Métodos Probabilísticos para a Engenharia Informática

Avaliação PL03

Rafael Santos - 98466; Gustavo Silveira - 96141

I. Introdução

As funções utilizadas em toda a prática estão localizadas num script chamado *common.m*, onde a função principal **common(func_name)** mapeia o nome das funções com suas referências.

II. Componente Prática

1. Gerador de Palavras Aleatórias

a. Matriz de Transição

Como pedido no exercício, foi gerado a matriz de transição da Cadeia de Markov, como mostrada na *Figura 1* :

```
% state(1)='r', state(2)='o', state(3)='m', state(4)='a', state(5)='.'
%   'r'     'o'    'm'      'a'    '.'
T = [
    0       1/3     0       0.25    0   % 'r'
    0.5     0       0.5     0.25    0   % 'o'
    0       1/3     0       0.25    0   % 'm'
    0.5     0       0.5     0       0   % 'a'
    0       1/3     0       0.25    0   % '.'
    ];
```

Figura 1: Matriz de Transição

Note que essa matriz também é utilizada nas outras alíneas. Assim, para se gerar os números basta gerar uma *array de estados*, terminado no quinto estado, e utilizando um conjunto de letras, mapeamos os valores do *array de estados*, exceto o quinto estado, com a posição de cada letra no conjunto. Para isso, utilizamos a função **gen_word**, definida *Figura 2* para calcularmos uma única palavra, como é descrito na *Figura 3* :

```
211  % GEN_WORD
212  % Generate a word with the transition matrix and a set_of_letters
213  % Inputs:
214  %   T              = state transition matrix
215  %   first          = initial state
216  %   last           = final or absorving state
217  %   set_of_letters  = string with the set of letters to form a word
218  % Return:
219  %   word           = word randomly generated
220  function word = gen_word(T, first, last, set_of_letters)
221      state = crawl(T, first, last);
222      state(end) = [];
223      word = set_of_letters(state);
224  end
```

Figura 2: Função para gerar uma única palavra utilizando a matriz de transição

```
% Function mapping
gen_word = common('gen_word');

first = randi(4);   % Generate random first state, help generating distinct words
last = 5;           % last state
set_of_letters = ['r' 'o' 'm' 'a'];
fprintf ("Generated word: \'%s\'\n", gen_word(T, first, last, set_of_letters)); % generate the word and print
```

Figura 3: Código escrito para gerar uma palavra aleatória

Com isso, obtivemos o seguinte resultado, como o mostrado na *Figura 4*:

```
Generated word: 'ramora'
```

Figura 4: Resultado na alínea 1a)

b. Simulação das Cinco Palavras Mais Comuns

Para esta simulação, utilizando a matriz de transição citada anteriormente, são geradas 100.000 palavras aleatórias com o auxílio da função **gen_words** que, num loop, gera **N** palavras aleatórias utilizando a função **gen_word**, explicada na alínea anterior. A *Figura 5* mostra o seu funcionamento:

```
% GEN_WORDS
% Generate a cell of words with the transition matrix and a set of letters
% Inputs:
%   T               = state transition matrix
%   first           = initial state
%   last            = final or absorving state
%   set_of_letters  = string with the set of letters to form a word
%   N               = sample space
% Return:
%   words           = cell array of reult words
function words = gen_words(T, first, last, set_of_letters, N)
    words = cell(N, 1);

    for i=1:N
        words{i} = gen_word(T, randi([ first,last-1]), last, set_of_letters);
    end
end
```

Figura 5: Função para gerar N palavras aleatórias

A partir dessa função, é gerado um *Map*, utilizando a função **gen_WordMap**, que transforma o *cell array* resultante da função **gen_words** num *categorial array* e mapeia as palavras com suas ocorrências, retornando o mapa. A *Figura 6* demostra o funcionamento desta função:

```
% GEN_WORDMAP
% Function to map each word with it's number of occurrence in the
% cell array
% Input:
%   words = cell map with 'strings'
% Return:
%   M     = map (char, double)
function M = gen_WordMap(words_cell_arr)
    if class(words_cell_arr) ~= "cell"
        return;
    end

    cat_arr = categorical(words_cell_arr);
    M = containers.Map(categories(cat_arr), countcats(cat_arr));
end
```

Figura 6: Função gen_WordMap

Para encontrar as cinco palavras com mais ocorrências, foi necessário criar uma função auxiliar, **find_maxkey**, como mostrado na *Figura 7*.

```
% Helper function to find the max value of M and return it key
% Input:
%   M       = Map(char, double)
% Return:
%   max_key = the key with max value in M
function max_key = find_maxkey(M)
    max = -1;
    for k=keys(M)
        value = M(k{1});
        if value > max
            max = value;
            max_key = k{1};
        end
    end
end
```

Figura 7: Função find_maxkey

Desta forma, encontramos a *key* com o maior valor no mapa, imprimimos sua probabiliade e a retiramos para encontrarmos as palavras restantes, resultando no seguinte:

```
Number of different words generated 18364 of 100000

5 most common words:
1) P('o') = 8.37%
2) P('a') = 6.22%
3) P('ro') = 4.09%
4) P('mo') = 4.07%
5) P('ra') = 3.24%
```

Figura 8: Resultado da simulação da alínea 1b)

c. Cálculo Teórico da Palavras mais Comuns

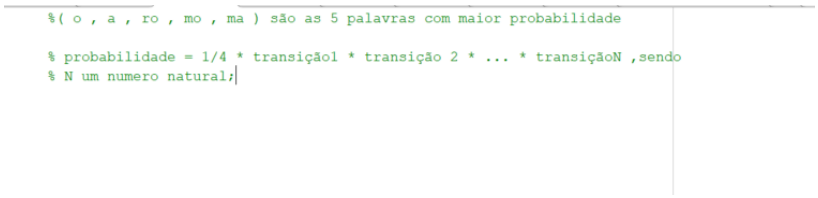


Figura 9

Sabemos da alínea anterior que **'o'**, **'a'**, **'ro'**, **'mo'** e **'ma'** são as 5 palavras com maior probabilidade de serem geradas.

Tal como era dito enunciado a probabilidade de transição de cada estado é a mesma para cada estado, no entanto uma palavra não começa pelo ponto final, portanto a probabilidade de cada letra(estado) será ¼.

Sabendo as probabilidades de transição entre estados basta fazer **¼*(transição1) *(transição2) *…*(transiçãoN)**, sendo **N** um nº natural.

Aplicando ao exercício:

- P('o') = ¼ * 1/3 * = 0.08 = 8%.
- P('a') = ¼* ¼*= 0.06 = 6%.
- P('ro') = ¼ * ½ * 1/3 = 0.0416 = 4.16%.
- P('mo') = ¼ * ½ * 1/3 = 0.0416 = 4.16%.
- P('ma') = ¼* ½ * ¼ * = 0.03125 = 3.1%.

Verificando os resultados da alínea anterior verifica-se a coerência dos resultados sendo bastante próximos do calculado em MatLab.

d. Palavras Válidas

Para importar as palavras válidas do ficheiro *wordlist-preao-20201103.txt* foi utilizado a função, **get_wordlist** que retorna um *cell array* com o conteúdo do ficheiro passado como parâmetro. Sua implementação é descrita na *Figura 10*.

```
90  % GET_WORDLIST
91  % Reads a file of valid words and put in a cell array
92  % Input:
93  %   filename = file name with the list of words
94  % Return:
95  %   wordlist = cell array with the list of words. returns empty if file
96  %              is empty or doesn't exists
97  function wordlist = get_wordlist(filename)
98      wordlist = {};
99
100     if ~isfile(filename)
101         uiwait(msgbox(sprintf('Error: File %s not found!', filename)));
102         return;
103     end
104
105     file = fopen(filename, 'r');
106     wordlist = textscan(file, '%s');
107     fclose(file);
108     wordlist = wordlist{1, 1};
109 end
```

Figura 10: Função get_wordlist

Utilizando a matriz de transição e as funções supracitadas foi gerada a lista de **N** palavras para que possamos filtrá-las utilizando a função **get_ValidWords**, descrita na *Figura 11*.

```
% Function to get the valid words from the map of generated words and
% display it's probability, if 'display' parameter is set to true
% Input:
%   WordMap     = Map containing all the unique words and it's occurrence
%   wordlist    = cell array with the valid words
%   display     = logical value to determine if it prints the probability
%   N           = sample space to calculate the probability
% Return:
%   ValidWords  = Map containing the valid words
function ValidWords = get_ValidWords(WordMap, wordlist, display, N)
    ValidWords = containers.Map('keyType', 'char', 'valueType', 'double');
    if class(display) ~= "logical"
        fprintf("\'display\' parameter must be an logical value\n");
        return;
    end

    for k=keys(WordMap)
        key = k{1};
        if ismember(key, wordlist)
            ValidWords(key) = WordMap(key);
            if display
                fprintf("P(\'%s') = %.2f%%\n", key, prob(ValidWords(key), N));
            end
        end
    end
end
```

Figura 11: Função get_ValidWords

Note que **prob(val, N)** é uma função auxiliar para o cálculo da porcentagem, também utilizada nas outras alíneas. Assim, a implementação deste problema é como mostrada na *Figura 12* :

```
% Function mapping
get_wordlist = common('get_wordlist');
gen_words = common('gen_words');
gen_WordMap = common('gen_WordMap');
get_ValidWords = common('get_ValidWords');
prob = common('prob');

wordlist = get_wordlist('wordlist-preao-20201103.txt');
if isempty(wordlist)
    return;
end

WordMap = gen_WordMap(gen_words(T, 1, length(T), ['r' 'o' 'm' 'a'], N));

asw = input('Print valid words? [Y/N]: ', 's');
ValidWords = get_ValidWords(WordMap, wordlist, ((asw == "y")||(asw == "Y")), N);

fprintf('Number of different words generated %d of %d\n', WordMap.Count, N);
fprintf('Number of valid words generated: %d\n', ValidWords.Count);
valid_values = values(ValidWords);
fprintf('Probability to generate a valid word: %.2f%%\n', prob(sum([valid_values{:}]), N));
```

Figura 12: Implementação do exercício 1d)

O resultado da simulação é demonstrado na *Figura 13* :

```
Print valid words? [Y/N]: y
P('a') = 6.31%
P('ama') = 0.77%
P('amara') = 0.10%
P('amarara') = 0.02%
P('amaro') = 0.12%
P('amo') = 1.07%
P('amora') = 0.13%
P('ao') = 2.03%
P('ara') = 0.75%
P('arama') = 0.08%
P('aramara') = 0.01%
P('aramo') = 0.12%
P('arara') = 0.11%
P('aro') = 1.02%
P('aroma') = 0.13%
P('aromara') = 0.01%
P('aromo') = 0.16%
P('ma') = 3.12%
P('mama') = 0.40%
P('mamara') = 0.05%
P('mamo') = 0.52%
P('mo') = 4.14%
P('mora') = 0.56%
P('morara') = 0.06%
P('moro') = 0.70%
P('o') = 8.34%
P('ora') = 1.05%
P('orara') = 0.13%
P('oro') = 1.40%
P('rama') = 0.41%
P('ramo') = 0.54%
P('rara') = 0.35%
P('raro') = 0.51%
Number of different words generated 18372 of 100000
Number of valid words generated: 33
Probability to generate a valid word: 35.22%
```

Figura 13: Resultados do exercício 1d)

e. Gerador com Limite de Letras

Com o objetivo de gerar palavras onde terminam no ultimo estado ou quando o seu tamanho é maior do que o prédefinido, foi necessario criar uma função **crawlN** que é uma alteração da função previamente citada onde recebe mais um parametro, que traduz o tamanho máximo do array de estados. A *Figura 14* demonstra sua implementação:

```
226  % CRAWLN
227  %
228  % Inputs:
229  %   T           = state transition matrix
230  %   first       = initial state
231  %   last        = terminal or absorving state
232  %   max_size    = max word size
233  % Return:
234  %   state       = list of generated states
235  function state = crawlN(T, first, last, n)
236      state = [first];
237      if first >= last
238          return;
239      end
240
241      while ~(n == 1)
242          state(end+1) = next_state(T, state(end));
243          if (state(end) == last) || (~(n <= 0) && (length(state) >= n))
244              break;
245          end
246      end
247  end
```

Figura 14: Função crawlN

Note que se **n** for menor ou igual a **zero** o tamanho máximo é ignorado e será gerado um *array* que termina no estado terminal. A partir dessa função foram criadas outras duas, **gen_wordN**, representada pela *Figura 15,* e **gen_wordsN**, definida pela *Figura 16,* que possúem comportamentos parecidos com **ge_word** e **gen_words**, respectivamente:

```
% GEN_WORDN
% Generate a random word with a max size using the transition matrix,
% first state and last state
% Inputs:
%   T               = state transition matrix
%   fist            = initial state
%   last            = final or absorving state
%   set_of_letters  = string with the set of letters to form a word
%   n               = max length for a word
% Return:
%   word            = random generated word
function word = gen_wordN(T, first, last, set_of_letters, n)
    state = crawlN(T, first, last, n);
    if (state(end) == last)
        state(end) = [];
    end
    word = set_of_letters(state);
end
```

Figura 15: Função gen_wordN

```
% GEN_WORDSN
% Generate a cell of words with the transition matrix and a set of letters
% with n number of characters. If n <= 0, then it has no limit
% Inputs:
%   T               = state transition matrix
%   fist            = initial state
%   last            = final or absorving state
%   set_of_letters  = string with the set of letters to form a word
%   N               = size of cell
%   n               = max size for word
% Return:
%   words           = cell array of random genetated words
function words = gen_wordsN(T, first, last, set_of_letters, N, n)
    words = cell(N, 1);

    for i=1:N
        words{i} = gen_wordN(T, randi([first last-1]), last, set_of_letters, n);
    end
end
```

Figura 16: Função gen_wordsN

Com essas alterações é possível gerar palavras com limite de tamanho.

f. Gerador de Palavras com Limite de Letras

Utilizando a função **gen_wordsN** citada na alínea anterior, foi implementado um *script*, como o mostrado na *Figura 17* :

```
% Function map
get_wordlist = common('get_wordlist');
gen_wordsN = common('gen_wordsN');
gen_WordMap = common('gen_WordMap');
get_ValidWords = common('get_ValidWords');
prob = common('prob');

wordlist = get_wordlist('wordlist-preao-20201103.txt');
if isempty(wordlist)
    return;
end

n = input('Word max size (less or equal than 0 for no limit) -> ');
WordMap = gen_WordMap(gen_wordsN(T, 1, length(T), ['r' 'o' 'm' 'a'], N, n));
asw = input('Print valid words? [Y/N]: ', 's');
ValidWords = get_ValidWords(WordMap, wordlist, ((asw == 'Y') || (asw == 'y')), N);

fprintf('Number of different words generated %d of %d\n', WordMap.Count, N);
fprintf('Number of valid words generated: %d\n', ValidWords.Count);
valid_values = values(ValidWords);
fprintf('Probability to generate a valid word: %.2f%%\n', prob(sum([valid_values{:}]), N));
```

Figura 17: Script do exercício 1f)

Note que o código é parecido com o da alínea **1d)**, com exceção da função **gen_words** que foi substituida pela função **gen_wordsN**. Após recebermos um valor lido do usuário, esse valor é utilizado para limitar o tamanho da palavra gerada por **gen_words**. Assim, como pedido no exercício, obtemos o resultado para *n = 8 (Figura 18)*, *n = 6 (Figura 19)* e *n = 4 (Figura 20)*:

```
Print valid words? [Y/N]: y
P('a') = 6.25%
P('ama') = 0.78%
P('amara') = 0.10%
P('amarara') = 0.02%
P('amararam') = 0.02%
P('amaro') = 0.14%
P('amo') = 1.04%
P('amora') = 0.13%
P('ao') = 2.10%
P('ara') = 0.79%
P('arama') = 0.10%
P('aramara') = 0.01%
P('aramaram') = 0.01%
P('aramo') = 0.14%
P('arara') = 0.10%
P('aro') = 1.09%
P('aroma') = 0.13%
P('aromara') = 0.02%
P('aromaram') = 0.01%
P('aromo') = 0.19%
P('ma') = 3.08%
P('mama') = 0.37%
P('mamara') = 0.06%
P('mamo') = 0.49%
P('mo') = 4.08%
P('mora') = 0.49%
P('morara') = 0.06%
P('moro') = 0.70%
P('o') = 8.36%
P('ora') = 1.00%
P('orara') = 0.15%
P('oro') = 1.45%
P('rama') = 0.44%
P('ramo') = 0.56%
P('rara') = 0.38%
P('raro') = 0.55%
Number of different words generated 1515 of 100000
Number of valid words generated: 36
Probability to generate a valid word: 35.39%
```

Figura 18: Resultado para n=8

```
P('ama') = 0.76%
P('amara') = 0.09%
P('amaram') = 0.09%
P('amarar') = 0.11%
P('amaro') = 0.13%
P('amo') = 1.09%
P('amora') = 0.12%
P('ao') = 2.08%
P('ara') = 0.80%
P('arama') = 0.11%
P('aramam') = 0.09%
P('aramar') = 0.11%
P('aramo') = 0.11%
P('arara') = 0.12%
P('araram') = 0.12%
P('aro') = 1.10%
P('aroma') = 0.14%
P('aromam') = 0.14%
P('aromar') = 0.13%
P('aromo') = 0.17%
P('ma') = 3.12%
P('mama') = 0.40%
P('mamara') = 0.20%
P('mamo') = 0.53%
P('mo') = 4.18%
P('mora') = 0.50%
P('morara') = 0.29%
P('moro') = 0.65%
P('o') = 8.38%
P('ora') = 0.96%
P('orara') = 0.14%
P('oraram') = 0.14%
P('oro') = 1.36%
P('rama') = 0.39%
P('ramo') = 0.54%
P('rara') = 0.38%
P('raro') = 0.51%
Number of different words generated 307 of 100000
Number of valid words generated: 38
Probability to generate a valid word: 36.47%
```

Figura 19: Resultado para n=6

```
Word max size (less or equal than 0 for no limit) -> 4
Print valid words? [Y/N]: y
P('a') = 6.19%
P('ama') = 0.78%
P('amam') = 0.80%
P('amar') = 0.83%
P('amo') = 1.00%
P('amor') = 1.05%
P('ao') = 2.08%
P('ara') = 0.77%
P('aram') = 0.80%
P('arar') = 0.78%
P('aro') = 1.08%
P('ma') = 3.11%
P('mama') = 1.54%
P('mamo') = 1.57%
P('mo') = 4.17%
P('mora') = 2.12%
P('moro') = 2.09%
P('o') = 8.21%
P('ora') = 1.08%
P('oram') = 1.07%
P('orar') = 1.10%
P('oro') = 1.45%
P('rama') = 1.54%
P('ramo') = 1.52%
P('rara') = 1.61%
P('raro') = 1.55%
Number of different words generated 61 of 100000
Number of valid words generated: 26
Probability to generate a valid word: 49.88%
```

Figura 20: Resultado para n=4

Podemos observar que, diferente das questões **b)** e **d)**, poder controlar o tamanho máximo das palavras geradas é uma grande vantagem para ter um gerador eficiente. Notamos que para o menor limite de **n (tamanho máximo do *array de estados*),** maior é a probabilidade de gerar palavras válidas, porém, maior é a quantidade de palavras repetidas, ou seja, menor é o número de palavras distintas.

2. Gerador de Palavras Limitadas em n

A matriz de transição necessária para o dercorrer deste exercício é dada pela figura 21:

```
% state(1)='r', state(2)='o', state(3)='m', state(4)='a', state(5)='.'
%   'r'     'o'    'm'      'a'   '.'
T = [
    0       0.3     0       0.3    0   % 'r'
    0.3     0       0.3     0.1    0   % 'o'
    0       0.2     0       0.2    0   % 'm'
    0.7     0       0.7     0      0   % 'a'
    0       0.5     0       0.4    0   % '.'
    ];
```

Figura 21: Matriz de Transição

O *script* para esta simulação é semelhante ao da alínea **1f)**. Logo, após corrido, obtemos os resultados para *n=0 (Figura 22) n=4 (Figura 23) n=6 (Figura 24) n=7 (Figura 25)* :

```
Word max size (less or equal than 0 for no limit) -> 0
Print valid words? [Y/N]: y
P('a') = 9.95%
P('ama') = 1.37%
P('amara') = 0.28%
P('amarara') = 0.06%
P('amaro') = 0.16%
P('amo') = 0.74%
P('amora') = 0.12%
P('ao') = 1.26%
P('ara') = 2.08%
P('arama') = 0.27%
P('aramara') = 0.06%
P('aramo') = 0.15%
P('arara') = 0.46%
P('aro') = 1.16%
P('aroma') = 0.11%
P('aromara') = 0.03%
P('aromo') = 0.07%
P('ma') = 6.84%
P('mama') = 0.95%
P('mamara') = 0.20%
P('mamo') = 0.53%
P('mo') = 3.87%
P('mora') = 0.60%
P('morara') = 0.13%
P('moro') = 0.33%
P('o') = 12.38%
P('ora') = 2.09%
P('orara') = 0.47%
P('oro') = 1.09%
P('rama') = 1.03%
P('ramo') = 0.53%
P('rara') = 1.45%
P('raro') = 0.78%
Number of different words generated 7278 of 100000
Number of valid words generated: 33
Probability to generate a valid word: 51.61%
```

Figura 22:resultados para n=0

```
Word max size (less or equal than 0 for no limit) -> 4
Print valid words? [Y/N]: y
P('a') = 10.14%
P('ama') = 1.41%
P('amam') = 0.65%
P('amar') = 1.05%
P('amo') = 0.70%
P('amor') = 0.46%
P('ao') = 1.20%
P('ara') = 2.07%
P('aram') = 1.07%
P('arar') = 1.60%
P('aro') = 1.11%
P('ma') = 7.10%
P('mama') = 2.49%
P('mamo') = 1.09%
P('mo') = 3.69%
P('mora') = 1.61%
P('moro') = 0.70%
P('o') = 12.36%
P('ora') = 2.08%
P('oram') = 1.07%
P('orar') = 1.68%
P('oro') = 1.11%
P('rama') = 2.41%
P('ramo') = 1.02%
P('rara') = 3.73%
P('raro') = 1.59%
Number of different words generated 61 of 100000
Number of valid words generated: 26
Probability to generate a valid word: 65.19%
```

Figura 23: Resultados para n=4

```
P('amara') = 0.28%
P('amaram') = 0.16%
P('amarar') = 0.22%
P('amaro') = 0.16%
P('amo') = 0.74%
P('amora') = 0.13%
P('ao') = 1.23%
P('ara') = 2.03%
P('arama') = 0.28%
P('aramam') = 0.16%
P('aramar') = 0.23%
P('aramo') = 0.17%
P('arara') = 0.41%
P('araram') = 0.23%
P('aro') = 1.13%
P('aroma') = 0.15%
P('aromam') = 0.05%
P('aromar') = 0.08%
P('aromo') = 0.08%
P('ma') = 7.07%
P('mama') = 0.93%
P('mamara') = 0.50%
P('mamo') = 0.56%
P('mo') = 3.64%
P('mora') = 0.69%
P('morara') = 0.30%
P('moro') = 0.34%
P('o') = 12.61%
P('ora') = 2.17%
P('orara') = 0.42%
P('oraram') = 0.24%
P('oro') = 1.11%
P('rama') = 0.97%
P('ramo') = 0.51%
P('rara') = 1.48%
P('raro') = 0.73%
Number of different words generated 307 of 100000
Number of valid words generated: 38
Probability to generate a valid word: 53.52%
```

Figura 24: Resultados para n=6

```
Print valid words? [Y/N]: y
P('a') = 10.10%
P('ama') = 1.43%
P('amara') = 0.27%
P('amarara') = 0.06%
P('amararam') = 0.03%
P('amaro') = 0.15%
P('amo') = 0.77%
P('amora') = 0.12%
P('ao') = 1.28%
P('ara') = 2.13%
P('arama') = 0.29%
P('aramara') = 0.07%
P('aramaram') = 0.03%
P('aramo') = 0.17%
P('arara') = 0.45%
P('aro') = 1.17%
P('aroma') = 0.12%
P('aromara') = 0.03%
P('aromaram') = 0.01%
P('aromo') = 0.06%
P('ma') = 6.98%
P('mama') = 0.98%
P('mamara') = 0.21%
P('mamo') = 0.53%
P('mo') = 3.88%
P('mora') = 0.61%
P('morara') = 0.14%
P('moro') = 0.36%
P('o') = 12.37%
P('ora') = 2.06%
P('orara') = 0.41%
P('oro') = 1.15%
P('rama') = 0.99%
P('ramo') = 0.57%
P('rara') = 1.52%
P('raro') = 0.73%
Number of different words generated 1452 of 100000
Number of valid words generated: 36
Probability to generate a valid word: 52.21%
```

Figura 25: Resultados para n=8

Podemos notar que assim como na alínea **1f)**, quanto maior o limite de tamanho do *array de estados*, ou seja, o tamanho da palavra, menor é a probabilidadede gerar palavras válidas. Porém, como a matriz de transição possui outros valores para a mudança de estados em cada letra, é possível perceber que a probabilidade é ainda maior.

3. Número de Ocorrência das Iniciais com Um Gerador de Palavras Limitadas em n

Para esta simulação, foi necessário desenvolver uma função **gen_LetterMap** que dado um *Map de palavras* válidas e um *conjunto de letras* faz o mapeamento do número de vezes em que cada letra do *conjunto de letras* é a inicial das palavras dada pelo *Map de palavras* e os retorna. A implementação desta função é descrita na *Figura 25* :

```
% GEN_LETTERMAP
%
% Input:
%   ValidWords      = Map of valid words
%   set_of_letters  = string with the set of letters to form a word
% Return:
%   LetterMap       = Map type (char, double)
function LetterMap = gen_LetterMap(ValidWords, set_of_letters)
    LetterMap = containers.Map(num2cell(set_of_letters), zeros(1, length(set_of_letters)));

    for k=keys(ValidWords)
        letter = extractBetween(k{1}, 1, 1);
        if isKey(LetterMap, letter)
            LetterMap(letter{1}) = LetterMap(letter{1}) + 1;
        end
    end
end
```

Figura 26: Função gen_LetterMap

Além desta função, foi desenvolvida uma função capaz de filtrar a lista de palavras contidas num ficheiro dado um conjunto de letras. A implementação dessa função, nomeada **filter_wordlist** é dada pela *Figura 27* :

```
% FILTER_WORDLIST
% Get the words in file that maches with set_of_letters
% Input:
%   filename        = name of file
%   set_of_letters  = string with the set of letters to form a word
% Return:
%   filtered        = cell array with the filtered words
function filtered = filter_wordlist(filename, set_of_letters)
    wordlist = get_wordlist(filename);
    filtered = {};
    for i=1:length(wordlist)
        if min(ismember(wordlist{i}, set_of_letters))
            filtered{end+1} = wordlist{i};
        end
    end
end
```

Figura 27: Função filter_wordlist

Assim, foi escrito um código capaz de obter os resultados de acordo com o pedido, como mostrado na *Figura 28* :

```
n = input('Word max size (less or equal than 0 for no limit) -> ');
asw = input('Print valid words? [Y/N]: ', 's');
set_of_letters = ['a' 'm' 'o' 'r'];
wordlist = filter_wordlist("wordlist-preao-20201103.txt", set_of_letters);
if isempty(wordlist)
    return;
end

WordMap = gen_WordMap(gen_wordsN(T, 1, length(T), set_of_letters, N, n));
ValidWords = get_ValidWords(WordMap, wordlist, ((asw == 'Y') || (asw == 'y')), N);

fprintf('Number of different words generated %d of %d\n', WordMap.Count, N);
fprintf('Number of valid words generated: %d\n', ValidWords.Count);
valid_values = values(ValidWords);
fprintf('Probability to generate a valid word: %.2f%%\n', prob(sum([valid_values{:}]), N));
LetterMap = gen_LetterMap(ValidWords, set_of_letters);

fprintf("\nProbability and number of words that begins with:\n");
for l=keys(LetterMap)
    l_val = LetterMap(l{1});
    fprintf("P(\'%s') = %.2f%%\tN = %d\n", l{1}, prob(l_val, double(ValidWords.Count)), l_val);
end
```

Figura 28: Implementação do exercicio 2

Dessa forma, obtemos os resultados para *n=0 (Figura 29) n=4 (Figura 30) n=6 (Figura31) n=8 (Figura 32)* :

```
Word max size (less or equal than 0 for no limit) -> 0
Print valid words? [Y/N]: n
Number of different words generated 7304 of 100000
Number of valid words generated: 30
Probability to generate a valid word: 18.27%

Probability and number of words that begins with:
P('a') = 63.33% N = 19
P('m') = 26.67% N = 8
P('o') = 10.00% N = 3
P('r') = 0.00%  N = 0
```

Figura 29: Resultado para n=0

```
Word max size (less or equal than 0 for no limit) -> 4
Print valid words? [Y/N]: n
Number of different words generated 61 of 100000
Number of valid words generated: 20
Probability to generate a valid word: 31.43%

Probability and number of words that begins with:
P('a') = 40.00% N = 8
P('m') = 30.00% N = 6
P('o') = 10.00% N = 2
P('r') = 20.00% N = 4
```

Figura 30: Resultado para n=4

```
Word max size (less or equal than 0 for no limit) -> 6
Print valid words? [Y/N]: n
Number of different words generated 307 of 100000
Number of valid words generated: 27
Probability to generate a valid word: 19.94%

Probability and number of words that begins with:
P('a') = 59.26% N = 16
P('m') = 29.63% N = 8
P('o') = 11.11% N = 3
P('r') = 0.00%  N = 0
```

Figura 31: Resultado para n=6

```
Number of different words generated 1482 of 100000
Number of valid words generated: 30
Probability to generate a valid word: 18.25%

Probability and number of words that begins with:
P('a') = 63.33% N = 19
P('m') = 26.67% N = 8
P('o') = 10.00% N = 3
P('r') = 0.00%  N = 0
```

Figura 32: Resultado para n=8

4. Gerador de Palavras Limitadas em n

Aplicada a matriz correspondente do exercício 4 ao exercício 3 é possível concluir que, como é possível ver pelas Figuras, o gerador do exercício 3 não é tão eficiente quanto o do exercício 4. Isto acontece, pois, a probabilidade de gerar uma palavra valida em português no ex4 é superior à do gerador do ex3.

```
Word max size (less or equal than 0 for no limit) -> 0
Print valid words? [Y/N]: y
P('amam') = 0.36%
P('amar') = 0.61%
P('amaram') = 0.05%
P('amarar') = 0.13%
P('amararam') = 0.01%
P('amor') = 0.41%
P('ar') = 7.00%
P('aram') = 0.76%
P('aramam') = 0.06%
P('aramar') = 0.13%
P('aramaram') = 0.02%
P('arar') = 1.47%
P('araram') = 0.18%
P('armam') = 0.09%
P('armar') = 0.15%
P('armaram') = 0.03%
P('aromam') = 0.04%
P('aromar') = 0.10%
P('aromaram') = 0.01%
P('mamam') = 0.08%
P('mamar') = 0.22%
P('mamaram') = 0.03%
P('mar') = 2.10%
P('mor') = 1.41%
P('moram') = 0.14%
P('morar') = 0.27%
P('moraram') = 0.05%
P('oram') = 0.82%
P('orar') = 1.45%
P('oraram') = 0.17%
Number of different words generated 7351 of 100000
Number of valid words generated: 30
Probability to generate a valid word: 18.34%

Probability and number of words that begins with:
P('a') = 63.33% N = 19
P('m') = 26.67% N = 8
P('o') = 10.00% N = 3
P('r') = 0.00%  N = 0
```

Figura 33: Resultado para n=0

```
Command Window
  P('amarara') = 0.06%
  P('amaro') = 0.17%
  P('amo') = 0.78%
  P('amora') = 0.13%
  P('ao') = 1.26%
  P('ara') = 1.77%
  P('arama') = 0.25%
  P('aramara') = 0.05%
  P('aramo') = 0.12%
  P('arara') = 0.33%
  P('arma') = 0.25%
  P('armara') = 0.03%
  P('armo') = 0.11%
  P('aro') = 1.22%
  P('aroma') = 0.13%
  P('aromara') = 0.03%
  P('aromo') = 0.08%
  P('ma') = 6.98%
  P('mama') = 0.99%
  P('mamara') = 0.19%
  P('mamo') = 0.51%
  P('mo') = 3.76%
  P('mora') = 0.54%
  P('morara') = 0.11%
  P('moro') = 0.35%
  P('o') = 12.38%
  P('ora') = 1.75%
  P('orara') = 0.33%
  P('oro') = 1.13%
  P('rama') = 0.79%
  P('ramo') = 0.43%
  P('rara') = 1.08%
  P('raro') = 0.64%
  Number of different words generated 9049 of 100000
  Number of valid words generated: 36
  Probability to generate a valid word: 50.32%
  Probability and number of words that begins with:
  P('a') = 55.56% N = 20
  P('m') = 22.22% N = 8
  P('o') = 11.11% N = 4
  P('r') = 11.11% N = 4
fx >>
```

```
Command Window
  Word max size (less or equal than 0 for no limit) -> 3
  Print valid words? [Y/N]: y
  P('ama') = 2.25%
  P('amo') = 1.52%
  P('ar') = 7.03%
  P('ara') = 5.16%
  P('aro') = 3.66%
  P('mar') = 5.28%
  P('mor') = 3.56%
  P('ora') = 5.29%
  P('oro') = 3.42%
  Number of different words generated 27 of 100000
  Number of valid words generated: 9
  Probability to generate a valid word: 37.18%

  Probability and number of words that begins with:
  P('a') = 55.56% N = 5
  P('m') = 22.22% N = 2
  P('o') = 22.22% N = 2
  P('r') = 0.00%  N = 0
fx >>
```

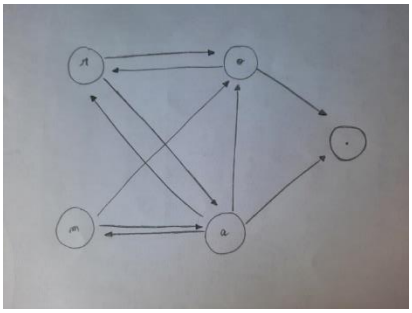
Figura 34: Resultado para n=3

Verifica-se também que os geradores não geram palavras cuja 1º letra seja ‘r’ quando o tamanho máximo da palavra seja < 4.

5. Probabilidade do Gerador

Para a resolução do ex.5 utilizamos o ex3 como suporte. Como realizado antes, o ex3 utiliza as palavras do ficheiro dado que possuem apenas as letras ‘a’, ‘r’, ‘o’ e ‘m’ e calcula a probabilidade. Ora para este exercício pegamos apenas nas palavras geradas e deixamos as probabilidades de lado.

Através das palavras geradas e válidas foi gerado o grafo representado em baixo.



```
% state(1)='r', state(2)='o', state(3)='m', state(4)='a', state(5)='.'
%   'r'    'o'     'm'     'a'    '.'
T = [
    0      1/2     0       1/4     0
    1/2    0       1/2     1/4     0
    0      0       0       1/4     0
    1/2    0       1/2     0       0
    0      1/2     0       1/4     0
    ];
```

III. Análise dos Resultados

Para fazer a comparação entre os geradores estamos a assumir que o gerador é mais eficiente quanto maior for a probabilidade de gerar palavras válidas em português.

Analisando e comparando o gerador do ex5, com os geradores dos exercícios anteriores, concluímos que os geradores mais eficientes são os do ex2 e ex4 com uma percentagem de palavras válidas em português de aproximadamente 51% e 50 %, respetivamente, enquanto que o ex3 e ex5 apresentam percentagem de aproximadamente 18% e 12 %, respetivamente.