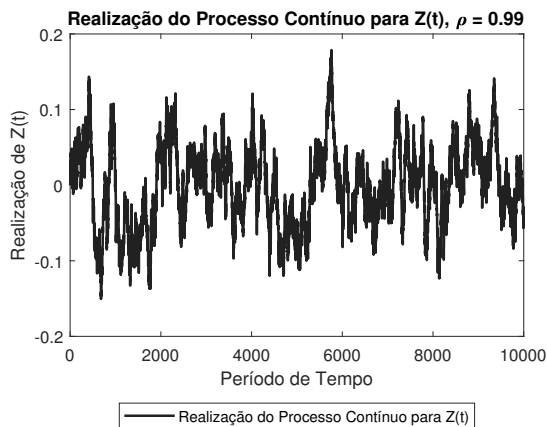
```
1 >> lm1_99
2
3 lm1_99 =
4
5 Linear regression model:
6   y ~ x1
7
8 Estimated Coefficients:
9           Estimate          SE      tStat      pValue
10          _____          _____
11
12      x1      0.99876      0.00051122    1953.7      0
13
14
15 Number of observations: 9999, Error degrees of freedom: 9998
16 Root Mean Squared Error: 0.00333
17 >> lm2_99
18
19 lm2_99 =
20
21 Linear regression model:
22   y ~ x1
23
24 Estimated Coefficients:
25           Estimate          SE      tStat      pValue
26          _____          _____
27
28      x1      0.99046      0.0013801     717.69      0
29
30
31 Number of observations: 9999, Error degrees of freedom: 9998
32 Root Mean Squared Error: 0.00676
```

Já o resultado dos testes de hipóteses nos dizem que:

```
1 >> tStat_tauschen
2
3 tStat_tauschen =
4
5     17.1319
6
7 >> tStat_rouwenh
8
9 tStat_rouwenh =
10
11     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 ($\text{abs}(tStat_tauchen) \geq 1.96$), enquanto que o método de Rouwenhorst não rejeita ($\text{abs}(tStat_tauchen) < 1.96$). 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 estatística t será:

```
1 >> tStat_tauchen
2
3 tStat_tauchen =
4
5     2.5875
6
7 >> tStat_rouwenh
8
9 tStat_rouwenh =
10
11     1.1073
```

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