Resumo Prático MPEI

Capítulo 2

Parte 1

```
p = 0.5; % rate ter filho
N = 1e5; % experiencias
k = 2;   % filhos por familia

familias = rand(k, N);
rapazes = familias > 0.5;
numRapazes = sum(rapazes) >= 1;
numRapazes = sum(numRapazes) / N;
fprintf("A) %f%%", numRapazes * 100);
```

Devemos definir um N para numero de experiências. Vai ser o numero de colunas.
No fim temos quer fazer sum(o que queremos) / N.
Fazer sempre P(A), P(B)
Saber que P(A|B) = P(A∩B) / P(B).
Lembrar funçao unique, que devolve valores unicos num vetor.
matriz(:,1) dá nos a primeira coluna.
~= é !=.

Parte 2

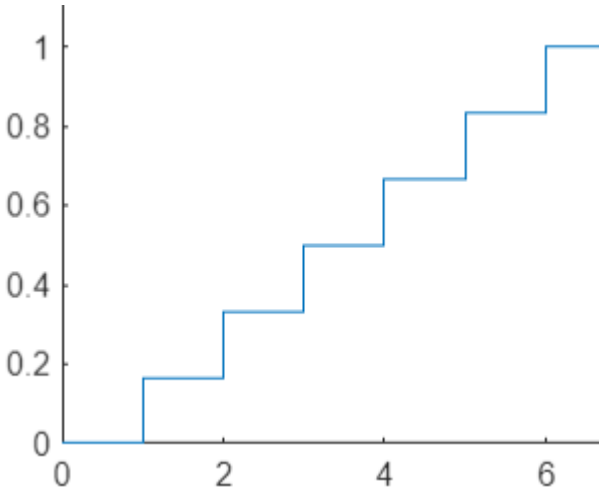
Função massa de probabilidade

```
xi = 1:6;
px=ones(1,6)/6;
stem(xi,px), xlabel('x'), ylabel('px(x)'), title('Funcao massa de
probabilidade');
```

`xi` sao os outcomes possiveis
`px` sao as probabilidades para cada outcome, respetivamente. Logo, o size de cada um tem que ser igual.

Função de distribuição acumulada

```
prob_cum = cumsum(px);
```



isto pega em cada elemento no array de probabilidade e faz uma soma cumulativa.
`px = [0.166667, 0.166667, 0.166667, 0.166667, 0.166667, 0.166667]`.

Capítulo 3

- Prob faltar à proxima aula, sabendo que ja faltou à ultima - 0.2%.
- Prob faltar à proxima aula, sabendo que nao faltou à ultima - 0.3%.
- Prob nao faltar à proxima aula, sabendo que ja faltou à ultima - 0.8%.
- Prob nao faltar à proxima aula, sabendo que nao faltou à ultima - 0.7%.

Faz-se a Matriz M

```
M = [0.7 0.8
    0.3 0.2];
```

As colunas têm todas que dar 1. Acontecendo isto, chama-se matriz estocastica.

Digamos que nao faltou a uma aula. Depois dessa, acontecem duas aulas. Qual a prob de estar presente nessa última?

```
x = [1 0]';
p = M^2*x;
disp(p(1,1));
```

Antes de aplicarmos este `x` à matriz, primeiro fazemos o M^2, para obter as probabilidades na ultima aula. Como sabemos que ele foi à primeira, aplicamos o x à matriz, que devolve o vetor `[x y]`. x é a prob de nao faltar sabendo que nao faltou e y é a prob de nao faltar sabendo que faltou. Queremos o `x`.

Probabilidades limite

```
% para calcular isto, é preciso fazer (matriz identidade - Q)^-1

M = [M - eye(size(M)); ones(1,4)];
```

```
x = [0 0 0 0 1]'; % ou [zeros(4,1); 1]
p = M\x;
fprintf("P(A) = %.5f\tP(B) = %.5f\tP(C) = %.5f\tP(D) = %.5f\n", p);

% valores iguais, calcula a probabilidade limite teorica
```

`eye(size(M))` cria uma matriz identidade com size M. Digamos que M é de size 4.
Fica do tipo:

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Com o comando `M = [M - eye(size(M)); ones(1,4)];`, M fica com uma linha final extra, feitas de 1's.

Depois fazendo a divisao de matrizes `M\x`, ficamos com as probabilidades limite de A,B,C e D em M, ou seja, fica um vetor de size M.

Misc

No problema do weather:

```
% sol nuvens chuva
% sol
% nuvens
% chuva

T = [0.7 0.2 0.3
    0.2 0.3 0.3
    0.1 0.5 0.4]
```

Se tivermos que calcular chances em dias seguidos temos que fazer:

```
x = [1 0 0]';
x2=M*x;
x3=M*x;
p = x(1) * x2(1) * x3(1)
```

Isto calcula a chance de ser sol dia 2 e dia 3, sabendo que no dia 1 foi sol.

Nao se pode fazer `p = T^2*x;` porque estamos a dar disregard aos dias do meio (assumindo que qualquer outcome pode acontecer nesses dias) e apenas a pensar no dia final.
Dá `0.7 * 0.7`, porque o 0.7 do dia 1 não conta. Sabemos que ja aconteceu.

Probabilidade de nao chover no dia 2 nem no dia 3, sabendo que o dia 1 é sol.

```
x2 = T*x;
w = x2/(1-x2(3)); % no segundo dia so pode estar sol ou nuvens, logo indice 1 ou
2, 0.7 ou 0.2
x3 = T*w;
p = (1-x2(3)) * (x3(1)+x3(2))
```

Matriz Q

Matriz inicial é

```
%   1 2 3 4 5
%  1
%  2
%  3
%  4
%  5

T = [0.8 0   0  0.3 0
     0.2 0.6 0  0.2 0
     0   0.3 1  0   0
     0   0.1 0  0.4 0
     0   0   0  0.1 1]
```

Os estados 3 e 5 sao absorventes, porque so ha chance de voltarem para eles mesmos.

Para fazer a matriz Q, metemos isto em forma canonica. Troca-se o lugar da coluna 4 pela 3, e ficam os 2 estados absorventes num canto.

```
aux = [0.8 0   0.3  0 0
       0.2 0.6 0.2  0 0
       0   0.1 0.4  0 0
       0   0.3 0    1 0
       0   0   0.1  0 1];
```

Com isto assim, `Q = aux(1:3, 1:3)`. Raciocínio:

```
aux = [0.8 0   0.3 | 0 0
       0.2 0.6 0.2 | 0 0
       0   0.1 0.4 | 0 0
       ___ ___ ___ ___ ___
       0   0.3 0   | 1 0
       0   0   0.1 | 0 1];
```

Matriz fundamental F

`F = (eye(3) - Q)^-1`

Tem que ser a Q.

Média (Valor esperado) do nº de passos antes de absorção.

Qual a média comecando no estado 1? E comecando no estado 2? E se comecando no estado 4?

Para estado 1:

```
x1 = [1 0 0]';
media = sum(F*x1);
```

Para o estado 2:

```
x2 = [0 1 0]';
media = sum(F*x2);
```

Para o estado 4:

```
x4 = [0 0 1]';
media = sum(F*x4);
```

Lembrar que a coluna 4 tinha trocado com a 3 na canonica.

Probabilidade de absorção

Neste exercicio, o estados 3 e 5 sao estados absorventes.

```
R = aux(4:5, 1:3); % slide 86
B = R*F; %slide 102 coluna comecar em estado transitorio e linha acabar no estado absovente
fprintf('Prob(estado 3) = %f, Prob(estado 5) = %f',B(1,1),B(2,1));
```

O R pega nas linhas 4 e 5 das colunas 1 a 3. Isto é o lado de baixo do que ficou de fora para a matriz Q.

Pagerank

Temos tipo 6 paginas, a prob de ir para cada pagina é equivalente. Fica-se com esta matriz:

```
H = [0 0   0 0 1/3 0
     1 0   0 0 1/3 0
     0 0.5 0 1 0   0
     0 0   1 0 0   0
     0 0.5 0 0 0   0
     0 0   0 0 1/3 0];
```

Faz-se o vetor de probabilidade para cada pagina (equivalente, por isso 1/6 pa todos). Para calcular o pagerank ao fim de 10 iterações, faz-se.

```
x = [1/6 1/6 1/6 1/6 1/6 1/6]';
p = H^10*x;
```

p fica com 6 linhas e 1 coluna

```
0.0012
0.0033
0.3315
0.3304
0.0027
0.0012
```

Podemos ver que a 3a pagina fica com maior pagerank.

Spider trap

Spider trap é quando ha loops, por exemplo C aponta apenas para D e D aponta apenas para C. Na matriz, podemos ver que o estado 3 tem prob 1 para ir para 4 e 4 tem prob 1 para ir para 3.

Como resolver spider traps:

No enunciado devem dar o valor de β.
assuma β = 0.8.

1. multiplica-se β pela matriz.
2. multiplica-se 1-β por um vetor de size(matriz), cujos elementos têm valor equivalente (estocastico).
3. Adiciona-se o valor em 1 e em 2.

No problema ficava assim:

`A = 0.8 * H + 0.2 * ones(6)/6`

Dead-end

Na matriz, pode-se ver que o estado 6 é um dead end, porque nao vai para lado nenhum.

Como resolver dead-ends:

1. Identificar todos os estados dead-ends.
2. Colocar as probabilidades outgoing desse estado equivalentes nao nulas.

Ou seja, o estado 6 de H, em vez de ser 0 0 0 0 0 0, fica 1/6 1/6 1/6 1/6 1/6 1/6.
`H(:,6) = 1/6;`

Capítulo 4

Key Gen

Gerar chaves é pegar no que ja ta feito. meter as chaves num cellarray tbm é facil.

Hash

Pegar numa key, meter numa hash function. Essa hash function vai devolver um inteiro, provavelmente muito grande. Esse inteiro vai nos servir como indice.

Digamos que temos um vetor de size 10000. Se a hash function devolve 351, colocamos esse indice a 1. Mas se devolve 54234, precisamos de fazer o mod `mod(string2hash(chaves{j}, 'djb2'), length(vectorTables))`.

Se o indice ja tiver um 1, e as keys sao todas diferentes, é porque ha colisao.

Bloom filters

Bloom filter é basicamente o que descrevi no hash.

Temos um vetor de size 10000, com todos os elementos inicializados a 0.
Temos 1 palavra. Fazemos hash dessa palavra e devolve 1 numero(indice).
Colocamos o elemento correspondente a esse indice a 1.
Normalmente, um bloom filter usa mais do que uma função. Então, faz-se este processo para mais algumas funçoes.

Para verificar a existencia de um elemento, faz-se o hash do elemento e verifica-se o elemento correspondente ao indice. Se, em alguma função, nos dá 0, sabemos com certeza que o elemento nao existe.
Se nos dá sempre 1, não temos a certeza se existe, pode haver falsos positivos, especialmente se o vetor for muito pequeno, ou se usarmos muitas ou poucas hash functions. O **sweet spot é *5***.

As grandes **vantagens** de um filtro de Bloom são a **rapidez e taxa de erro**.

Árvores e heaps são melhores soluções para conjuntos pequenos.

Counting bloom filters

- Permite ver quantas colisoes aconteceram num dado indice, portanto os elementos ja nao sao apenas 0 ou 1.
- Permite remoção de elementos.

Portanto, na inserção, o contador é incrementado. Na remoção, é decrementado.

Uma vez que existe remoção, introduz a possibilidade de falsos negativos.

Ocupa mais espaço que o bloom filter normal, 3 a 4 vezes mais.

MinHash

bruh